openstax™

Introduction to Computer Science

Introduction to Computer Science

SENIOR CONTRIBUTING AUTHOR

DR. JEAN-CLAUDE FRANCHITTI, NYU COURANT INSTITUTE



openstax™

OpenStax
Rice University
6100 Main Street MS-375
Houston, Texas 77005

To learn more about OpenStax, visit https://openstax.org.
Individual print copies and bulk orders can be purchased through our website.

©2026 Rice University. Textbook content produced by OpenStax is licensed under a Creative Commons Attribution Non-Commercial ShareAlike 4.0 International License (CC BY-NC-SA 4.0). Under this license, any user of this textbook or the textbook contents herein can share, remix, and build upon the content for noncommercial purposes only. Any adaptations must be shared under the same type of license. In any case of sharing the original or adapted material, whether in whole or in part, the user must provide proper attribution as follows:

- If you noncommercially redistribute this textbook in a digital format (including but not limited to PDF and HTML), then you must retain on every page the following attribution:
  "Access for free at openstax.org."
- If you noncommercially redistribute this textbook in a print format, then you must include on every physical page the following attribution:
  "Access for free at openstax.org."
- If you noncommercially redistribute part of this textbook, then you must retain in every digital format page view (including but not limited to PDF and HTML) and on every physical printed page the following attribution:
  "Access for free at openstax.org."
- If you use this textbook as a bibliographic reference, please include https://openstax.org/details/books/introduction-computer-science in your citation.

For questions regarding this licensing, please contact support@openstax.org.

Trademarks
The OpenStax name, OpenStax logo, OpenStax book covers, OpenStax CNX name, OpenStax CNX logo, OpenStax Tutor name, Openstax Tutor logo, Connexions name, Connexions logo, Rice University name, and Rice University logo are not subject to the license and may not be reproduced without the prior and express written consent of Rice University.

Kendall Hunt and the Kendall Hunt Logo are trademarks of Kendall Hunt. The Kendall Hunt mark is registered in the United States, Canada, and the European Union. These trademarks may not be used without the prior and express written consent of Kendall Hunt.

COLOR PAPERBACK BOOK ISBN-13	**978-1-711471-83-9**
B&W PAPERBACK BOOK ISBN-13	**978-1-711471-82-2**
DIGITAL VERSION ISBN-13	**978-1-961584-58-7**
ORIGINAL PUBLICATION YEAR	**2024**

2 3 4 5 6 7 8 9 10 CJP 26 24

OpenStax

OpenStax provides free, peer-reviewed, openly licensed textbooks for introductory college and Advanced Placement® courses and low-cost, personalized courseware that helps students learn. A nonprofit ed tech initiative based at Rice University, we're committed to helping students access the tools they need to complete their courses and meet their educational goals.

Rice University

OpenStax is an initiative of Rice University. As a leading research university with a distinctive commitment to undergraduate education, Rice University aspires to path-breaking research, unsurpassed teaching, and contributions to the betterment of our world. It seeks to fulfill this mission by cultivating a diverse community of learning and discovery that produces leaders across the spectrum of human endeavor.



Philanthropic Support

OpenStax is grateful for the generous philanthropic partners who advance our mission to improve educational access and learning for everyone. To see the impact of our supporter community and our most updated list of partners, please visit openstax.org/foundation.

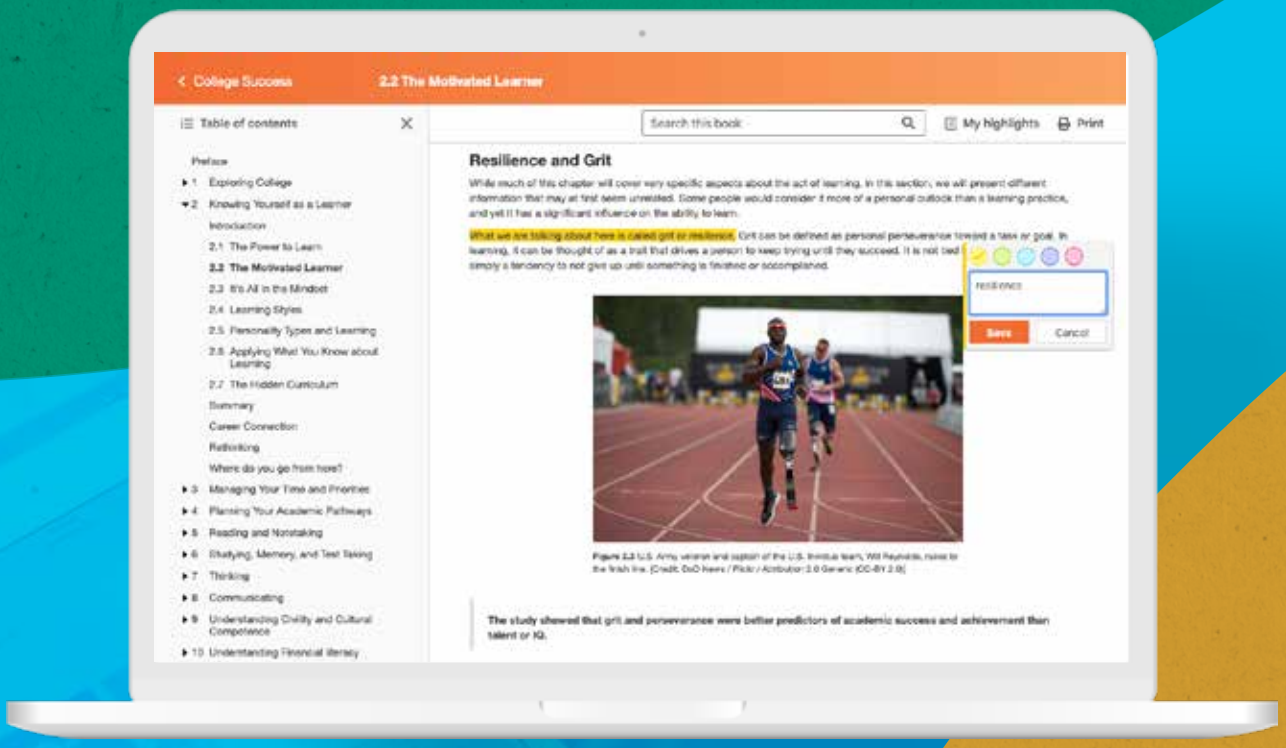
Arnold Ventures
Chan Zuckerberg Initiative
Chegg, Inc.
Arthur and Carlyse Ciocca Charitable Foundation
Digital Promise
Ann and John Doerr
Bill & Melinda Gates Foundation
Girard Foundation
Google Inc.
The William and Flora Hewlett Foundation
The Hewlett-Packard Company
Intel Inc.
Rusty and John Jaggers
The Calvin K. Kazanjian Economics Foundation
Charles Koch Foundation
Leon Lowenstein Foundation, Inc.
The Maxfield Foundation
Burt and Deedee McMurtry
Michelson 20MM Foundation
National Science Foundation
The Open Society Foundations
Jumee Yhu and David E. Park III
Brian D. Patterson USA-International Foundation
The Bill and Stephanie Sick Fund
Steven L. Smith & Diana T. Go
Stand Together
Robin and Sandy Stuart Foundation
The Stuart Family Foundation
Tammy and Guillermo Treviño
Valhalla Charitable Foundation
White Star Education Foundation
Schmidt Futures
William Marsh Rice University

openstax™

Study where you want, what you want, when you want.

When you access your book in our web view, you can use our new online **highlighting and note-taking** features to create your own study guides.

Our books are free and flexible, forever.
Get started at **openstax.org/details/books/introduction-computer-science**



Access. The future of education.
openstax.org

RICE

CONTENTS

Preface 1

PART 1 PROBLEM SOLVING AND ALGORITHMS

1 Introduction to Computer Science 9

Introduction 9
1.1 Computer Science 10
1.2 Computer Science across the Disciplines 20
1.3 Computer Science and the Future of Society 25
Chapter Review 33

2 Computational Thinking and Design Reusability 39

Introduction 39
2.1 Computational Thinking 40
2.2 Architecting Solutions with Adaptive Design Reuse in Mind 52
2.3 Evolving Architectures into Useable Products 71
Chapter Review 81

3 Data Structures and Algorithms 89

Introduction 89
3.1 Introduction to Data Structures and Algorithms 89
3.2 Algorithm Design and Discovery 97
3.3 Formal Properties of Algorithms 104
3.4 Algorithmic Paradigms 110
3.5 Sample Algorithms by Problem 116
3.6 Computer Science Theory 123
Chapter Review 128

PART 2 REALIZATIONS OF ALGORITHMS

4 Linguistic Realization of Algorithms: Low-Level Programming Languages 143

Introduction 143
4.1 Models of Computation 144
4.2 Building C Programs 156
4.3 Parallel Programming Models 173
4.4 Applications of Programming Models 179
Chapter Review 182

5 Hardware Realizations of Algorithms: Computer Systems Design 193

Introduction 193

5.1 Computer Systems Organization 194
5.2 Computer Levels of Abstraction 198
5.3 Machine-Level Information Representation 205
5.4 Machine-Level Program Representation 211
5.5 Memory Hierarchy 217
5.6 Processor Architectures 226
Chapter Review 230

6 Infrastructure Abstraction Layer: Operating Systems 239

Introduction 239
6.1 What Is an Operating System? 240
6.2 Fundamental OS Concepts 246
6.3 Processes and Concurrency 257
6.4 Memory Management 267
6.5 File Systems 274
6.6 Reliability and Security 279
Chapter Review 285

PART 3 DESIGNING AND DEVELOPING SOFTWARE SOLUTIONS

7 High-Level Programming Languages 297

Introduction 297
7.1 Programming Language Foundations 298
7.2 Programming Language Constructs 311
7.3 Alternative Programming Models 331
7.4 Programming Language Implementation 340
Chapter Review 347

8 Data Management 359

Introduction 359
8.1 Data Management Focus 360
8.2 Data Management Systems 366
8.3 Relational Database Management Systems 372
8.4 Nonrelational Database Management Systems 390
8.5 Data Warehousing, Data Lakes, and Business Intelligence 397
8.6 Data Management for Shallow and Deep Learning Applications 402
8.7 Informatics and Data Management 412
Chapter Review 414

9 Software Engineering 429

Introduction 429
9.1 Software Engineering Fundamentals 430
9.2 Software Engineering Process 440

9.3 Special Topics 468
Chapter Review 490

10 Enterprise and Solution Architectures Management 501

Introduction 501
10.1 Patterns Management 502
10.2 Enterprise Architecture Management Frameworks 510
10.3 Solution Architecture Management 545
Chapter Review 550

PART 4 BUILDING MODERN END-TO-END SOLUTIONS TO BUSINESS AND SOCIAL PROBLEMS

11 Web Applications Development 559

Introduction 559
11.1 Modern Web Applications Architectures 560
11.2 Sample Responsive WAD with Bootstrap and Django 578
11.3 Sample Responsive WAD with Bootstrap/React and Node 604
11.4 Sample Responsive WAD with Bootstrap/React and Django 619
11.5 Sample Native WAD with React Native and Node or Django 624
11.6 Sample Ethereum Blockchain Web 2.0/Web 3.0 Application 637
Chapter Review 650

12 Cloud-Native Applications Development 659

Introduction 659
12.1 Introduction to Cloud-Native Applications 660
12.2 Cloud-Based and Cloud-Native Applications Deployment Technologies 684
12.3 Example PaaS and FaaS Deployments of Cloud-Native Applications 705
Chapter Review 744

13 Hybrid Multicloud Digital Solutions Development 755

Introduction 755
13.1 Hybrid Multicloud Solutions and Cloud Mashups 756
13.2 Big Cloud IaaS Mainstream Capabilities 760
13.3 Big Cloud PaaS Mainstream Capabilities 768
13.4 Towards Intelligent Autonomous Networked Super Systems 784
Chapter Review 799

PART 5 HUMAN-CENTERED RESPONSIBLE COMPUTING

14 Cyber Resources Qualities and Cyber Computing Governance 811

Introduction 811
14.1 Cyber Resources Management Frameworks 812
14.2 Cybersecurity Deep Dive 833

14.3 Governing the Use of Cyber Resources 893
Chapter Review 900

A Appendix A: Network Design Application of Algorithms 911

Index 917

Preface

About OpenStax

OpenStax is part of Rice University, which is a 501(c)(3) nonprofit charitable corporation. As an educational initiative, it's our mission to improve educational access and learning for everyone. Through our partnerships with philanthropic organizations and our alliance with other educational resource companies, we're breaking down the most common barriers to learning. Because we believe that everyone should and can have access to knowledge.

About OpenStax Resources

Customization

Introduction to Computer Science is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 (CC BY NC-SA) license, which means that you can non-commercially distribute, remix, and build upon the content, as long as you provide attribution to OpenStax and its content contributors, and distribute all derivatives under the same license.

Because our books are openly licensed, you are free to use the entire book or select only the sections that are most relevant to the needs of your course. Feel free to remix the content by assigning your students certain chapters and sections in your syllabus, in the order that you prefer. You can even provide a direct link in your syllabus to the sections in the web view of your book.

Instructors also have the option of creating a customized version of their OpenStax book. Visit the Instructor Resources section of your book page on OpenStax.org for more information.

Art Attribution

In *Introduction to Computer Science,* art contains attribution to its title, creator or rights holder, host platform, and license within the caption. For any art that is openly licensed, non-commercial users or organizations may reuse the art as long as they provide the same attribution to its original source. (Commercial entities should contact OpenStax to discuss reuse rights and permissions.) To maximize readability and content flow, some art does not include attribution in the text. If you reuse art from this text that does not have attribution provided, use the following attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license.

Errata

All OpenStax textbooks undergo a rigorous review process. However, like any professional-grade textbook, errors sometimes occur. In addition, the wide range of evidence, standards, practices, data, and legal circumstances in computer science change frequently, and portions of the text may become out of date. Since our books are web-based, we can make updates periodically when deemed pedagogically necessary. If you have a correction to suggest, submit it through the link on your book page on OpenStax.org. Subject matter experts review all errata suggestions. OpenStax is committed to remaining transparent about all updates, so you will also find a list of past and pending errata changes on your book page on OpenStax.org.

Format

You can access this textbook for free in web view or PDF through OpenStax.org, and for a low cost in print. The web view is the recommended format because it is the most accessible—including being WCAG 2.2 AA compliant – and most current. Print versions are available for individual purchase, or they may be ordered through your campus bookstore.

About *Introduction to Computer Science*

Introduction to Computer Science provides a comprehensive foundation in core computer science concepts and principles, aligning with the scope and sequence of most introductory computer science courses. The textbook serves as an engaging entry point for students pursuing diverse fields of study and employment,

including computer science, business, engineering, data science, social sciences, and related disciplines. By addressing a broad learner audience—ranging from computer science majors to non-majors—the book offers a thorough introduction to computational thinking and its applications across multiple domains.

Introduction to Computer Science is designed to be both interactive and practical, focusing on real-world applications that showcase how core computer science concepts can be used to solve complex problems. Students will explore foundational topics, such as algorithms, data structures, computer systems organization, and software development, using an array of engaging, hands-on activities. The textbook integrates meaningful learning experiences through chapter-based scenarios, problem-solving exercises, and project-based assessments that encourage students to apply what they learn in authentic contexts.

Features such as embedded coding exercises, industry insights, and explorations of emerging technology trends provide a holistic approach to learning that extends beyond theory. With a forward-looking perspective, Introduction to Computer Science prepares students to engage with advanced topics in computer science, such as machine learning, cybersecurity, and cloud computing, ensuring they have a solid foundation for continued study and future professional success.

Coverage and Scope

The authors and contributors consulted with other educators and industry professionals from a range of institutions and organizations in order to ensure that *Introduction to Computer Science* meets the diverse needs of both computer science majors and non-majors. The book is structured into five main parts, each focusing on critical areas of the discipline:

- **Part 1: Problem Solving and Algorithms** This section introduces students to the foundations of computer science, focusing on computational thinking, problem-solving techniques, and algorithm design. Topics include data structures, formal properties of algorithms, and algorithmic paradigms. Through practical examples and exercises, students will develop the skills needed to construct and analyze algorithms and understand their applications across various domains.
- **Part 2: Realizations of Algorithms** In this part, students explore how algorithms are realized in hardware and software, starting with low-level programming languages and moving into hardware design and computer systems organization. Students will learn about models of computation, machine-level representation, processor architectures, and memory hierarchy. This foundational understanding enables students to see the connections between abstract algorithms and their physical implementations.
- **Part 3: Designing and Developing Software Solutions** This section covers the principles of software development, high-level programming languages, and data management. Students will learn the fundamentals of software engineering and gain hands-on experience with both relational and non-relational database systems. Emphasis is placed on designing robust software solutions and managing complex data structures, ensuring students are well-prepared for future roles in software development and engineering.
- **Part 4: Building Modern End-to-End Solutions to Business and Social Problems** Students apply their knowledge to design and build web and cloud-native applications. This part includes examples of modern web architectures, responsive design techniques, and cloud-based solutions using PaaS and FaaS technologies. Additionally, students will explore the development of hybrid multi-cloud digital solutions, providing them with experience in addressing complex business and social challenges using modern computing technologies.
- **Part 5: Human-Centered Responsible Computing** The final section delves into the ethical and societal implications of computing. Topics include cybersecurity, governance of cyber resources, and responsible computing practices. Students will learn to navigate the complexities of cybersecurity and governance while considering the broader impacts of technology on society.

Each core concept is designed to build on the previous one, ensuring a coherent learning experience that provides students with a clear view of the field. The book's approach enables students to not only understand

the principles of computer science but also see how they can be applied to address practical, real-world problems.

Pedagogical Foundation and Features

The *Introduction to Computer Science* textbook is designed to engage students through a combination of practical, real-world applications and thought-provoking scenarios that promote critical thinking and a deeper understanding of core concepts. The pedagogical approach is centered on making computer science relevant and accessible for students from diverse backgrounds, whether they are pursuing a degree in computer science or exploring how computational thinking can be applied to their respective fields. To support this vision, the textbook incorporates several key features:

- **Concepts in Practice** features present how computer science concepts are applied in real-world contexts by both professionals and non-professionals. Each box profiles personas and practical applications that demonstrate how core topics, such as algorithms, data management, and software engineering, are utilized across various industries. The purpose is to inspire students—particularly non-majors—by showing them the value of computer science in solving everyday challenges and to foster a greater appreciation for the discipline.
- **Global Issues in Technology** features help students think globally about the societal impact of technology. These boxes highlight how technology affects communities and economies around the world and may introduce topics such as digital equity, environmental sustainability, and global data security. Students are encouraged to consider the broader implications of technological advancements and to think critically about their potential to drive positive change or create new challenges in global contexts.
- **Industry Spotlight** boxes focus on specific industry challenges and how technology progress or application can help solve them. *Industry Spotlight* features introduce students to various sectors—such as healthcare, finance, education, and law—providing a glimpse into how computer science can drive innovation and efficiency. By connecting theoretical concepts to industry-specific problems, these features encourage students to explore the wide-ranging applications of computer science and understand its value across different fields.
- **Link to Learning** features provide a very brief introduction to online resources—videos, interactives, collections, maps, and other engaging resources that are pertinent to students' exploration of the topic at hand.
- **Technology in Everyday Life** features connect computer science principles to students' personal experiences and the world around them. These boxes explore how technology intersects with daily life or current events, making computer science concepts more relatable and relevant. Some features may prompt students to think creatively and propose their own ideas for applying computer science solutions to familiar scenarios.
- **Think It Through** scenarios present students with thought-provoking dilemmas or complex problems related to the use of technology. Students are asked to reflect on ethical questions, problem-solving strategies, and real-world decision-making processes. These features emphasize that not all problems have straightforward answers and encourage students to weigh the pros and cons of different approaches. By navigating these scenarios, students learn to develop judgment skills that are crucial in business and technology environments.

Overall, these features are integrated throughout the material to foster active learning, critical thinking, and an appreciation for the practical applications of computer science. By connecting theory to practice and encouraging students to explore real-world issues, *Introduction to Computer Science* provides a meaningful and supportive learning experience that equips students with the knowledge and skills necessary for success in their academic and professional journeys.

Answers to Questions in the Book

The end-of-chapter Review, Conceptual Questions, Practice Exercises, Problem Sets, Thought Provokers, and

Labs are intended for homework assignments or classroom discussion; thus, student-facing answers are not provided in the book. Answers and sample answers are provided in the Instructor Answer Guide, for instructors to share with students at their discretion, as is standard for such resources.

About the Author

Senior Contributing Author



Senior contributing author: Dr. Jean-Claude Franchitti

Dr. Jean-Claude Franchitti is a Clinical Associate Professor of Computer Science and the Associate Director of Graduate Studies for the CS Master's program in Information Systems at NYU Courant Institute. He earned his M.S. in Electrical Engineering (1985) and his M.S. and PhD. in Computer Science from the University of Colorado at Boulder (1988, 1993). He is the founder and CEO of Archemy, Inc., and has over 40 years of experience in a myriad of industries, and over 30 years of teaching and corporate training experience. He held executive positions in large US-based corporations and leading business technology consulting firms such as Computer Sciences Corporation. He has been involved in many large business technology strategy and modernization projects and has a proven record of delivering large scale business solutions. He was the original designer and developer of jcrew.com and the suite of products now known as IBM InfoSphere DataStage. He also created the Agile Enterprise Architecture Management (AEAM) methodology and a corresponding framework that are patented components of the Archemy business evolution platform. He has developed partnerships with many companies at New York University to incubate new methodologies, transition research into business solutions, help recruit skilled graduates, and increase the companies' visibility. Dr. Franchitti has been a reviewer member on several industry standards committees including OMG, ODMG, and X3H2. Dr. Franchitti taught at CU-Boulder, Denver University, Columbia University, NYU SCPS, before joining NYU Courant Institute in 1997. He has extensive experience with corporate training and developed and delivered training and mentoring programs for the top corporate education providers. He conducted research as part of several NSF- and DARPA-funded research programs.

Dr. Franchitti's teaching and research interests include machine learning, artificial intelligence, data management systems and software engineering with an emphasis on large-scale software architectures and business solutions. He has published articles in numerous refereed publications including the Proceedings of Third Int. Conf. on Cooperative Information Systems, Proceedings of the Sixth International Workshop on Persistent Object Systems, and the 16th International Conference on Software Engineering. Dr. Franchitti received an award for Outstanding Service from NYU's School of Continuing and Professional Studies.

Contributing Authors

Amal Alhosban, University of Michigan–Flint

Mark Buckler, Grand Canyon University

Joanna Gilberti, Archemy, Inc.

Scott Gray, Nashua Community College

Matthew Hertz, University at Buffalo

Andrew Hurd, Empire State University

Kevin Lin, University of Washington

Sai Mukkavilli, Georgia Southwestern State University

Phuc (Brian) Nguyen, UC Irvine

Shahab Tayeb, Fresno State

Zdeněk Troníček, Tarleton State University

Kevin Wortman, Cal State Fullerton

Mohamed Zahran, New York University

Reviewers

Reni Abraham, Houston Community College

Shakil Akhtar, Clayton State University

Kiavash Bahreini, Florida International University

Tammie Bolling, Pellissippi State Community College

Phillip Bradford, UConn

Quiana Bradshaw, Campbellsville University

Christopher Bunton, Austin Community College

Komal Chhibber, South Mountain Community College

Chen-Fu Chiang, SUNY Polytechnic Institute

Gabriel de la Cruz, North Idaho College

Gabriel Ferrer, Hendrix College

David Fogarty, New York University

Yuexing Hao, Massachusetts Institute of Technology

Nazli Hardy, Millersville University

Angela Heath, Baptist Health System

Rania Hodhod, Columbus State University

Benita Hubbard, Southern New Hampshire University

Sumit Jha, Florida International University

Mohamed E. Khalefa, SUNY Old Westbury

Steven Ko, Simon Fraser University

Jessica Kong, University of Washington

Alex Krasnok, Florida International University

Blaise Liffick, Millersville University

Sue McCrory, Missouri State University

Morgan McKie, Florida International University

Sandeep Mitra, SUNY Brockport

Mourya Narasareddygari, Rider University

Saty Raghavachary, University of Southern California

Muhammad Rahman, Clayton State University

Amir Rahmati, Stony Brook University

Caryl Rahn, Florida International University

Jerry Reed, Valencia College

Jordan Ringenberg, University of Findlay

Eman Saleh, University of Georgia

Vincent Sanchez, Florida International University

John Schriner, Queensborough Community College

Tiffanie R. Smith, Lincoln University

Hann So, De Anza College

Jayesh Soni, Florida International University

Derrick Stevens, Mohawk Valley Community College

Kathleen Tamerlano, Cuyahoga Community College

Chintan Thakkar, Rasmussen University

Jingnan Xie, Millersville University

Ning Xie, Florida International University

Additional Resources

Student and Instructor Resources

We have compiled additional resources for both students and instructors, including Getting Started Guides, an instructor's answer guide, test bank, and image slides. Instructor resources require a verified instructor account, which you can apply for when you log in or create your account on OpenStax.org. Take advantage of these resources to supplement your OpenStax book.

Instructor's answer guide. Each component of the instructor's guide is designed to provide maximum guidance for delivering the content in an interesting and dynamic manner.

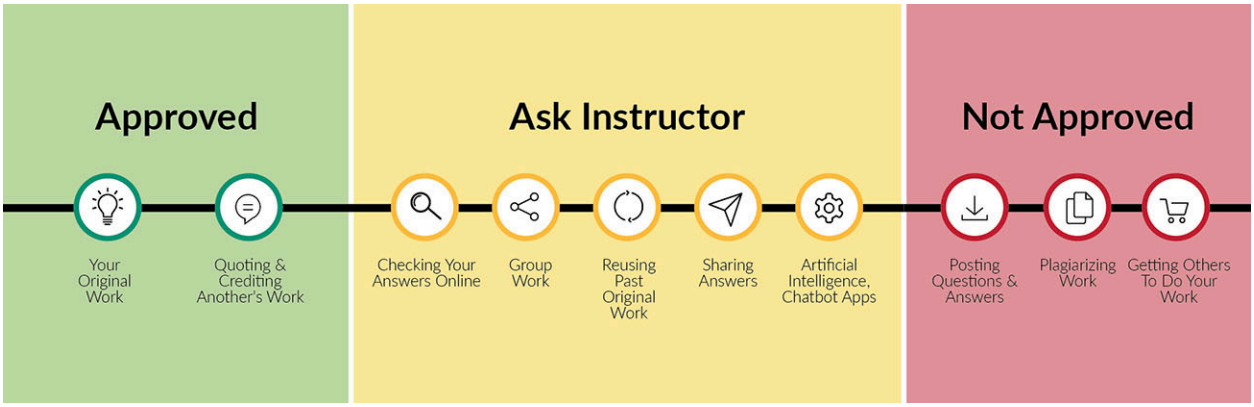
Test bank. With hundreds of assessment items, instructors can customize tests to support a variety of course objectives. The test bank includes review questions (multiple-choice, identification, fill-in-the-blank, true/false), short answer questions, and long answer questions to assess students on a variety of levels. The test bank is available in Word format.

PowerPoint lecture slides. The PowerPoint slides provide learning objectives, images and descriptions, feature focuses, and discussion questions as a starting place for instructors to build their lectures.

Academic Integrity

Academic integrity builds trust, understanding, equity, and genuine learning. While students may encounter significant challenges in their courses and their lives, doing their own work and maintaining a high degree of authenticity will result in meaningful outcomes that will extend far beyond their college career. Faculty, administrators, resource providers, and students should work together to maintain a fair and positive experience.

We realize that students benefit when academic integrity ground rules are established early in the course. To that end, OpenStax has created an interactive to aid with academic integrity discussions in your course.



Visit our academic integrity slider (https://view.genial.ly/61e08a7af6db870d591078c1/interactive-image-defining-academic-integrity-interactive-slider). Click and drag icons along the continuum to align these practices with your institution and course policies. You may then include the graphic on your syllabus, present it in your first course meeting, or create a handout for students. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

At OpenStax we are also developing resources supporting authentic learning experiences and assessment. Please visit this book's page for updates. For an in-depth review of academic integrity strategies, we highly recommend visiting the International Center of Academic Integrity (ICAI) website at https://academicintegrity.org/.

Community Hubs

OpenStax partners with the Institute for the Study of Knowledge Management in Education (ISKME) to offer Community Hubs on OER Commons—a platform for instructors to share community-created resources that support OpenStax books, free of charge. Through our Community Hubs, instructors can upload their own materials or download resources to use in their own courses, including additional ancillaries, teaching material, multimedia, and relevant course content. We encourage instructors to join the hubs for the subjects most relevant to your teaching and research as an opportunity both to enrich your courses and to engage with other faculty. To reach the Community Hubs, visit www.oercommons.org/hubs/openstax.

Technology partners

As allies in making high-quality learning materials accessible, our technology partners offer optional low-cost tools that are integrated with OpenStax books. To access the technology options for your text, visit your book page on OpenStax.org.



1 Introduction to Computer Science

Figure 1.1 Computing is everywhere, affecting everyone, for better and for worse. (credit: modification of "Whereas design is expansive, engineering is narrowing" by Jessie Huynh/Critically Conscious Computing, CC0)

Chapter Outline

1.1 Computer Science
1.2 Computer Science across the Disciplines
1.3 Computer Science and the Future of Society

Introduction

This textbook will introduce you to the exciting and complex world of computer science. In this chapter, you'll review the history of computer science, learn about its use in different fields, and explore how computer science will impact the future of society. Computer science is a powerful tool, and computer scientists have used their vast knowledge of technology to create and implement technology that has transformed societies around the world.

This book will also introduce the computational thinking aspects of problem-solving and analytical thinking that enable the study of algorithms, which are step-by-step instructions for solving specific problems or carrying out computations. Therefore, this book also covers algorithms and their realization via programming languages, computer systems architectures, networks, and operating systems. The book subsequently delves into computer science areas that enable the design and development of software solutions using high-level programming languages (i.e., coding languages designed to be more intuitive for humans), architectural styles and related models, data management systems, and software engineering. Finally, the book demonstrates how to leverage computer science realizations and areas to build modern end-to-end solutions to business and social problems. In particular, the book focuses on modern web applications development, cloud-native applications development, and hybrid Cloud/on-premise digital solutions. The various chapters emphasize how to achieve software solution qualities such as performance and scalability. The last chapter explains how to secure software applications and their applications in the context of various cyber threats. It also explains how to make the right decisions about using computers and information in society to navigate social, ethical, economic, and political issues that could result from the misuse of technology. To conclude this textbook, we'll introduce you to cybersecurity and help you understand why responsible computing is essential to promote

ethical behavior in computer science. The book is designed to help students grasp the full meaning of computer science as a tool that can help them think, build meaningful solutions to complex problems, and motivate their careers in information technology (IT).

You're already familiar with computer science. Whenever you use a laptop, tablet, cell phone, credit card reader, and other technology, you interact with items made possible by computer science. Computer science is a challenging field, and the outputs of computer science offer many benefits for society. At the same time, we have to be cautious about how we use computer science to ensure it impacts society in ethical ways. To help you understand this, the next section will explain how computer science came to be and discuss the field's potential.

1.1 Computer Science

Learning Objectives

By the end of this section, you will be able to:

- Discuss the history that led to the creation of computer science as a field
- Define computer science
- Assess what computer science can do, as well as what it should not do

The field of **computer science (CS)** is the study of **computing**, which includes all phenomena related to computers, such as the Internet. With foundations in engineering and mathematics, computer science focuses on studying algorithms. An **algorithm** is a sequence of precise instructions that enables computing. This includes components computers use to process information. By studying and applying algorithms, computer science creates applications and solutions that impact all areas of society. For example, computer science developed the programs that enable online shopping, texting with friends, streaming music, and other technological processes.

While computers are common today, they weren't always this pervasive. For those whose lives have been shaped by computer technology, it can sometimes seem like computer technology is ahistorical: computing often focuses on rapid innovation and improvement, wasting no time looking back and reflecting on the past. Yet the foundations of computer science defined over 50, and as much as 100, years ago very much shape what is possible with computing today.

The Early History of Computing

The first computing devices were not at all like the computers we know today. They were physical calculation devices such as the abacus, which first appeared in many societies across the world thousands of years ago. They allowed people to tally, count, or add numbers (Figure 1.2). Today, abaci are still used in some situations, such as helping small children learn basic arithmetic, keeping score in games, and as a calculating tool for people with visual impairments. However, abaci are not common today because of the invention of number systems such as the Arabic number system (0, 1, 2, 3, . . .), which included zero and place values that cannot be computed with abaci. The concept of an algorithm was also invented around this time. Algorithms use inputs and a finite number of steps to carry out arithmetic operations like addition, subtraction, multiplication, and division, and produce outputs used in computing. Today's computers still rely on the same foundations of numbers, calculations, and algorithms, except at the scale of billions of numbers and billions of calculations per second.

To introduce a concrete example of an algorithm, let us consider binary search algorithm, which is used to locate a number in a sorted array of integers efficiently. The algorithm operates by repeatedly dividing the search interval in half to perform the search. If the number being searched is less than the integer in the middle of the interval, the interval is narrowed to the lower half. In the alternative, the interval is narrowed to the upper half. The algorithm repeatedly checks until the number is found or the interval is empty.

Algorithms may sound complicated, but they can be quite simple. For example, recipes to prepare food are algorithms with precise directions for ingredient amounts, the process to combine these, and the temperatures and cooking methods needed to transform the combined ingredients into a specific dish. The dish is the output produced by following the algorithm of a recipe.

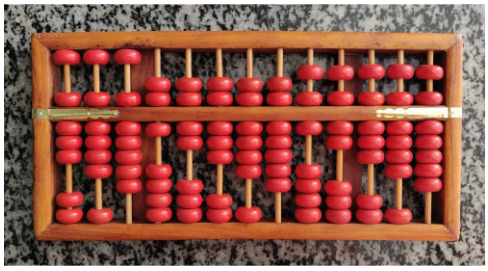


Figure 1.2 An abacus is one of the first calculators. (credit: "Traditional Chinese abacus illustrating the suspended bead use" by Jccsvq/Wikimedia Commons, CC0)

The next major development in the evolution of computing occurred in 1614 when John Napier, a Scottish mathematician, developed logarithms, which express exponents by denoting the power that a number must be raised to obtain another value. Logarithms provided a shortcut for making tedious calculations and became the foundation for multiple analog calculating machines invented during the 1600s.

Scientists continued to explore different ways to speed up or automate calculations. In the 1820s, English mathematician Charles Babbage invented the Difference Engine with the goal of preventing human errors in manual calculations. The Difference Engine provided a means to automate the calculations of polynomial functions and astronomical calculations.

Babbage followed the Difference Engine with his invention of the Analytical Engine. With assistance from Ada Lovelace, the Analytical Engine was program-controlled and included features like an integrated memory and an arithmetic logic unit. Lovelace used punched cards to create sequencing instructions that could be read by the Analytical Engine to automatically perform any calculation included in the programming code. With her work on the Analytical Engine, Lovelace became the world's first computer programmer.

The next major development in computing occurred in the late 1800s when Herman Hollerith, an employee of the U.S. Census Office, developed a machine that could punch cards and count them. In 1890, Hollerith's invention was used to tabulate and prepare statistics for the U.S. census.

By the end of the 1800s and leading into the early 1900s, calculators, adding machines, typewriters, and related machines became more commonplace, setting the stage for the invention of the computer. In the 1940s, multiple computers became available, including IBM's Harvard Mark 1. These were the forerunners to the advent of the digital computer in the 1950s, which changed everything and evolved into the computers and related technology we have today.

Around this time, computer science emerged as an academic discipline rooted in the principles of mathematics, situated primarily in elite institutions, and funded by demand from the military for use in missile guidance systems, airplanes, and other military applications. As computers could execute programs faster than humans, computer science replaced human-powered calculation with computer-powered problem-solving methods. In this way, the earliest academic computer scientists envisioned computer science as a discipline that was far more intellectual and cognitive compared to the manual calculation work that preceded it.

Richard Bellman was a significant contributor to this effort. A mathematics professor at Princeton and later at Stanford in the 1940s, Bellman later went to work for the Rand Corporation, where he studied the theory of multistage decision processes. In 1953, Bellman invented dynamic programming,[1] which is a mathematical optimization methodology and a technique for computer programming. With dynamic programming, complex problems are divided into more manageable subproblems. Each subproblem is solved, and the results are stored, ultimately resulting in a solution to the overall complex problem.[2] With this approach, Bellman helped revolutionize computer programming and enable computer science to become a robust field.

What Is Computer Science?

The term *computer science* was popularized by George E. Forsythe in 1961. A mathematician who founded Stanford University's computer science department, Forsythe defined computer science as "the theory of programming, numerical analysis, data processing, and the design of computer systems." He also argued that computer science was distinguished from other disciplines by the emphasis on algorithms, which are essential for effective computer programming.[3]

Computer science is not only about the study of how computers work, but also everything surrounding computers, including the people who design computers, the people who write programs that run on computers, the people who test the programs to ensure correctness, and the people who are directly and indirectly affected by computers. In this way, computer science is as much about people and how they work with computers as it is about just computers.

Not everyone agrees with this definition. Some people argue that computer science is more about computers or software than the people it affects. However, even if we were to study just the "things" of computer science, the people are still there. When someone designs a computer system, they are thinking about what kinds of programs people might want to run. Typically, effort is made to design the computer system so it is more efficient at running certain kinds of programs. A computer optimized for calculating missile trajectories, for example, won't be optimized for running social media apps.

Many computing innovations were initially developed for military research and communication purposes, including the predecessor to the Internet, the ARPANET (Figure 1.3).

1 S. Golomb, "Richard E. Bellman 1920–1984," n.d. https://www.nae.edu/189177/RICHARD-E-BELLMAN-19201984
2 Geeks for Geeks, "Dynamic Programming or DP," 2024. https://www.geeksforgeeks.org/dynamic-programming/
3 D. E. Knuth, "George Forsythe and the Development of Computer Science," *Communications of the ACM*, vol. 15, no.8, pp. 722–723. 1972. https://dl.acm.org/doi/pdf/10.1145/361532.361538



Figure 1.3 The ARPANET, circa 1974, was an early predecessor to the Internet. It allowed computers at Pentagon-funded research facilities to communicate over phone lines. (credit: modification of "Arpanet 1974" by Yngvar/Wikipedia, Public Domain)

What Is a Computer?

While computer science is about much more than just computers, it helps to know a bit more about computers because they are an important component of computer science. All computers are made of physical, real-world material that we refer to as **hardware**. Hardware—which has four components, including processor, memory, network, and storage—is the computer component that enables computations. The **processor** can be regarded as the computer's "brain," as it follows instructions from algorithms and processes data. The **memory** is a means of addressing information in a computer by storing it in consistent locations, while the **network** refers to the various technological devices that are connected and share information. The hardware and physical components of a computer that permanently house a computer's data are called **storage**.

One way to understand computers is from a hardware perspective: computers leverage digital electronics and the physics of materials used to develop transistors. For example, many of today's computers rely on the physical properties of a brittle, crystalline metalloid called silicon, which makes it suitable for representing information. The batteries that power many of today's smartphones and mobile devices rely on lithium, a soft, silvery metal mostly harvested from minerals in Australia, Zimbabwe, and Brazil, as well as from continental brine deposits in Chile, Argentina, and Bolivia. Computer engineers combine these substances to build circuitry and information pathways at the microscopic scale to form the physical basis for modern computers.

However, the physical basis of computers was not always silicon. The Electronic Numerical Integrator and Computer (ENIAC) was completed in 1945, making it one of the earliest digital computers. The ENIAC operated on different physical principles. Instead of silicon, the ENIAC used the technology of a **vacuum tube**, a physical device like a light bulb that was used as memory in early digital computers. When the "light" in the vacuum tube is off, the vacuum tube represents the number 0. When the "light" is on, the vacuum tube represents the number 1. When thousands of vacuum tubes are combined in a logical way, we suddenly have memory. The ENIAC is notable in computer history because it was the first general-purpose computer, meaning that it could run not just a single program but rather any program specified by a programmer. The ENIAC was often run and programmed by women programmers (Figure 1.4). Despite its age and differences in hardware properties, it shares a fundamental and surprising similarity with modern computers. Anything that can be computed on

today's computers can also be computed by the ENIAC given the right circumstances—just trillions of times more slowly.

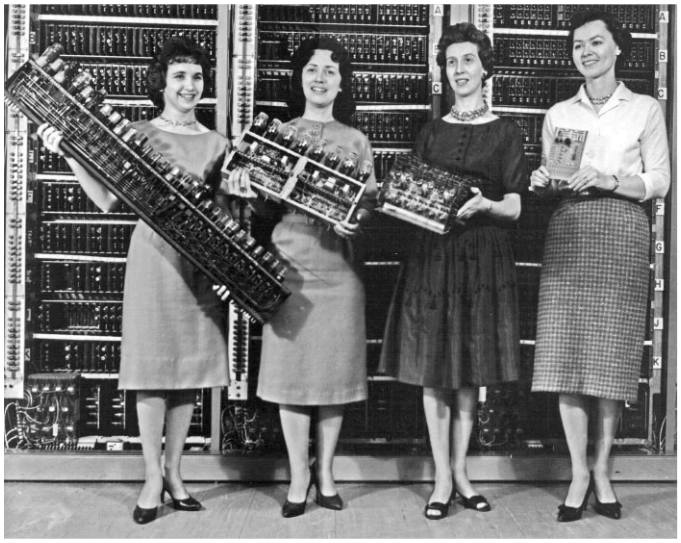


Figure 1.4 This image depicts women programmers holding boards used in computers such as the ENIAC, many of which were designed expressly for ballistics and ordinance guidance research. Today, these room-size computers can be reproduced at a literally microscopic scale—basically invisible to the human eye. (credit: modification of "Women holding parts of the first four Army computers" by U.S. Army/Wikimedia Commons, Public Domain)

How is this possible? The algorithmic principles that determine how results are computed makes up **software**. Almost all computers, from the ENIAC to today's computers, are considered **Turing-complete** (or Computationally Universal, as opposed to specialized computing devices such as scientific calculators) because they share the same fundamental model for computing results and every computer has the ability to run any algorithm. Alan Mathison Turing was an English mathematician who was highly influential in the development of theoretical computer science, which focuses on the mathematical processes behind software, and provided a formalization of the concepts of algorithm and computation with the Turing machine. A Turing-complete computer stores data in memory (either using vacuum tubes or silicon) and manipulates that data according to a **computer program**, which is an algorithm that can be run on a computer. These programs are represented using symbols and instructions written in a **programming language** consisting of symbols and instructions that can be interpreted by the computer. Programs are also stored in memory, which allows programmers to modify and improve programs by changing the instructions.

While both hardware and software are important to the practical operation of computers, computer science's historical roots in mathematics also emphasize a third perspective. Whereas software focuses on the program details for solving problems with computers, theoretical computer science focuses on the mathematical processes behind software. The idea of Turing-completeness is a foundational concept in theoretical computer science, which considers how computers in general—not just the ENIAC or today's computers, but even tomorrow's computers that we haven't yet invented—can solve problems. This theoretical perspective expands computer science knowledge by contributing ideas about (1) whether a problem can be computed by a Turing-complete computer at all, (2) how that problem might be computed using an algorithm, and (3) how quickly or efficiently a computer can run such an algorithm. The answers to these questions suggest the limits of what we can achieve with computers from a technical perspective: Using mathematical ideas, is it possible to use a computer to compute all problems? If the answer to a problem is yes, how much of a computing resource is needed to get the answer?

Clearly both humans and computers have their strengths and limitations. An example of a problem that humans can solve but computers struggle with is interpreting subtle emotions or making moral judgments in complex social situations. While computers can process data and recognize patterns, they cannot fully understand the nuances of human emotions or ethics, which often involve context, empathy, and experience.

On the flip side, there are tasks that neither computers nor humans can perform, such as accurately predicting chaotic systems like long-term weather patterns. Despite advancements in **artificial intelligence (AI)**, computer functions that perform tasks, such as visual perception and decision-making processes that usually are performed by human intelligence, these problems remain beyond our collective reach due to the inherent unpredictability and complexity of certain natural systems.

Theoretical computer science is often emphasized in undergraduate computer science programs because academic computer science emerged from mathematics, often to the detriment of perspectives that center on the social and technical values embodied by applications of computer technology. These perspectives, however, are gradually changing. Just as the design of ARPANET shaped the design of the Internet, computer scientists are also learning that the physical aspects of computer hardware determine what can be efficiently computed. For example, many of today's artificial intelligence technologies rely on highly specialized computer hardware that is fundamentally different at the physical level compared to the general-purpose programmable silicon that has been the traditional focus of computer science. Organizations that develop **human-computer interaction (HCI)**, a subfield of computer science that emphasizes the social aspects of computation, now host annual conferences that bring together thousands of researchers in academia and professionals in the industry. Computer science education is another subfield that emphasizes the cognitive, social, and communal aspects of learning computer science. Although these human-centered subfields are not yet in every computer science department, their increasing representation reflects computer scientists' growing desire to serve not only more engaged students, but also a more engaged public in making sense of the values of computer technologies.

The Capabilities and Limitations of Computer Science

Computers can be understood as sources, tools, and opportunities for changing social conditions. Many people have used computer science to achieve diverse goals beyond this dominant vision for computer science. For example, consider computers in education.

Around the same time that the ARPANET began development in the late 1960s, Wally Feurzeig, Seymour Papert, and Cynthia Solomon designed the LOGO programming language to enable new kinds of computer-mediated expression and communication. Compared to contemporary programming languages such as FORTRAN (FORmula TRANslation System) that emphasized computation toward scientific and engineering applications, LOGO is well known for its use of turtle graphics, whereby programs were used to control the actions of a digital turtle using instructions such as moving forward some number of units and turning left or right some number of degrees. Papert argued that this turtle programming enabled *body-syntonic reasoning*, a kind of experience that could help students more effectively learn concepts in mathematics such as angles, distance, and geometric shapes by instructing the turtle to draw them, and physics by constructing their own understandings via reasoning through the physical motion of turtle programs by showing concepts of velocity, repeated commands to move forward the same amount; acceleration, by making the turtle move forward in increasing amounts; and even friction, by having the turtle slow down by moving forward by decreasing amounts. In this way, computers could not only be used to further education in computer science, but also offer new, more dynamic ways to learn other subjects. Papert's ideas have been expanded beyond the realm of mathematics and physics to areas such as the social sciences, where interactive data visualization can help students identify interesting correlations and patterns that precipitated social change and turning points in history while also learning new data fluencies and the limits of data-based approaches.[4]

Yet despite these roots in aspirations for computers as a medium for learning anything and everything, the study of computer science education emerged in the 1970s as a field narrowly concerned with producing more effective software engineers. Higher-education computer science faculty, motivated by the demand for

4 B. Naimipour, M. Guzdial, and T. Shreiner. 2019. Helping Social Studies Teachers to Design Learning Experiences Around Data: Participatory Design for New Teacher-Centric Programming Languages. In *Proceedings of the 2019 ACM Conference on International Computing Education Research (ICER '19)*. Association for Computing Machinery, New York, NY, USA, 313. DOI: https://doi.org/10.1145/3291279.3341211

software engineers, designed their computer science curricula to teach the concepts that early computer companies such as IBM desperately needed. These courses had an emphasis on efficiency, performance, and scalability, because a university computer science education was only intended to produce software engineers. We live with the consequences of this design even today: the structure of this textbook inherits the borders between concepts originally imagined in the 1970s when university computer science education was only intended to prepare students for software development jobs. We now know that there are many more roles for computer scientists to play in society—not only software engineers, but also data analysts, product managers, entrepreneurs, political advisors or politicians, environmental engineers, social activists, and scientists across every field from accounting to zoology.

Although the role of computers expanded with the introduction of the Internet in the late 1990s, Papert's vision for computation as a learning medium has been challenging to implement, at least partly because of funding constraints. But as computers evolve, primary and secondary education in the United States is striving for ways to help teachers use computers to more effectively teach all things—not just computers for their own sake, but using computers to learn everything.

Computers and Racial Justice

Our histories so far have centered the interests of White American men in computer science. But there are also countless untold, marginalized histories of people of other backgrounds, races, ethnicities, and genders in computing. The book and movie *Hidden Figures* shares the stories of important Black women who were not only human computers, but also some of the first computer scientists for the early digital computers that powered human spaceflight at NASA (Figure 1.5).



Figure 1.5 Katherine Johnson, a Black computer scientist, recalculated the computations done by early digital computers for space flight planning at NASA. Her contributions were portrayed in the book and movie *Hidden Figures*. (credit: "Katherine Johnson at NASA, in 1966" by NASA/Wikimedia Commons, Public Domain)

In one chapter of *Black Software*, Charlton McIlwain shares stories from "The Vanguard" of Black men and women who made a mark on computer science in its early years from the 1950s through the 1990s through the rise of personal computing and the Internet, but whose histories have largely been erased by the dominant Silicon Valley narratives. Their accomplishments include leading computer stores and developing early Internet social media platforms, news, and blog websites. For example, Roy L. Clay Sr., a member of the Silicon Valley Engineering Hall of Fame, helped Hewlett-Packard develop its first computer lab and create the company's first computers. Later, Clay provided information to venture capitalists that motivated them to invest in start-ups such as Intel and Compaq.[5] In another example, Mark Dean was an engineer for IBM whose work was instrumental in helping IBM develop the Industry Standard Architecture (ISA) bus, which created a method of connecting a computer's processor with other components and enabling them to communicate. This led to the creation of PCs, with Dean owning three of the nine patents used to create the original PC.[6]

Yet their efforts were often hampered by the way that computer science failed to center, or even accommodate, Black people. Historically, American Indians and Hispanic people did not have the same access as even Black Americans to computers and higher education. Kamal Al-Mansour, a technical contract negotiator at the NASA Jet Propulsion Lab, worked on space projects while Ronald Reagan was president. He recounts:

"It was conflicting . . . doing a gig . . . supporting missiles in the sky, (while) trying to find my own identity and culture . . . JPL was somewhat hostile . . . and I would come home each day [thinking] What did I accomplish that benefited people like me? And the answer every day would be 'Nothing.'"[7]

Al-Mansour would go on to start a new company, AfroLink, finding purpose in creating software that centered on Black and African history and culture. This story of computer technologies in service of African American communities is reflected in the creation of the Afronet (an early social media for connecting Black technologists) and the NetNoir (a website that sought to popularize Black culture). These examples serve as early indicators of the ways that Black technologists invented computer technologies for Black people in the United States. Yet *Black Software* also raises challenging political implications of the historical exclusion of Black technologists. Black culture on the Internet has greatly influenced mainstream media and culture in the United States, but these Black cultural products are ultimately driving attention and money to dominant platforms such as X and TikTok rather than those that directly benefit Black people, content creators, and entrepreneurs. Computer technologies risk reproducing social inequities through the ways in which they distribute benefits and harms.

The digital divide has emerged as a significant issue, as many aspects of society -- including education, employment, and social mobility -- become tied to computing, computer science, and connectivity. The divide refers to the uneven and unequal access and distribution of technology across populations from different geographies, socioeconomic statuses, races, ethnicities, and other differentiators. While technological access generally improves over time, communities within the United States and around the world have different levels of access to high-speed Internet, cell towers, and functioning school computers. Unreliable electricity can also play a significant role in computer and Internet usage. And beyond systemic infrastructure-based differences, individual product or service access can create a divide within communities. For example, if powerful AI-based search and optimization tools are only accessible through high-priced subscriptions, specific populations can be limited in benefiting from those tools.

5 J. Dreyfuss, "Blacks in Silicon Valley," 2011. https://www.theroot.com/blacks-in-silicon-valley-1790868140
6 IBMers, "Mark Dean," n.d. https://www.ibm.com/history/mark-dean
7 C. D. McIlwain, (2019). *Black software: The Internet and racial justice, from the AfroNet to Black Lives Matter*, New York: Oxford University Press.

GLOBAL ISSUES IN TECHNOLOGY

H-1B Visas Address Worker Shortages

According to the U.S. Bureau of Labor Statistics (BLS), by 2033, the number of jobs available for computer and information research scientists is expected to increase by 26%. This is much faster job growth than the average expected in total for all occupations. BLS predicts that this will result in about 3,400 job openings per year in technology, including computer science.[8]

To fill some of these jobs, U.S. employers likely will continue to rely on H-1B visas. This visa enables employers to recruit well-educated professionals from other countries. These professionals temporarily reside in the United States and work in specialty occupations, like computer science, that require a minimum education of a bachelor's degree or its equivalent.[9] To participate in the visa program, employers must register and file a petition to hire H-1B visa holders. Each year, the U.S. Citizenship and Immigration Services accepts applications from individuals from other countries who compete for a pool of 65,000 visa numbers, as well as an additional pool of 20,000 master's exemption visa numbers awarded that year and valid for a period of three years. At the end of three years, employers can petition to have each worker's visa extended for a period of three additional years.[10] This program helps U.S. employers fill vacancies in many fields, including computer science while providing job opportunities for highly skilled workers around the world.

Computers and Global Development

Computer technology, like any other cutting-edge technology, changes the balance of power in society. But access to new technologies is rarely ever equal. Computer science has improved the quality of life for many people who have access to computer technology and the means of controlling it to serve their interests. But for everyone else in the world, particularly people living in the Global South, computer technologies need context-sensitive designs to meet their needs. In the 1990s, for instance, consumer access to the Internet was primarily based on "dial-up" systems that ran on top of public telephone network systems. Yet many parts of the world, even today, lack telephone coverage, let alone Internet connectivity. Research in computers for global development aims to improve the quality of life for people all over the world by designing computer solutions for low-income and underserved populations across the world—not just those living in the wealthiest countries.

Computer technologies for global development require designing around unique resource constraints such as a lack of reliable power, limited or nonexistent Internet connectivity, and low literacy. Computer scientists employ a variety of methods drawing from the social sciences to produce effective solutions. However, designing for diverse communities is difficult, particularly when the designers have little direct experience with the people they wish to serve. In *The Charisma Machine*, Morgan Ames criticizes the One Laptop Per Child (OLPC) project, a nonprofit initiative announced in 2005 by the Massachusetts Institute of Technology Media Lab. The project attempted to bring computer technology in the form of small, sturdy, and cheap laptops that were powered by a hand crank to children in the Global South. Based on her fieldwork in Paraguay, Ames argues that the project failed to achieve its goals for a variety of reasons, such as electricity infrastructure problems, hardware reliability issues, software frustrations, and a lack of curricular materials. Ames argues that "charismatic technologies are deceptive: they make both technological adoption and social change appear straightforward instead of as a difficult process fraught with choices and politics." When the computers did

8 U.S. Bureau of Labor Statistics, "Computer and Information Research Scientists: Job Outlook," 2024. https://www.bls.gov/ooh/computer-and-information-technology/computer-and-information-research-scientists.htm#tab-6
9 U.S. Citizenship and Immigration Services, "H-1B Specialty Occupations," 2024. https://www.uscis.gov/working-in-the-united-states/h-1b-specialty-occupations
10 American Immigration Council, "The H-1B Visa Program and Its Impact on the U.S. Economy," 2024. https://www.americanimmigrationcouncil.org/research/h1b-visa-program-fact-sheet

work, OLPC's vision for education never truly materialized because children often used the computers for their own entertainment rather than the learning experiences the designers intended. Though Ames's account of the OLPC project (Figure 1.6) itself has been criticized for presenting an oversimplified narrative, it still represents a valuable argument for the risks and potential pitfalls associated with designing technologies for global development: technology does not act on its own but is embedded in a complicated social context and history.



Figure 1.6 In 2005, MIT's Media Lab started the OLPC initiative to bring laptops to children in the Global South. An unexpected outcome they discovered was that designing technologies for global communities is not as straightforward as designers may initially believe. (credit: "One Laptop per Child" by OLE Nepal Cover/Flickr, CC BY 2.0)

THINK IT THROUGH

Internet Commerce

Many products and companies offer services or products over the Internet. While online shopping provides additional sales opportunities for businesses, while offering consumers a convenient shopping option, it is not without risks. For example, online businesses and their shoppers may be victims of data breaches and identity theft. Other risks include fake reviews that motivate consumers to make a purchase, phishing that leads to hacking, and fake online stores that take consumers' money without delivering a product. What can we do to mitigate the risks and dangers of online shopping?

Addressing these risks is not as simple as practicing humility and including communities in the design process. Many challenges in computing for global development are sociopolitical or technopolitical rather than purely technical. For example, carrying out a pilot test to evaluate the effectiveness of a design can appear as favoritism toward the pilot group participants. These issues and social tensions are especially exacerbated in the Global South, where the legacies of imperialism and racial hierarchies continue to produce or expand social inequities and injustices.

The identities of people creating computer technologies for global development are ultimately just as important as the technologies they create. In *Design Justice*, Sasha Costanza-Chock reiterates the call for computer scientists to "build with, not for," the communities they wish to improve. In this way, *Design Justice* seeks to address the social justice tensions raised when asking the question, "Who does technology ultimately benefit?" by centering the ingenuity of the marginalized "user" rather than the dominant "designer."

In some cases, underdeveloped countries can quickly catch up without spending the money that was invested to develop the original technologies. For example, we can set up ad hoc networks quickly today and at a

portion of the cost in Middle Eastern and African countries using technology that was developed (at a high cost) in the United States and Europe over the past several decades. This means that sometimes, progress in one part of the world can be shared with another part of the world, enabling that area to quickly progress and advance technologically.

LINK TO LEARNING

The Design Justice Network (https://openstax.org/r/76DesignJust) is an organization that aims to advance the principles of design justice and to include people who are marginalized in the technology design process.

1.2 Computer Science across the Disciplines

Learning Objectives

By the end of this section, you will be able to:

- Differentiate between discovery and invention
- Describe how science, mathematics, and engineering each play a role in computer science
- Discuss how data science, computational science, and information science each relate to computer science
- Explain why the various areas of computer science are synergistic

Computer science is an incredibly diverse field not because of what it can achieve on its own but because of how it contributes to every other field of human knowledge and expertise. From its early days, it was understood that there would be cross-collaboration between computer scientists and colleagues in other disciplines. Today, almost all modern technologies either depend on computer technologies or benefit significantly from them. Computer technologies and the study of computer science have reshaped almost all facets of life today for everyone.

Data Science

Across business, financial, governmental, scientific, and nonprofit workplaces, millions of people are programming, and most of the time, they don't even know it! A **spreadsheet** is an example of a data-centric programming environment where data is organized into cells in a table. Instead of presenting programs as primarily about algorithms, spreadsheets present programs as primarily about data. Spreadsheets are often used for data analysis by offering a way to organize, share, and communicate ideas about data. Spreadsheets are uniquely effective and accessible because they allow for the visual organization of data in whichever structure makes the most sense to the user. Instead of hiding data behind code, spreadsheets make data as transparent and up to date as possible.

Although spreadsheets make computation accessible for millions of people across the world, they have several shortcomings. Unless limits are removed, many popular spreadsheet software products such as Microsoft Excel may have a limitation to the number of rows of data they can store that is less than the data of modern computers. One such example occurred in October 2020 when Public Health England failed to report 15,841 positive cases of COVID-19 in the United Kingdom due to mismanaged row limits in the spreadsheet used. This shortcoming attests not only to the technical limit on the number of rows supported by spreadsheets, but also to the design limitations of software that fails to communicate data loss, irregularities, or errors to users. Errors in spreadsheet software data entry can often go unnoticed because spreadsheets do not enforce data types. Cells can contain any content: numbers, currencies, names, percentages, labels, and legends. The meaning of a cell is determined largely by the user rather than the software. Spreadsheets are an expressive and accessible technology for data analysis, but this creative power that spreadsheets afford to users is the

very same power that limits spreadsheets as a data management and large-scale data analysis tool. The more data and the more people involved in a spreadsheet, the greater the potential for spreadsheet problems.

The interdisciplinary field that applies computing to managing data and extracting information from data is called **data science**. Data scientists are practitioners who combine computing and data analysis skills with the domain knowledge specific to their field or business. The demand for data scientists is becoming increasingly important as more and more research and business contexts involve analyzing **big data**, or very large datasets that are not easily processed using spreadsheets. These datasets often involve high-volume measurements of user interactions on the Internet at a very fine grain, such as tracking a customer's web browser history across an online storefront. Data scientists can then analyze browser patterns using machine learning methods in order to recommend related products, target advertisements to specific customers over social media, and reengage customers over email or other means. Machine learning (ML) is a subset of artificial intelligence that relies on algorithms and data to make it possible for artificial intelligence to learn, actually mimicking the way humans learn. For example, ML is used to identify fraudulent versus legitimate banking transactions. Once a computer learns how to distinguish fraudulent transactions, it can be alert and call attention to suspicious banking activity.

GLOBAL ISSUES IN TECHNOLOGY

Targeted Advertising

Although data scientists can produce immense value for business and research alike, their work also raises significant social concerns. For example, web browser history tracking enables companies to target advertising of products to people and also allows for targeting of political advertisements. In *Antisocial Media*, Siva Vaidhyanathan argues that the "impact of Facebook on democracy is corrosive [because political campaigns] can issue small, cheap advertisements via platforms like Facebook and Instagram that may target "groups as small as twenty, and then disappear, so they are never examined or debated." This undermines the process of discussions among voters in democracies like the United States, as well as countries like Germany, United Kingdom, and Spain, which spent the most on targeted political advertising on Facebook in the spring of 2019.[11] Given their lack of transparency, such ads are a questionable practice.

Another interesting topic worth mentioning here is targeted advertising toward children.[12] It is important to consider the ethical implications of using the data collected through tracking, especially when it comes to targeting at-risk populations. It raises questions about the accountability of platforms and advertisers in safeguarding users' rights and ensuring transparency in how data is used for these purposes.

Computational Science

Beyond data science, computer science can also fundamentally change how science is researched and developed. The field of **computational science** refers to the application of computing concepts and technologies to advance scientific research and practical applications of scientific knowledge in a wide range of fields, including civil engineering, finance, and medicine (among many others). For example, algorithms and computer software play a key role in enabling numerical weather prediction (Figure 1.7) or the use of mathematical models to forecast weather based on current conditions in order to assist peoples' everyday lives and contribute to our understanding of the climate models, climate changes, and climate catastrophes. These algorithms may rely on a large amount of computer hardware power that might not be available in a single system, so the work may need to be distributed across many computers. Computational science studies methods for realizing these algorithms and computer software.

11 Statista, "European Elections: Countries that spent the most on targeted political advertising on Facebook from March 1 to May 26, 2019*," 2019. https://www.statista.com/statistics/1037329/targeted-political-ad-spend-on-facebook-by-eu-countries/
12 Maya Brownstein. Harvard study is first to estimate annual ad revenue attributable to young users of these platforms. January 2, 2024. https://news.harvard.edu/gazette/story/2024/01/social-media-platforms-make-11b-in-ad-revenue-from-u-s-teens/

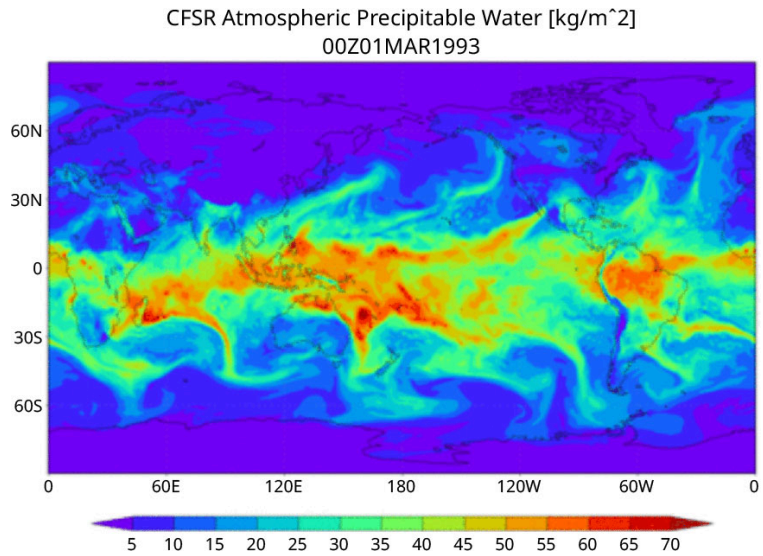


Figure 1.7 Meteorologists collect data from a variety of sources and use the data, algorithms, and computers to predict the weather. (data source: Climate Forecast System, National Centers for Environmental Information, National Oceanic and Atmospheric Administration, https://www.ncei.noaa.gov/products/weather-climate-models/climate-forecast-system; credit: modification of "CFSR Atmospheric Precipitable Water" by NOAA/ncei.noaa.gov, Public Domain)

INDUSTRY SPOTLIGHT

Computer Science and Climate Change

Computer science fights climate change and limits the impacts of climate catastrophes by enabling technologies for decarbonization through power consumption optimization and advancing renewable energy sources. Numerical weather forecasting not only supports our everyday lives but also helps climate scientists determine the precise locations for wind turbines and simulate how they should be designed to enable the greatest energy production. To support the power grid, data science methods can help predict peak power consumption, optimize power sources to produce exactly the right amount of power needed, and adjust power storage to reduce the amount of energy that needs to be generated from nonrenewable sources. Computer models and algorithms assist energy engineers in optimizing building air conditioning and power demands so that they efficiently serve the people living, working, and playing within them.

Although computer science has been used to support scientific discovery, the theory of knowledge of computer science has historically been considered quite different from that of the natural sciences, such as biology, physics, and chemistry. Computer science does not study natural objects, so to most, it would not be considered a natural science but rather an applied science. Unlike natural sciences such as biology, physics, and chemistry, which emphasize the *discovery* of natural phenomena, computer science often emphasizes *invention* or *engineering*.

However, computer science is today deeply interdisciplinary and involves methods from across science, mathematics, and engineering. Computer scientists design, analyze, and evaluate computational structures, systems, and processes.

- Mathematics plays a key role in theoretical computer science, which emphasizes how a computational problem can be defined in mathematical terms and whether that mathematical problem can be efficiently solved with a computer.
- Engineering plays a key role in **software engineering**, which emphasizes how problems can be solved with computers as well as the practices and processes that can help people design more effective software solutions.

- Science plays a key role in human-computer interaction, which emphasizes experimentation and evaluation of the interface (boundary) between humans and computers, often toward designing better computer systems.

Information Science

Not only is computation interdisciplinary, but other disciplines are also becoming more and more computational. In *The Invisible Future*, Nobel Laureate biologist David Baltimore defines DNA in computational terms. He states that biology is an information science because DNA encodes for the outputs of biological systems. The interdisciplinary field studying information technologies and systems as they relate to people, organizations, and societies is called **information science**. The role of information in natural sciences can also be found in the physics of quantum waves that carry information about physical effects, in the chemical equations that specify information about chemical reactions, in the information flows that drive the evolution of economies and political organizations, and in the information processes underlying social, management, and communication sciences.[13]

CONCEPTS IN PRACTICE

Computer Science and DNA

Research into DNA sequencing and indexing is opening new ways of helping medical providers offer personalized treatments for patients. Large-scale genome sequencing of not only the human genome, but also the DNA signatures for viruses has made it possible for medical providers to take human fluid samples and analyze them for the presence of infectious diseases. This research requires computer science concepts, including specialized medical computer devices to sequence the billions of nucleotides that form a DNA sequence, data structures and algorithms to efficiently process and identify DNA signatures, and the miniaturization of computer hardware so that this technology is accessible (both in terms of price and physical size) in more and more care centers.

Although information science has its roots in information classification, categorization, and management in the context of library systems, information science today is a broad field that encompasses the many diverse ways information shapes society. For example, today's social media networks provide more personable and instantaneous information communication compared to traditional news outlets—billions of people around the world are using social media to engage with information about the world. For many people, social media may be the primary way that they learn about and make sense of the world (Figure 1.8). Yet, we've already seen risks associated with information technologies such as the Internet. In today's "information age," information has more power than ever before to reshape society. Information scientists, data scientists, computational scientists—and, therefore, computer scientists—have a social responsibility: "One cannot reap the reward when things go right but downplay the responsibility when things go wrong."[14]

13 P. J. Denning, "Computing is a natural science," *Commun. ACM*, vol. 50, no. 7, pp. 13–18, July 2007. https://doi.org/10.1145/1272516.1272529.
14 R. Benjamin, "Race after technology: Abolitionist tools for the new Jim code," 2019, Polity.

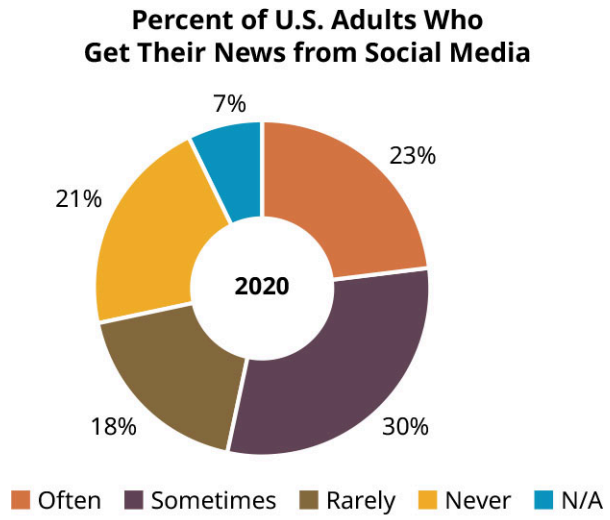


Figure 1.8 While Americans used to primarily get their news from newspapers, as technology has advanced, their primary source of news media has shifted. As of 2020, 53% of American adults surveyed stated they got their news from social media at least some of the time. (data source: Elisa Shearer and Amy Mitchell, Pew Research Center. "About Half of Americans Get News on Social Media at Least Sometimes." From Survey of U.S. adults conducted Aug. 31–Sept. 7, 2020. In: E. Shearer, A. Mitchell, "News Use Across Social Media Platforms in 2020," Jan 12, 2021.; attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Despite the centrality of information to decision-making and social change, dominant approaches to computer science tend to focus on computational structures, systems, and processes (such as algorithms) that describe one kind of information by focusing on the *what* or *how* of solving problems with computers, but less often the *why* or *who* questions. Information science broadly centers people, organizations, and society in the study of information technologies.

Computer Science Is an Interdisciplinary Field

In presenting data science, computational science, and information science, we've introduced the idea that computer science can shape other disciplines. But we've also raised questions about what computer science is today. If computer science is the study of all "phenomena surrounding computers," it could also involve data science, computational science, bioinformatics, cheminformatics, computational social science, medical informatics, and information science. As you will learn in Chapter 13 Hybrid Multicloud Digital Solutions Development, another aspect of computer science is responsible computing, which includes the appropriate management of cyber resources as well as robust cybersecurity. It is difficult to define computer science today because it is so widely used by people across the world in diverse capacities. Definitions are about defining boundaries and excluding practices, which may be helpful for understanding the practices of a certain culture or group that is "doing" computer science, but it can never truly represent everyone and all the things that people are doing with computer science. However, computer science's historical roots in mathematics shape the way it categorizes subfields:

- Theoretical computer science
  - Theory of computation
  - Information representation
  - Data structures and algorithms
  - Programming language and formal methods
- Computer systems
  - Architecture
  - Artificial intelligence
  - Networks

- Security
- Databases
- Distributed computing
- Graphics

- Applied computer science
  - Scientific computing
  - Human-computer interaction
  - Software engineering

In this hierarchy, theoretical computer science and computer systems are treated separately from applied computer science and human-computer interaction, suggesting that the mathematics of computing are pure and separate from social questions. Yet we've seen several examples that question this paradigm and instead point to a structure where human-computer interaction is infused throughout the study of computer science and all its subfields.

Today, computer science is a field that is just as much about people as it is about computer technology because each of these subfields is motivated by the real-world problems that people ultimately want to solve. The subfields of artificial intelligence and machine learning have applications that directly influence human decision-making, ranging from advertisement targeting to language translation to self-driving cars. Effective computational solutions to research or business problems require combining specific knowledge with computer science concepts from a combination of areas. For example, the computational science application of weather prediction combines knowledge about various subfields of computer science (algorithms, distributed computing, computer systems) with knowledge about climate systems. Theoretical computer scientists are increasingly interested in asking questions such as, "How do we design, analyze, and evaluate algorithms or information systems for fairness? How do we even define fairness in a computer system that strips away the complexities of the real world? What ideas or information are encoded in the data? And what are the limits of our approaches?" Computer science is a complex field, and its synergistic nature means that when computer science is used in an interdisciplinary manner that shapes other disciplines, its impact on society is much greater than when each discipline functions on its own.

1.3 Computer Science and the Future of Society

Learning Objectives

By the end of this section, you will be able to:

- Discuss how computer scientists develop foundational technologies
- Discuss how computer scientists evaluate the negative consequences of technologies
- Discuss how computer scientists design technologies for social good

As noted earlier, computer science is a powerful tool, and computer scientists have vast technological knowledge that continues transforming society. Computer scientists have an obligation to be ethical and good stewards of technology with an emphasis on responsible computing. Written code influences daily life, from what we see on social media to the news stories that pop up in a Google search and even who may or may not receive a job interview. When computer scientists don't consider the ramifications of their code, there can be unintended consequences for people around the world. The Y2K problem, also known as the "millennium bug," is a good example of shortsighted decisions that allowed computer scientists to only store the last two digits of the year instead of four. This made sense at a time when memory was expensive on both mainframe computers and early versions of personal computers. The Y2K problem was subsequently coined by John Hamre, the United States Deputy Secretary of Defense, as the "electronic equivalent of the El Niño."[15] The future of computer science will highly affect the future of the world. Although we often think of computer technologies as changing the way the world works, it is actually people and their vision for the future that are amplified by computing. The relationship between computer science and people is about how computer technologies can bias society and how the choices made through computer systems can both promote and discourage social inequities. Computer technologies can encode either value or both values in their designs. In this section, we'll introduce three ways that computer science can shape the future of society: developing foundational technologies, evaluating negative consequences of technologies, and designing technologies for social good.

Developing Foundational Technologies

We've seen how foundational technologies like artificial intelligence, algorithms, and mathematical models enable important applications in data science, computational science, and information science.

As noted previously, artificial intelligence (AI) is the development of computer functions to perform tasks, such as visual perception and decision-making processes, that usually are performed by human intelligence. AI refers to a subfield of CS that is interested in solving problems that require the application of machine learning to human cognitive processes to achieve goals. AI research seeks to develop algorithm architectures that can make progress toward solving problems. One such example is **image recognition**, or the problem of identifying objects in an image. This problem is quite difficult for programmers to solve using traditional programming methods. Imagine having to define very precise rules or instructions that could identify an object in an image regardless of its position, size, lighting conditions, or perspective. As humans, we have an intuitive sense of the qualities of an object. However, representing this human intelligence in a machine requiring strict rules or instructions is a much harder task. AI methods for image recognition involve designing algorithm architectures that can generalize across all the possible ways that an object can appear in an image.

INDUSTRY SPOTLIGHT

Agricultural Robots

Agricultural robots help large-scale industrial farmers produce crops more efficiently and support sustainability efforts. One agricultural robot is now being used to improve fertilizer and pesticide treatments by taking pictures of plants as a farmer drives a tractor over the field. Artificial intelligence techniques are used to recognize and identify the lettuce plants and weed plants in the image. For each identified lettuce or weed plant, the robot makes a personalized decision about the best chemical treatment for the plant in real time as the tractor moves to the next row of crops. This ability to personalize chemical treatments improves yields and plant quality for large-scale industrial agriculture by producing more crops with fewer chemicals.

15 "Looking at the Y2K bug," portal on CNN.com. Archived 7 February 2006 at the Wayback Machine. https://web.archive.org/web/20060207191845/http://www.cnn.com/TECH/specials/y2k/

Recent approaches to AI for image recognition draw on a family of methods called neural networks instead of having programmers craft rules or instructions by hand to form an algorithm. In humans, the neural network is a complex network in the human brain that consists of neurons, or nerve cells, connected by synapses that send messages and electrical signals to all parts of the body, enabling us to think, move, feel, and function.

In computer science, a **neural network** (Figure 1.9) is an AI algorithm architecture that emphasizes connections between artificial nerve cells whose behavior and values change in response to stimulus or input. These neural networks are not defined by individual neurons but by the combination of all the neurons in the network. Typically, artificial neurons are arranged in a hierarchy that aims to capture the structure of an image. Although the first level of neurons might respond to individual pixels, later levels of artificial neurons might respond in aggregate to the arrangement of several artificial neurons in the preceding layer. This is similar to how the human visual system responds to edges at the lower levels, then responds in aggregate to the specific arrangement of several edges in later levels, and ultimately identifies these aggregated arrangements as objects.

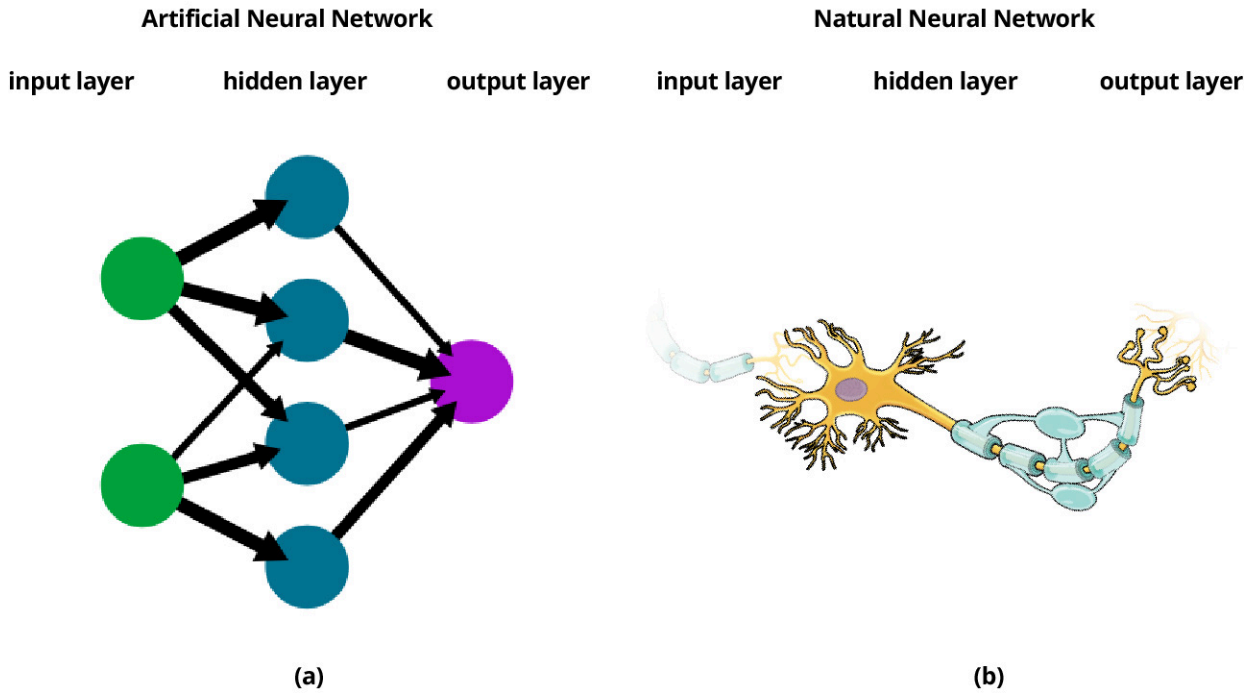


Figure 1.9 (a) An artificial neural network consists of three key layers: the input layer, where raw data enters the system; the hidden layer, where information is processed and patterns are identified; and the output layer, where results are presented. (b) A natural neural network, such as those in the human body, mirrors this structure. The input layer represents sensory receptors, like those in the retina. The hidden layer corresponds to the synapse, where partial processing of the sensory data occurs. Finally, the output layer represents the information sent to the brain for final processing and interpretation. (credit a: modification of "Neural network example" by Wiso/Wikipedia, Public Domain; credit b: modification of work from *Psychology 2e*. attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The idea of neural networks, however, is not as new as it might seem. Artificial neural networks were first imagined in the mid-1900s alongside contemporary research efforts in the cognitive sciences. The ideas of multilayered, hierarchical networks of neurons and the mathematical optimization methods for learning were all there, but these early efforts were limited by the computational processing power available at the time. In addition, the large datasets that drive neural network learning were not nearly as available as they are today with the Internet. Developments in foundational technologies such as computer architecture and computer networks paved the way for the more recent developments in neural network technologies. The broad area of computer systems investigates these architectures and networks that enable new algorithms and software. Without these technologies, neural networks would not be nearly as popular and revolutionary as they are today. Yet the relationship between computer systems and AI development is not one-directional. Today, computer scientists are using neural networks to help design new, more efficient computer systems. The

development of foundational computer technologies not only creates opportunities for direct and indirect applications, but also supports the development of other computer technologies.

Just as we saw how technological fixes embodied a powerful belief about the relationship between computer solutions and social good, a similar cultural belief exists about the relationship between foundational technologies and their social values. The belief that technologies are inherently neutral and that it is the people using technology who ultimately make it "good" or "bad" is considered **social determination of technology**.

THINK IT THROUGH

Social Determination of Technology

Do you agree with the social determination of technology? Is it possible for technology—before it is used by people to solve certain problems—to encode social values? Try to come up with an example that would support this belief. What about an example that refutes this belief? What are the social implications of agreeing or disagreeing with the social determination of technology?

Today's neural networks are designed to identify patterns and reproduce existing data. It is widely accepted that many big datasets can encode social preferences and values, particularly when the data is collected from users on the Internet. A social determination of technology accepts this explanation of AI bias and leaves the design of AI algorithms and techniques as neutral: the bias in an AI system is attributed to the social values of the data rather than the design of the AI algorithms. Critics of social determination point out that the way AI algorithms learn from big data represents a social value, one that encodes a default preference for reproducing the biases inherent in big data. This applies whether the AI application is about fair housing, medical imaging, ad targeting, drone strikes, or another topic. This is an issue that computer scientists must consider as they practice responsible computing and strive to ensure that data is gathered and handled as ethically as possible.

Evaluating Negative Consequences of Technology

Today's AI technologies work by reproducing existing patterns rather than imagining radically different futures. As much as neural networks are inspired by the human brain, it would be a stretch to suggest that AI systems have any semblance of general intelligence. Though these systems might be quite effective at identifying lettuce plants from weed plants in an image, their capacity for humanlike intelligence is limited by design. A neural network learns to recognize similar patterns that appear across millions or billions of sample images and represent these patterns with millions or billions of numbers. Mathematical optimization methods are used to choose the numeric values that best encode correlations across the sample images. However, current approaches lack a deeper, conceptual representation of objects. One criticism of very large neural networks is that there are often more numeric values than there are sample images—the network can effectively memorize the details of a million sample images by encoding them in a billion numbers. Many of today's neural networks recognize objects in images not by relying on some intrinsic idea or concept of objects but by memorizing every single configuration of edges as they appear in the sample images.

This limitation can lead to peculiar outcomes for image recognition systems. Often, neural network approaches for image recognition have certain examples of images where objects are misidentified in unusual ways: a person's face might be recognized in a piece of toast or in a bunch of clouds in the sky. In these examples, the pattern of edges might coincidentally trigger the neural network values so that it misidentifies objects. These are among the more human-understandable examples; there are many other odd situations that are less explainable. An **adversarial attack** is a sample input (e.g., an image) that is designed to cause a system to behave problematically. Researchers have found that even tweaking the color of just a single point in an image can cause a chain reaction in the neural network, leading it to severely misidentify objects. The

adversary can choose the color of the point in such a way as to almost entirely control the output of some neural networks: changing a single specific point in an image of a dog might cause the system to recognize the object as a car, airplane, human, or almost anything that the adversary so desires. Moreover, these adversarial attacks can often be engineered to cause the neural network to report extremely high confidence in its wrong answers. Self-driving cars that use neural networks for image recognition can be at risk of real-world adversarial attacks when specially designed stickers are placed on signs that cause the system to recognize a red light as a green light (Figure 1.10). By studying adversarial attacks, researchers can design neural networks that are more robust and resilient to these attacks.

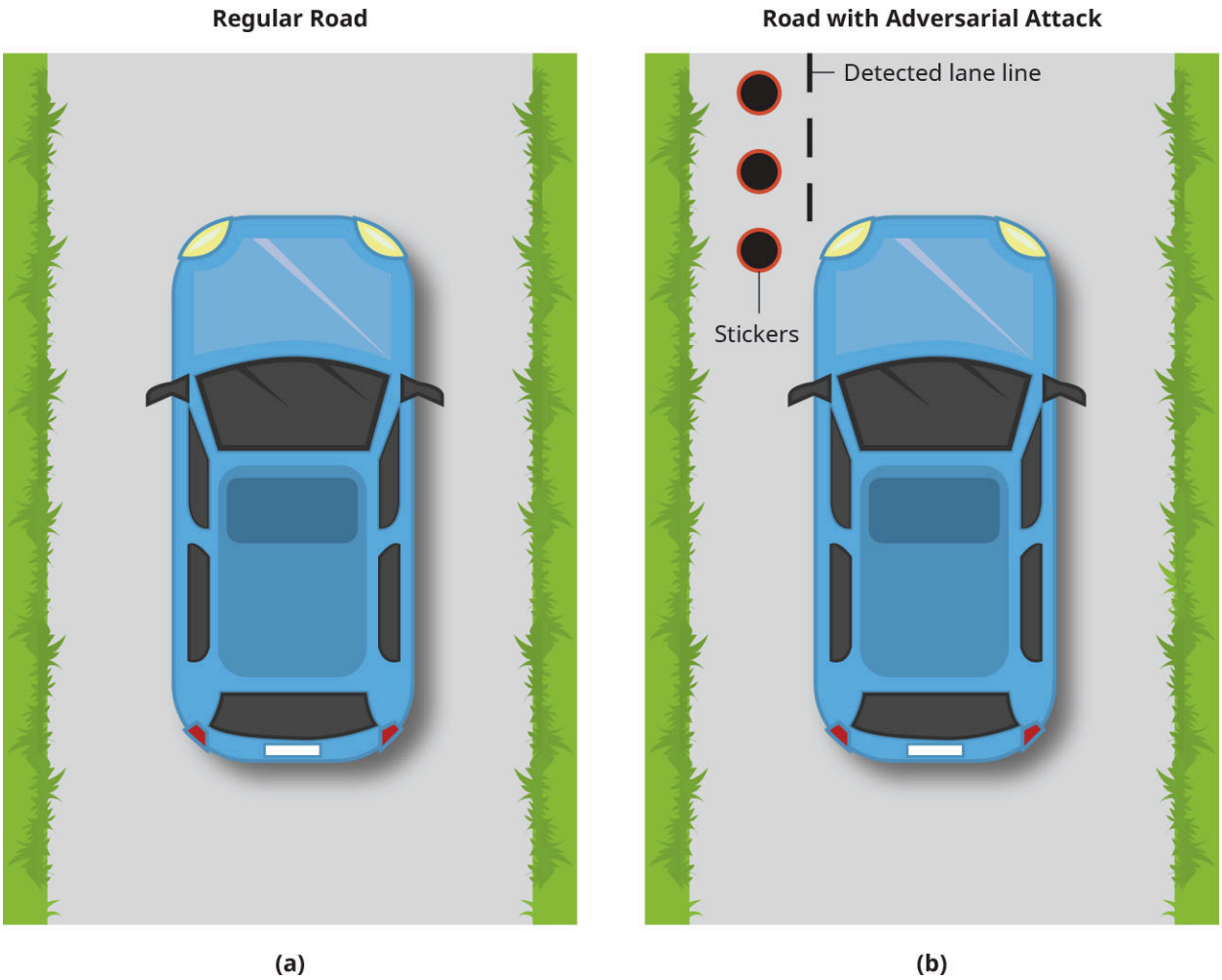


Figure 1.10 (a) Autopilot functions in self-driving cars generally identify roads and lanes using artificial intelligence to "see" road markings. (b) Researchers were able to trick these cars into seeing new lanes by using as few as three small stickers, to confuse the neural networks and force the cars to change lanes. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

In general, research is an important part of computer science. Through research, computer scientists analyze ways that technology can be used and gain insight and answers to address issues and improve various aspects of society. Research enables computer scientists to make advancements like the design of new algorithms, development of new hardware and software, and applications for emerging technologies such as AI.

One important use of research is to investigate adversarial attacks to gather answers needed for computer scientists to improve foundational technologies by evaluating the negative consequences of technologies. Computer technologies offer a unique medium for learning things (not just learning computer science), connecting with each other, and enhancing the lives of people all around the world. Yet, in each of these examples, we also raised concerns about how these technologies unfolded and affected people's lives in both positive and negative ways. While we can rarely, if ever, paint any one technology as purely "good" or "bad,"

computer scientists are interested in studying questions around how technologies are designed to center social values. Social scientists are not solely responsible for answering questions about technology, but computer scientists can also contribute important knowledge and methods toward understanding computer technologies.

Designing Technologies for Social Good

Computer science can advance social good by benefiting many people in many different areas, including public health, agricultural sustainability, climate sustainability, and education.

Computer technologies accelerate medical treatments for public and personal health from initial research and development to clinical trials to large-scale production and distribution. In January 2020, Chinese officials posted the genetic sequence of the coronavirus SARS-CoV-2. This helped pharmaceutical companies to begin developing potential vaccines for the virus at a significantly faster rate than for any other virus in the past (Figure 1.11).

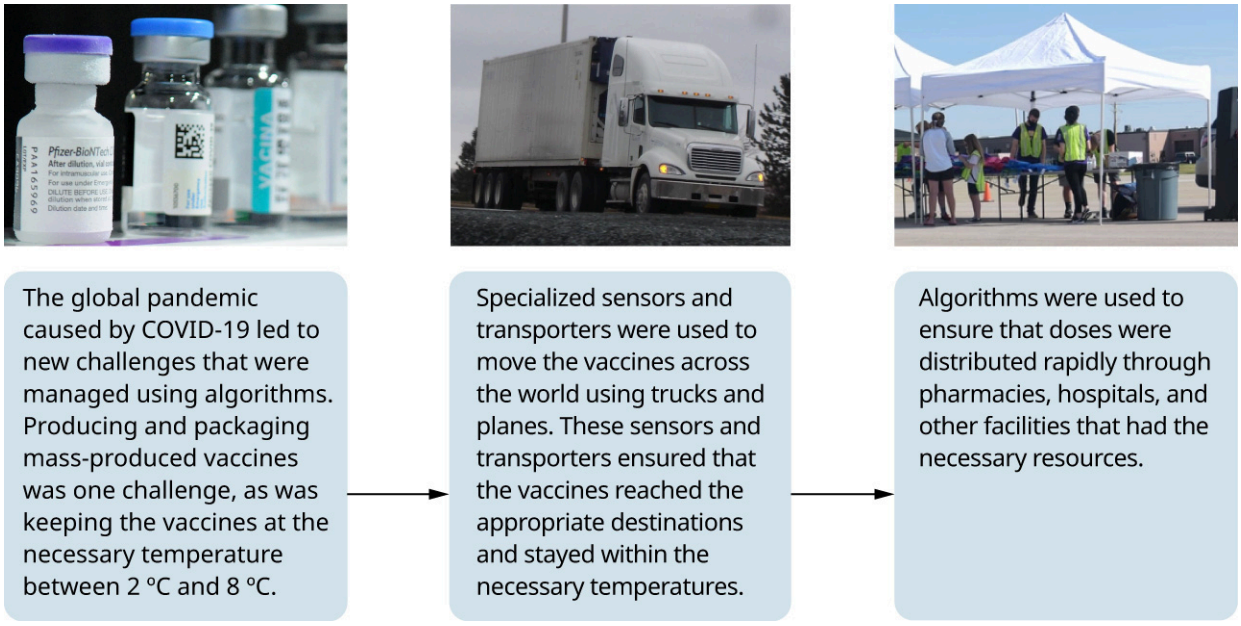


Figure 1.11 The SARS-CoV-2 outbreak that began in 2020 displayed how quickly computer science could be harnessed by governments, medical facilities, and scientists to decode the virus, develop treatments, and distribute vaccinations around the world. What would have been a very difficult feat to manage manually was simplified through the use of algorithms and computer technology. (credit left: modification of "COVID-19 vaccines" by Agência Brasília/Flickr, CC BY 2.0; credit center: modification of "T04" by Sarah Taylor/Flickr, CC BY 2.0; credit right: modification of “Back2School Brigade prepares families for upcoming school year” by Thomas Karol/DVIDS, Public Domain)

Computational science enables the miracles of modern medicine. Viral sequences can be digitized and rapidly shared between researchers across the world via the Internet. Computer algorithms and models can simulate the human immune system responses to particular treatments within hours rather than years. The first treatments can then be produced at a small scale using computer-engineered cells in less than a month from the initial sequencing. To ensure the treatments are safe and effective, clinical trials are held at disease transmission “hot spots” predicted using data science methods drawing on data aggregated and monitored from across the world. Once a treatment is proven safe and effective, it is mass-produced with the help of computer-controlled robots and automated assembly lines. Algorithms manage the inventory supply and demand and control the transportation of treatments on trucks and planes guided by computer navigation systems. Web apps and services notify people throughout the process.

Yet the use of computer technology throughout modern medicine is anything but politically neutral. Computers, algorithms, and mathematical models solve the problems that their creators wish to solve and encode the assumptions of their target populations. Supply and demand data for the data models are

determined by various factors, at least partly in response to the money and relationships between countries that control the technology, the Global North, and countries that don't, the Global South. Within local communities, the uptake of medical treatments is often inequitable, reflecting and reinforcing historical inequities and disparities in public health. Computer technology alone often doesn't address these issues. In fact, without people thinking about these issues, computer technologies can often amplify disparities. Consider datasets, which can be biased if they overrepresent or underrepresent specific groups of people. If decisions are made on the basis of biased data, people in the groups that are not represented fairly may receive inequitable treatment. For example, if a local government agency is working with a biased dataset, political leaders may make decisions that result in certain citizens receiving inadequate funding or services. This is an example of why responsible computing, which we will cover in Chapter 14 Cyber Resources Qualities and Cyber Computing Governance, is so important.

These problematic histories are not only aggravated in medicine and public health, but also reflected in housing. Redlining refers to the inequitable access to basic public services based on residents' neighborhoods and communities, which includes the practice of withholding financial services from areas with a large underrepresented population. In the United States, these communities reflect the histories of racial segregation and racial wealth inequalities. Fair housing laws are intended to prevent property owners from discriminating against buyers or renters because of race, color, ability, national origin, and other protected classes. But computer technologies also present new kinds of challenges. Microtargeted ads on social media platforms contribute to not only political polarization, but also discrimination in housing. This can be a particular problem when combined with redlining. Even if the ad targeting is not explicitly designed to discriminate, microtargeted ads can still reinforce historical redlining by incorporating data such as zip codes or neighborhoods. This may result in digital redlining, which is the practice of using technology, such as targeted ads, to promote discrimination. In 2021, a Facebook user filed a class-action lawsuit that argued nine companies in the Washington, D.C., area deliberately excluded people over the age of 50 from seeing their advertisements for housing because they wanted to attract younger people to live in their apartments.[16] This is an example of an issue in technology that should be addressed by responsible computing with an emphasis on ethical behavior.

With good intentions and attention to personal biases, technologies can be designed for social good. For example, a hypothetical algorithm for fair housing could evenly distribute new housing to people across protected classes and marginalized identities, such as older populations. Of course, algorithmic control and automated decision-making is challenged to consider the underlying conditions behind social problems. Still, algorithms can be important tools to enable us to distribute outcomes more fairly from a statistical perspective, and this can be an important step in addressing the larger societal systems and inequities that produce social problems.

LINK TO LEARNING

Review the Parable of the Polygons (https://openstax.org/r/76polygons) by Vi Hart and Nicky Case. In it, the authors show how a segregated world where people simply prefer living near other people who are like themselves (a "small individual bias") can re-create and reproduce "large collective bias."

As part of responsible computing, computer scientists must be aware of **technological fix**, which refers to the idea that technologies can solve social problems, but is now often used to critique blind faith in technological solutions to human problems. Unless the process is handled responsibly, the "fix" may cause more problems than it resolves. When considering how to address social and political problems, computer scientists must take care to ensure that they select the appropriate technology to address specific problems.

16 C. Silva, "Facebook ads have a problem. It's called digital redlining," 2022. https://mashable.com/article/facebook-digital-redlining-ads-protected-traits-section-230

To address social problems and advance social good, recall that human-centered computing emphasizes people rather than technologies in the design of computer solutions. A human-centered approach to fair housing might begin by centering local communities directly affected by redlining. Rather than replacing or disrupting the people and organizations already working on a problem, a human-centered approach would center them in the design process as experts. A human-centered approach requires that the designer ask why they are not already working with people in the community impacted by their work.

LINK TO LEARNING

Anatomy of an AI System (https://openstax.org/r/76AIanatomy) illustrates how an AI system like the Amazon Echo does not just involve computer technology, but also involves a vast and deeply interconnected web of human labor, data, and physical resources that are often taken for granted. Evaluation of the negative consequences of technology does not end at the technology itself, but also considers its broad-reaching impacts and implications for people around the world.

Chapter Review

Key Terms

adversarial attack sample input (e.g., an image) that is designed to cause a system to behave problematically
algorithm sequence of precise instructions
artificial intelligence (AI) development of computer functions to perform tasks, such as visual perception and decision-making processes, that usually are performed by human intelligence
big data very large datasets that aren't easily processed using spreadsheets
computational science application of computing concepts and technologies to advance scientific research and practical applications of science knowledge
computer program algorithms that can be run on a computer
computer science (CS) study of the phenomena surrounding computers
computing all phenomena related to computers
data science interdisciplinary field that applies computing toward managing data and extracting information from data
hardware physical, real-world materials that enable computation
human-computer interaction (HCI) subfield of computer science that emphasizes the social aspects of computation
image recognition problem of identifying objects in an image
information science interdisciplinary field studying information technologies and systems as they relate to people, organizations, and societies
memory means of addressing information in a computer by storing it in consistent locations
network various technological devices that are connected and share information
neural network AI algorithm architecture that emphasizes connections between artificial "neurons" whose behavior and values change in response to stimulus or input
processor computer's "brain," that follows instructions from algorithms and processes data
programming language language consisting of symbols and instructions that can be interpreted by a computer
social determination of technology belief that technologies are inherently neutral, and that it is the people who use a technology who ultimately make it "good" or "bad"
software algorithmic principles that determine how results are computed
software engineering subfield of computer science that emphasizes how problems can be solved with computers as well as the practices and processes that can help people design more effective software solutions
spreadsheet data-centric programming environment where data is organized into cells in a table
storage hardware and physical components of a computer that permanently house a computer's data
technological fix idea that technologies can solve social problems, but now often used to critique blind faith in technological solutions to human problems
theoretical computer science mathematical processes behind software
Turing-complete fundamental model for computing results and every computer has the ability to run any algorithm
vacuum tube physical device that works like a light bulb used as memory in early digital computers

Summary

1.1 Computer Science

- Computer science is pervasive in our daily lives, business and industry, scientific research and development, and social change.
- Computer science (CS) is the study of computing, which includes all phenomena related to computers,

such as the Internet. With foundations in engineering and mathematics, computer science focuses on studying algorithms, which are instructions that enable computing. This includes computer hardware and software and the way these are used to process information. Three perspectives on computers include the hardware perspective, software perspective, and theoretical perspective. These perspectives each emphasize different aspects of computation, and they're often centered in undergraduate computer science because of the history of computer science, but there are other perspectives on computer science.

- People have used computer science to advance many more diverse goals beyond making war or making money. Computing was imagined as: a new medium for helping people learn everything; a new technology that could enable anti-racism; a means of enabling global development for peoples across the world. Yet these visions and promises are still taking hold in a world largely focused on the dominant history of computer science.

1.2 Computer Science across the Disciplines

- By contributing tools and resources to handle tasks and improve operations, computer science enables many other fields and areas of research or development.
- Data science is an interdisciplinary field that applies computing to managing data and extracting information from data. Many millions of people engage in data science work by using spreadsheets. Still, data science also often emphasizes larger-scale problems involving big data that are hard to manage using spreadsheets alone.
- Computational science refers to applying computing concepts and technologies to advance scientific research and practical applications of science knowledge. Computer science's emphasis on creating things can help other sciences by, for example, contributing new models or simulations that can enable the discovery of new kinds of scientific knowledge previously inaccessible to scientists.
- Information science is an interdisciplinary field studying information technologies and systems as they relate to people, organizations, and societies. As computing is now so central to information management and information exchange, information science has significant overlap with computer science. Still, it tends to emphasize the social value of information, whereas computer science has (historically) emphasized algorithms and computation over people or information.
- Today, computer science is an interdisciplinary field that contributes to all other fields. Effective computational solutions to research or business problems require combining domain-specific knowledge with computer science concepts from a combination of areas.

1.3 Computer Science and the Future of Society

- Computer science is shaping the future of society. There are three ways in which computer science can shape the future of society: developing foundational technologies, evaluating negative consequences of technologies, and designing technologies for social good.
- As one example of developing foundational technologies, computer science's rapid development of artificial intelligence technologies (and the current trend around neural networks) has enabled many new applications like image recognition. These developments often do not occur in isolation: the popular use of neural networks, for example, depended on new computer architectures and advancements in the Internet (computer networks). Technologies can encode social values: neural networks are designed to learn from big data, so they encode a preference for the contemporary social realities that produced the data.
- As one example of evaluating negative consequences, computer science considers the philosophical and practical limitations of neural networks. Research into adversarial attacks can enable computer scientists to develop more robust neural networks that are safer and more effective.
- As one example of designing technologies for social good, computer science contributes to the research, development, mass production, delivery, and logistics of modern medicine from beginning to end. Yet applications for social good are often embedded in broader social and political dynamics that computer science has difficulty addressing. Even though computer technologies can be designed for social good, they can cause harm when their design processes fail to center on human values and diverse users.

Review Questions

1. What is computer science?
2. What two subjects does computer science combine the foundations of?
   a. math and engineering
   b. math and physics
   c. physics and engineering
   d. math and chemistry
3. What can execute an algorithm?
4. What enables the ENIAC (one of the first digital computers, invented in 1945) to be able to compute anything that can run on modern computers?
   a. Both the ENIAC and modern computers have memory.
   b. Both the ENIAC and modern computers share the same hardware.
   c. Both the ENIAC and modern computers are considered Turing-complete.
   d. Both the ENIAC and modern computers run the same software.
5. What invention was credited as the first calculator?
   a. punch cards
   b. abacus
   c. Difference Machine
   d. ENIAC
6. What term is considered an algorithm that can be run on a computer?
   a. artificial intelligence
   b. algorithm
   c. computer program
   d. programming language
7. Why is computer science often not considered a science?
   a. Computer science does not study natural objects.
   b. Computer science emphasizes the discovery of natural phenomena.
   c. Computer science is spreadsheet-based.
   d. Computer science focuses on computational structures.
8. What is the definition of data science?
   a. a subfield of computer science that emphasizes the social aspects of computation
   b. an interdisciplinary field studying information technologies and systems as they relate to people, organizations, and societies
   c. a subfield of computer science that emphasizes how problems can be solved with computers as well as the practices and processes that can help people design more effective software solutions
   d. an interdisciplinary field that applies computing toward managing data and extracting information from data
9. What term is used to describe a subfield of computer science that emphasizes how a computational problem can be defined in mathematical terms and whether that mathematical problem can be efficiently solved with a computer?
   a. computational science
   b. theoretical computer science
   c. information science
   d. data science
10. What subfield of computer science relates information technology to people and society?

a. computational science
b. theoretical computer science
c. information science
d. data science

11. How does computational science contribute new methods to the study of the sciences?

12. How do information science and computer science compare?

13. What does it mean to say that the various areas of computer science are synergistic?

14. What term is defined as an approach that emphasizes people rather than technologies in the design of computer solutions?
a. human-centered computing
b. neural network
c. social determination of technology
d. technological fix

15. A software application takes an image as an input and analyzes it. This is an example of what?
a. illustrative processing
b. image recognition
c. image generation
d. analytical modeling

16. What are adversarial attacks and why is it important to study them?

17. What is the relationship between artificial intelligence, image recognition, and neural networks?

18. How do neural networks recognize objects in images?

Conceptual Questions

1. Give two definitions of computer science. How do they compare?

2. Explain the concept of theoretical computer science.

3. What is body-syntonic reasoning and how has it affected education?

4. In terms of data science, what is a spreadsheet and why can it be said that by using spreadsheets, people are programming without realizing it?

5. How do discovery and invention differ and how are these involved in computer science?

6. Describe in your own words the difference between data (as in data science) and information (as in information science). How does computer science shape both fields?

7. Give a real-life example that refutes social determination of technology. Your example does not need to involve computing, but it should involve some technology designed by humans.

8. Artificial intelligence approaches are typically used to solve problems that requires specific kinds of "intelligence." Describe a real-life computational problem or application that *does not* need artificial intelligence.

9. Image recognition systems that can detect objects in images enable self-driving cars and many large-scale manufacturing or agricultural operations. Give another example where image recognition could be used as part of a larger system to automate decisions at scale.

Practice Exercises

1. Research where the term *debugging* originated from and why it refers to finding problems in our

programs today.

2. Research some examples of computational models and how they are a part of everyday life.
3. Research how computational models relate to mathematics.
4. Research some examples of how artificial intelligence is used across multiple industries. Summarize at least two different industries and how AI is currently being used.
5. Research ethical issues related to artificial intelligence and provide an example of how artificial intelligence can be misused for unethical and even criminal ways.

Problem Set A

1. Research examples of how modeling and simulations have led to new inventions and discoveries.
2. Research and provide a summary of the difference between artificial intelligence and machine learning.

Problem Set B

1. Research how artificial intelligence and machine learning can improve the accuracy of computational models and lead to cutting-edge technology inventions in the future. Provide a specific example of how AI and ML have been used in this way so far.
2. Research and provide a summary of how machine learning is a subset of artificial intelligence and plays a key role in artificial intelligence systems.

Thought Provokers

1. Corporate social responsibility is the idea that businesses have a responsibility to society, including the areas of environmental responsibility, ethical responsibility, philanthropic responsibility, and economic responsibility. Given the contentious history of computer science and computer technologies, what can businesses (or businesspeople) that wish to employ a "disruptive" computer technology do to ensure corporate social responsibility?
2. Computer technologies like the Internet have changed everyone's lives, regardless of whether they use the Internet directly or not. Yet, with computer technologies, the future is rarely certain. How can a business stay relevant and profitable in the face of new technologies while ensuring corporate social responsibility? In what ways does ensuring corporate social responsibility create economic value and more diverse kinds of value?
3. Give a real-life public policy problem involving a computer technology or dataset and use it to illustrate differences between the fields of data science, information science, and computer science.
4. Mathematics is one of three perspectives that computer scientists use to design, analyze, and evaluate computational structures, systems, and processes. However, mathematics is often regarded as an abstract or "pure" field of study that is not involved in social or political concerns. How might computer science's ability to automate and represent problems in mathematical terms have social or political consequences?
5. The future of society will be shaped by the *philosophy* of computer science and people's purposes, motivations, and political values. Give another philosophy that might influence or affect how computer scientists go about creating solutions.
6. If your organization is interested in artificial intelligence technology to enhance operations, what could you do to ensure the system is designed and implemented in a safe, socially responsible, and just manner?

Labs

1. Explore the Parable of the Polygons (https://openstax.org/r/76polygons). How does computer science

contribute to the simulation? What does the simulation suggest is needed in the world? What are the limitations of the simulation as a model of the much more complicated real world?

2. Explore the Anatomy of an AI System (https://openstax.org/r/76AIanatomy) that breaks down the "human, labor, data and planetary resources" behind an Amazon Echo device. What parts surprised you? Based on your understanding of computer science, what parts are emphasized in the public conscience? What parts are downplayed or minimized? Then, select one surprising aspect and investigate it further.
3. Explore DALL-E (https://openstax.org/r/76DALL-E), a very large neural network created by OpenAI "that creates images from text captions for a wide range of concepts expressible in natural language." If the neural network is learning from English language images and captions on the Internet, what are some of the social risks of this system? How might it encode problematic ideas about marginalized people in society?



2 Computational Thinking and Design Reusability

Figure 2.1 A work environment, such as this modifiable "sp.ace" at Johnson Space Center where furniture can move and any surfaces can be written on, can encourage computational thinking and lead to innovation. (credit: modification of "The sp.ace in Building 29 at Johnson Space Center" by Christopher Gerty, NASA JSC/APPEL Knowledge Services, Public Domain)

Chapter Outline

2.1 Computational Thinking
2.2 Architecting Solutions with Adaptive Design Reuse in Mind
2.3 Evolving Architectures into Useable Products

Introduction

In the rapidly evolving landscape of technology and innovation in various domains, computational thinking promotes design reusability and is a fundamental skill set essential for problem-solving. This chapter illustrates how computational thinking, through practical insights and theoretical frameworks, facilitates the creation of reusable designs that improve the qualities (e.g., scalability, efficiency) of solutions.

Developing innovative solutions to business problems today involves reinventing, rethinking, and rewiring existing business solutions and leveraging the latest technologies to assemble competitive products that solve business problems. It is key to apply computational thinking throughout this process to tackle new problems in specific areas. Computational thinking is a problem-solving and cognitive process rooted in principles derived from computer science. It involves breaking down complex problems into smaller, more manageable parts and devising systematic approaches to solve them.

Adaptive design reuse also makes it possible to quickly assemble business solutions by assembling existing design building blocks that require minimal customizations to be a good match for the problem at hand. TechWorks is a company focused on developing innovative products and services. TechWorks is looking to take advantage of computational thinking and adaptive design reuse to enable the creation of next-generation, secure, super society, intelligent, autonomous business solutions. At TechWorks, a skilled team of engineers, data scientists, and designers is on a mission to revolutionize society. Their goal is to create advanced and secure autonomous business solutions. The team believes in the power of computational thinking and adaptive design reuse for success.

Led by the CIO, the team gathered in a cutting-edge laboratory to tackle challenges in transportation, security,

and automation. They embraced computational thinking by applying algorithms and machine learning to analyze data. Recognizing the efficiency of adaptive design reuse, the team explored successful projects like robotics and self-driving cars for inspiration. These projects have become the foundation for their own innovation. With minimal adjustments, they seamlessly integrate these building blocks into comprehensive solutions such as self-driven cars that can smoothly navigate the city, drones that can monitor public spaces, and robotics that automate tasks. The company plans to bring their vision of the future to life by transforming cities into hubs of interconnected, intelligent systems. Knowing that innovation is a continuous process that requires rapidly evolving solutions, the team faced challenges while implementing their initial prototype. However, they are able to adapt their super society solutions using computational thinking and adaptive design reuse to ensure that they stay ahead of technological advancements. TechWorks is a symbol of the successful integration of forward-thinking strategies to create secure and technologically advanced super society solutions.

2.1 Computational Thinking

Learning Objectives

By the end of this section, you will be able to:

- Define computational thinking
- Discuss computational thinking examples

This chapter presents key aspects of computational thinking, including logical thinking, assessment, decomposition, pattern recognition, abstraction, generalization, componentization, and automation. These elements guide how computer scientists approach problems and create well-designed solution building blocks at both the business and technical levels. Computational thinking often involves a bottom-up approach, focusing on computing in smaller contexts, and seeks to generate innovative solutions by utilizing data structures and algorithms. Additionally, it may make use of existing design building blocks like design patterns and abstract data types to expedite the development of optimal combinations of data structures and algorithms.

What Is Computational Thinking?

The problem-solving and cognitive process, known as **computational thinking**, is rooted in principles derived from computer science. It involves breaking down complex problems into smaller, more manageable parts and devising systematic approaches to solve them. Complex problems are situations that are difficult because they involve many different interrelated parts or factors. These problems can be hard to understand and often don't have simple solutions.

While "computational thinking" is still perceived by some as an abstract concept without a universally accepted definition, its core value is to facilitate the application of separate strategies and tools to address complex problems. In problem-solving, computers play a central role, but their effectiveness centers on a prior comprehension of the problem and its potential solutions. Computational thinking serves as the bridge between the problem and its resolution. It empowers solution designers to navigate the complexity of a given problem, separate its components, and formulate possible solutions. These solutions, once developed, can be communicated in a manner that is comprehensible to both computers and humans, adopting effective problem-solving.

LINK TO LEARNING

Computational thinking serves as the intermediary that helps us read complex problems, formulate solutions, and then express those solutions in a manner that computers, humans, or a collaboration of both can implement. Read this article for a good introduction to computational thinking (https://openstax.org/r/

76CompThinking) from the BBC.

Vision

To further qualify computational thinking, Al Aho of the Columbia University Computer Science Department describes computational thinking as "the thought processes involved in formulating problems so their solutions can be represented as computational steps and algorithms." Jeannette Wing, also of Columbia University, brought the idea of computational thinking to prominence in a paper she wrote in 2006 while at Carnegie Mellon University. She believes that computational thinking details the mental acts needed to compute a solution to a problem either by human actions or machine. Computational thinking encompasses a collection of methods and approaches for resolving (and acquiring the skills to resolve) complex challenges, closely aligned with mathematical thinking through its utilization of abstraction, generalization, modeling, and measurement (Figure 2.2). However, it differentiates itself by being more definitely aware than mathematics alone in its capacity for computation and the potential advantages it offers.

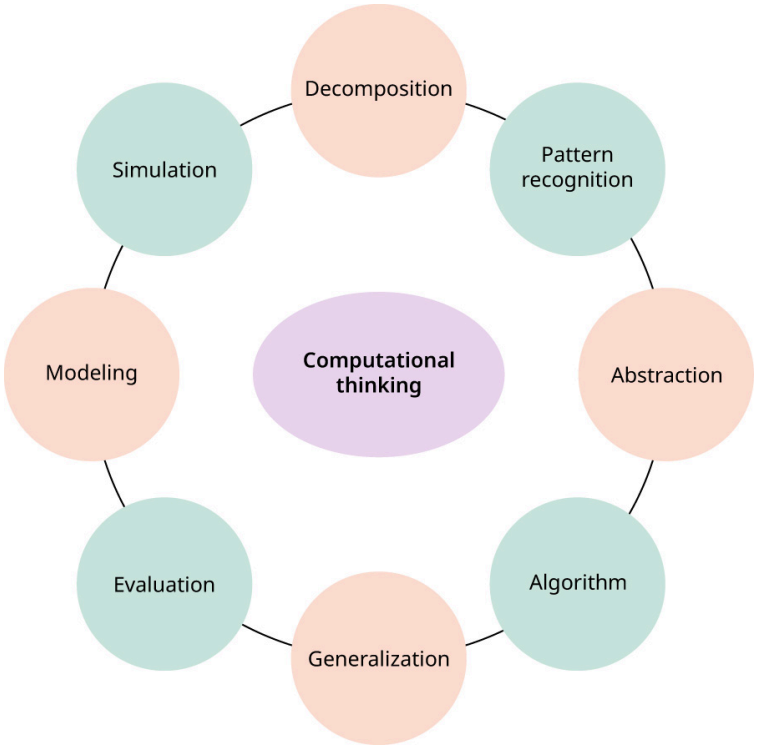


Figure 2.2 This diagram illustrates the main components of computational thinking. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Critical thinking is an important skill that can help with computational thinking. It boils down to understanding concepts rather than just mastering technical details for using software, prioritizing comprehension over rote learning. It's a core skill, not an extra burden on a curriculum checklist, and it uniquely involves humans, not computers, blending problem-solving and critical thinking. Critical thinking focuses on ideas, not tangible items, applying advanced thinking to devise solutions. Critical thinking is essential for everyone and is, comparable to foundational abilities like reading, writing, and arithmetic.

Computational Thinking Concepts

The description provided by the International Society for Technology in Education (ISTE) outlines the key components and dispositions of computational thinking. Let's explore each characteristic in more detail:

- Decomposition: The analytical process of breaking down complex concepts or problems into smaller parts

is called **decomposition**. This approach helps analyze and solve problems more effectively.

- Pattern recognition (logically organizing and analyzing data): Computational thinking emphasizes the logical organization and analysis of data. This includes the ability to structure information in a way that facilitates effective problem-solving.
- Representing data through abstractions: An **abstraction** is a simplified representation of complex systems or phenomena. Computational thinking involves representing data through an abstraction, such as a simulation, which uses models as surrogates for real systems.
- Automation through algorithmic thinking: Using a program or computer application to perform repetitive tasks or calculations is considered **automation**.
- Identification, analysis, and implementation of solutions: Computational thinking emphasizes identification, analysis, and implementation of potential solutions to achieve optimal efficiency and effectiveness through a combination of steps and resources.
- Generalization and transferability: Generalizing and transferring this problem-solving process across a wide variety of problems showcases the adaptability and applicability of computational thinking.

These abilities are supported and enriched by fundamental abilities integral to computational thinking. These abilities involve the following characteristics: confidence in navigating complexity, resilience in tackling challenging problems, an acceptance of ambiguity, adeptness in addressing open-ended issues, and proficiency in collaborating with others to attain shared objectives or solutions. Another illustration of computational thinking is the three As, which is organized into three phases, as visualized in Figure 2.3:

1. Abstraction: The initial step involves problem formulation.
2. Automation: Next, the focus shifts to expressing the solution.
3. Analysis: Finally, the process encompasses solution execution and evaluation.

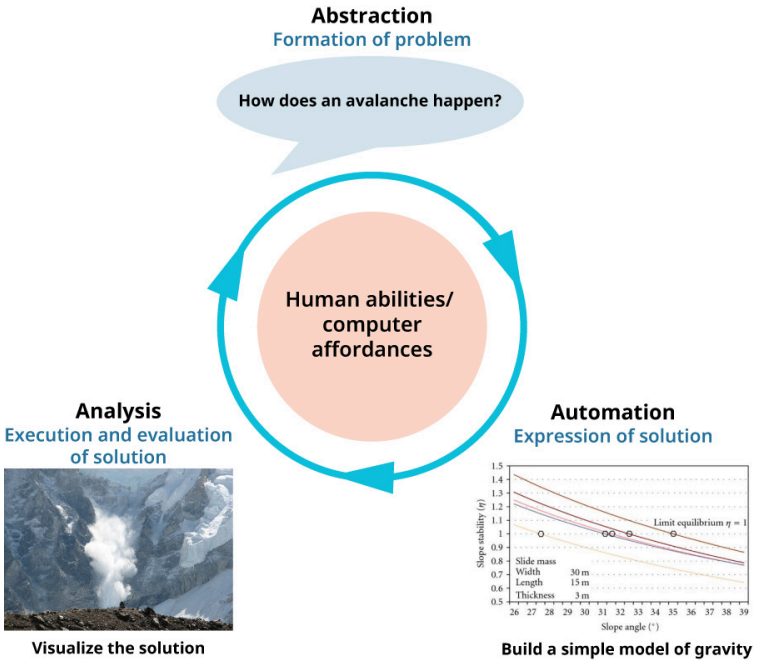


Figure 2.3 The three As—abstraction, automation, analysis—illustrate the power of computational thinking. (credit photo: modification of "Avalanche on Everest" by Chagai/Wikimedia Commons, Public Domain; credit graph: modification of "Slope stability calculation for a model landslide" by B. Terhost and Bodo Damm/Journal of Geological Research, CC BY)

Computational Thinking Techniques

In today's technology world, mastering computational thinking techniques is important. These techniques offer a systematic way to solve problems using tools like data structures, which are like containers used to organize and store data efficiently in a computer. They define how data is logically arranged and manipulated, making it easier to access and work with information in algorithms and programs. There are four key

techniques (cornerstones) to computational thinking, as illustrated in Figure 2.4:

- Decomposition is a fundamental concept in computational thinking, representing the process of systematically breaking down a complex problem or system into smaller, more manageable parts or subproblems. By breaking down complexity into simpler elements, decomposition promotes a more organized approach to problem-solving.
- Logical thinking and pattern recognition is a computational thinking technique that involves the process of identifying similarities among and within problems. This computational thinking technique emphasizes the ability to recognize recurring structures, relationships, or sequences in various problem-solving situations.
- Abstraction is a computational thinking technique that centers on focusing on important information while ignoring irrelevant details. This technique enables a clearer understanding of the core issues.
- Algorithms are like detailed sets of instructions for solving a problem step-by-step. They help break down complex tasks into manageable actions, ensuring a clear path to problem-solving.

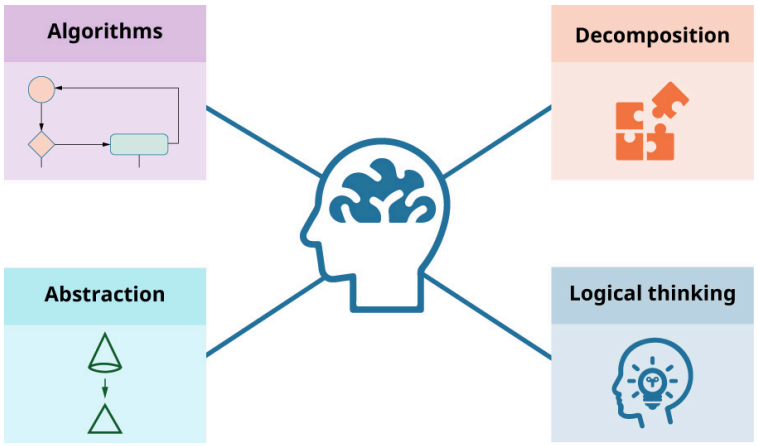


Figure 2.4 Users can explore the essence of computational thinking through decomposition, logical thinking, abstraction, and algorithms. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

In addition to the four techniques, computational thinking involves essential steps such as testing and debugging. Testing is crucial for uncovering errors within the step-by-step instructions or algorithms employed to tackle a problem. On the other hand, **debugging** entails identifying and rectifying issues within the code.

A programmer is someone who writes instructions for a computer to follow. A typical example is that of a programmer who gives instructions to a robot and tells it to make a jam sandwich. In this case, applying computational techniques to give instructions to the robot entails the following techniques: decomposition, logical thinking and pattern recognition, abstraction, and algorithms. These techniques are explained in the following subsections as they apply to the jam sandwich example.

TECHNOLOGY IN EVERYDAY LIFE

Traffic Accident Data

Analyzing data involves collecting and cleaning information, exploring patterns through visual and statistical methods, and forming hypotheses. Statistical analysis and visualization are used to draw conclusions, and findings are interpreted and communicated in reports or presentations to help in the process of decision-making. Analyze the patterns and trends in traffic accident data to understand the prevalence of road injuries and fatalities, and examine the progression of traffic incidents over time. To enhance road safety measures and policies, you should apply computational thinking skills to identify recurring patterns and abstract the most crucial information from the data. By extracting valuable insights,

you can contribute to the development and refinement of strategies that effectively improve road safety.

Decomposition

Decomposition involves solving a complex problem by breaking it up into smaller, more manageable tasks. Decomposition enables the consideration of various components essential for solving a seemingly complex task, allowing it to be redefined into a more manageable problem. In the case of the jam sandwich example, decomposition involves identifying all the required ingredients and the steps the robot must take to successfully create a jam sandwich.

Logical Thinking and Pattern Recognition

Pattern recognition makes it possible to group all the different features considered in decomposition into categories. In the jam sandwich example, pattern recognition leads to grouping the various things identified via decomposition into categories, in this case, ingredients, equipment, and actions. Therefore, applying decomposition and pattern recognition will lead to thinking of as many things as possible that are required to make a jam sandwich. The more things that can be thought of (i.e., ingredients, equipment, and actions), the clearer the instructions will be. A first attempt at decomposition and pattern recognition is summarized in Table 2.1.

Ingredients	Equipment	Actions
Bread	Plate	Repeat *x* times
Jam	Knife	Left hand (LH)
Butter		Right hand (RH)
		Pick up
		Unscrew

Table 2.1 Logical Thinking and Pattern Recognition Example The jam sandwich pattern recognition defines the ingredients, equipment, and actions needed for completion.

The process of identifying patterns typically requires logical thinking such as inductive or deductive reasoning. Inductive reasoning makes it possible to go from specific examples to general principles. For example, recognizing that dividing any number by 1 results in the original number leads to the broader conclusion that holds true for any number. Similarly, understanding that the sum of two odd numbers yields an even number leads to the generalization that adding two odd numbers always results in an even number. Inductive reasoning turns an observation into a pattern, which allows making a tentative hypothesis that can be turned into a theory. Deductive reasoning is the process of drawing valid conclusions from premises given the fact that it is not possible for the premises to be true and the conclusion to be false. A traditional example illustrates how the premises “all men are mortal” and “Socrates is a man” lead to the deductively correct conclusion that “Socrates is mortal.”

TECHNOLOGY IN EVERYDAY LIFE

Computational Thinking in Our Life

Computational thinking is a method of problem-solving that is extremely useful in everyday life. It involves breaking down complex issues into manageable parts, identifying patterns, extracting essential information, and devising systematic solutions. This process not only applies to technical fields, but also to everyday situations.

For example, imagine someone trying to manage their monthly expenses within a tight budget. Here's how you might apply computational thinking to this common problem of managing a monthly budget:

1. Decomposition: Break down the financial challenge into different categories such as rent, groceries, utilities, and entertainment.
2. Pattern recognition: Analyze past spending to identify patterns.
3. Abstraction: Focus on key areas where costs can be reduced.
4. Algorithmic thinking: Develop a systematic approach to allocate monthly income.

By using computational thinking, you can manage your finances more effectively, ensuring they cover essential costs while maximizing their savings.

Abstraction

Abstraction makes it possible to pull out the important details and identify principles that apply to other problems or situations. When applying abstraction, it may be useful to write down some notes or draw diagrams to help understand how to resolve the problem. In the jam sandwich example, abstraction means forming an idea of what the sandwich should look like. To apply abstraction here, you would create a model or draw a picture representing the final appearance of the jam sandwich once it is made. This simplifies the details, providing a clearer image of the desired outcome. Simple tools like the Windows Paint program can be used to do this, as shown in Figure 2.5.

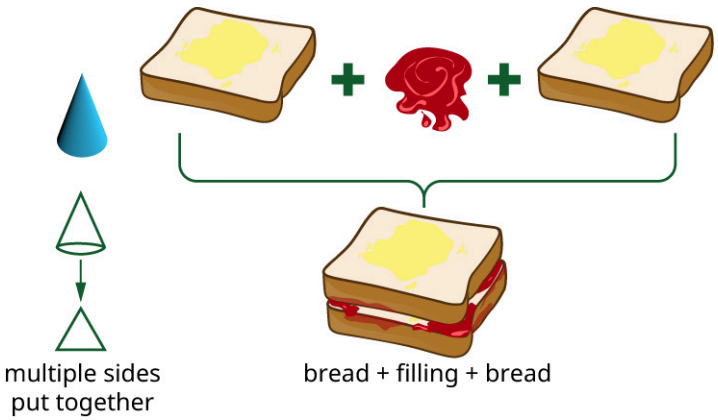


Figure 2.5 This jam sandwich abstraction example illustrates what the final product should look like. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

In technology, data are represented at different levels of abstraction to simplify user interaction and manage complex operations efficiently. Users interact with a web application through a straightforward interface, like requesting help from a GenAI tool, without seeing the underlying complexity. This GenAI prompt is then processed by the application's logic, which validates and directs it appropriately, often invisibly to the user. Finally, at the back end, the prompt is processed and a GenAI-generated response is provided. Each layer of abstraction serves a separate role, making the entire process efficient for both the user and the system (Figure 2.6).

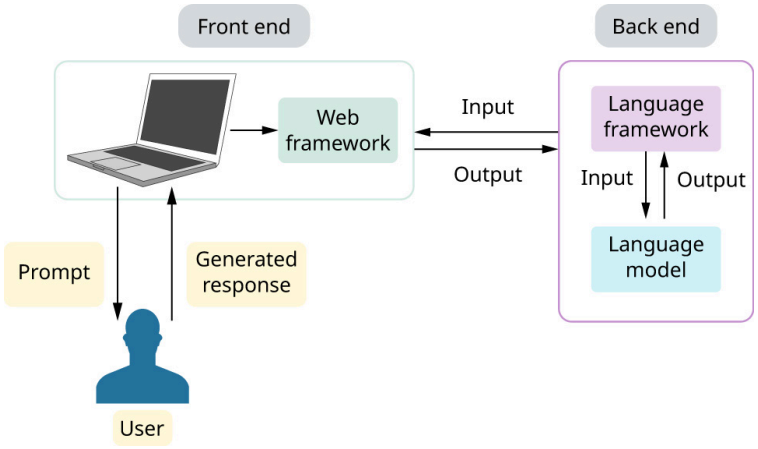


Figure 2.6 When using GenAI, a user interacts with the interface while the application processes the prompt with layers of abstraction on the back end. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Algorithm

An algorithm is a sequence of steps/instructions that must be followed in a specific order to solve a problem. Algorithms make it possible to describe a solution to a problem by writing down the instructions that are required to solve the problem. Computer programs typically execute algorithms to perform certain tasks. In the jam sandwich example, the algorithm technique is about writing instructions that the robot can follow to make the jam sandwich. As you will learn in Chapter 3 Data Structures and Algorithms, algorithms are most commonly written as either pseudocode or a flowchart. An outline of the logic of algorithms using a combination of language and high-level programming concepts is called **pseudocode**. Each step is shown in a clearly ordered, written structure. A **flowchart** clearly shows the flow and direction of decisions in a visual way using a diagram. Either way is fine, and it is a matter of personal preference. Basic templates for the flowchart and pseudocode are in Figure 2.7.

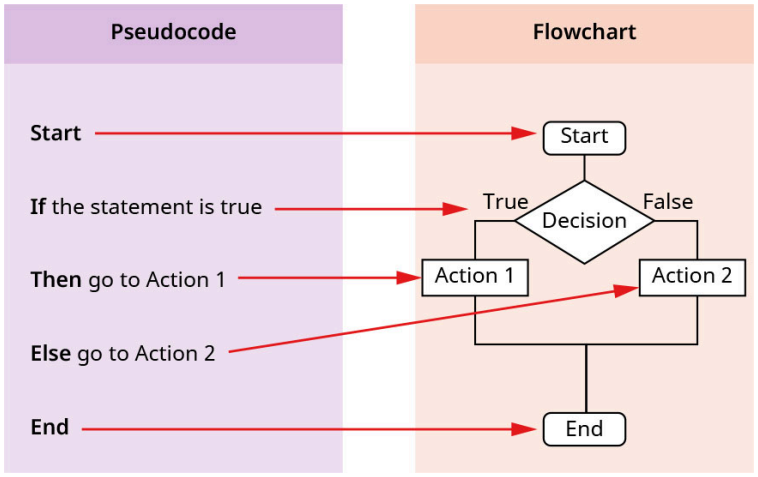


Figure 2.7 Pseudocode lists each step, while a flowchart visually outlines the process of decision-making. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Writing algorithms requires practice. Not everyone likes butter in their jam sandwich. The robot needs a method of making sure it adds or does not add butter, depending on preferences. It is therefore necessary to account for the following steps in the pseudocode and flowchart:

1. Ask whether there should be butter on the bread.
2. Either spread butter on the bread,
3. Or, do not use butter.

These steps can be added as actions in the table previously shown and expressed as steps in the pseudocode

using programming keywords such as INPUT, OUTPUT, IF, THEN, ELSE, and START. The corresponding instructions can then be converted into a flowchart using the symbols in Figure 2.8.

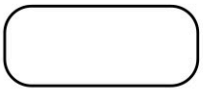
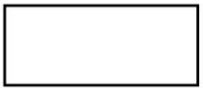
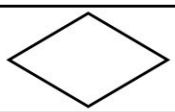
Symbol	Instruction
	Start/end
	Task (e.g., spread jam)
	Decision (e.g., do you want butter?)
	Direction of flow

Figure 2.8 The symbols used in a flowchart are associated with their instructions. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Algorithm Execution Model Patterns

Various patterns of execution models may be used to step through the instructions provided in an algorithm. So far, we have only considered the traditional sequential (i.e., step-by-step) execution model for algorithm instructions. However, it is also possible to leverage parallelism/concurrency and recursion as alternative models to drive the execution of algorithms' instructions.

Parallel/concurrent execution models are typically used to optimize algorithm execution efficiency. As an example, if you and a friend are buying tickets for a movie and there are three independent lines, you may opt for a parallel processing model of execution by having you and your friend join two separate lines to buy the tickets. In that case, you are guaranteed to be able to obtain the tickets quicker assuming one of the lines operating in parallel with the other ends up serving customers faster, which is most often the case. Note that executing the same algorithm simultaneously on a computer may not be possible if you only have one central processing unit (CPU) in your machine. In that case, you can simulate parallelism by having the operating system running on the machine execute the two algorithms concurrently as separate tasks while sharing the single processor resources. This approach is less efficient than true parallelism. More detail on the differences between concurrency and parallelism will be provided in Chapter 4 Linguistic Realization of Algorithms: Low-Level Programming Languages.

Recursive models of execution provide another elegant and effective alternative to the traditional sequential model of execution. The problem-solving technique where a process calls itself in order to solve smaller instances of the same problem is called **recursion**. It can be a powerful tool in programming because it allows for elegant solutions to complex problems by breaking them down into smaller, more manageable parts. By leveraging recursion, programmers can write concise and efficient code to solve a wide range of problems.

One of the key advantages of recursion is its ability to handle complex tasks with minimal code. Instead of writing lengthy iterative loops to solve repetitive tasks, recursion allows programmers to define a process that calls itself with modified input parameters, effectively reducing the amount of code needed. However, it's essential to be cautious when using recursion, as improper implementation can lead to stack overflow errors due to excessive process calls. Programmers should ensure that recursive processes have proper base cases to terminate the recursion and avoid infinite loops. Example:

```
#include <iostream>
using namespace std;
```

```
int recursiveSum (int x) {
  // Base case
  if (x == 0) {
    return 0;
  } else {
    // Recursive step
    return x + recursiveSum (x - 1);
  }
}
int main() {
  cout << recursiveSum (10);
  // Answer is 55
  return 0;
}
```

In this scenario, the process involves gradually adding values to the total variable as you iterate through a loop. However, a different approach involves leveraging computational thinking to deconstruct the problem, breaking it down into smaller subcomponents. This method tackles these subcomponents individually to address the overarching issue. When these smaller parts represent scaled-down versions of the original problem, recursion becomes a valuable tool.

In practical scenarios, recursion often manifests as a **function**, which is a set of commands that can be repeatedly executed. It may accept an input and may return an output. The base case represents the function's most straightforward operation for a given input. To effectively implement recursion, two primary steps must be followed: (a) identify the base case, and (b) outline the recursive steps. In the context of a recursive function, when *n* is 0, the cumulative sum from 0 to 0 is intuitively 0, representing the most fundamental subproblem of the main issue. Armed with this base case, you can commence crafting the initial part of the function.

```
int recursiveSum (int x) {
  // Base case
  if (x == 0)
    return 0;
}
```

Recursion operates through a process of simplification, progressively reducing the value of *x* until it meets the base condition, where *x* equals 0. This technique presents an alternative method, offering a refined and effective algorithmic solution for the current problem:

```
#include <iostream>
using namespace std;


int recursiveSum (int x) {
  // Base case
  if (x == 0) {
    return 0;
  }
  else {
```

```
    // Recursive step
    return x + recursiveSum (x - 1);
  }
}
int main() {
cout << recursiveSum(10); // Output will be the sum = 55.
  return 0;
}
```

While it looks like recursion amounts to calling the same function repeatedly, it is only partially true, and you should not think about it that way. What happens is much more than repeating the call of a function. It is more useful to think of it as a chain of deferred operations. These deferred operations are not visible in your code or your output—they are in memory. The program needs to hold them somehow, to be able to execute them at the end. In fact, if you had not specified the base case, the recursive process would never end. Figure 2.9 illustrates a flowchart for an iterative solution that adds *N* numbers.

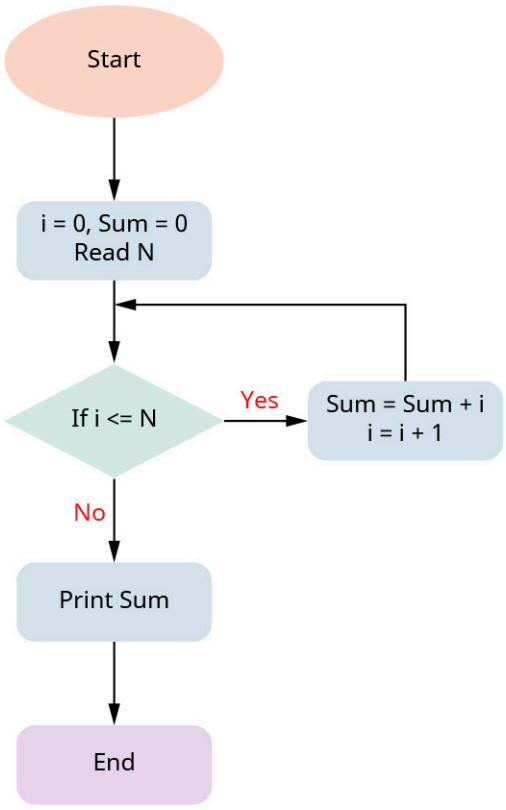


Figure 2.9 A flowchart represents an iterative solution for adding *N* numbers. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

CONCEPTS IN PRACTICE

Computational Thinking for Chess Problem-Solving

Computers can be used to help us solve problems. However, before a problem can be tackled, the problem itself and how it could be solved need to be understood. Computational thinking transcends mere programming; it doesn't equate to thinking in the binary fashion of computers, as they fundamentally lack the capacity for thought. Rather, while programming is the craft of instructing a computer on the actions to execute, computational thinking empowers you to meticulously determine what those instructions should

be. Take, for instance, the strategic gameplay involved in chess. To excel in a chess match, a player must:

- Understand the unique movements and strategic values of each piece, recognizing how each can be maneuvered to control the board.
- Visualize the board's layout, identifying potential threats and opportunities, and planning moves several steps ahead to secure an advantageous position.
- Recognize patterns from previous games, understanding common tactics and counters, to formulate a robust, adaptable strategy.

In devising a winning strategy, computational thinking is the underpinning framework:

- The complex game is dissected into smaller, more manageable components (e.g., the function of each chess piece, the state of the board)—this is decomposition.
- Attention is concentrated on pivotal elements that influence the game's outcome, such as the positioning of key pieces and the opponent's tendencies, sidelining less critical factors—this is an abstraction.
- Drawing from prior knowledge and experiences in similar scenarios, a step-by-step approach is developed to navigate through the game—this is algorithmic thinking.

Should you venture into developing your own chess program or strategy, these are precisely the types of considerations you would deliberate on and resolve before actual programming.

Testing and Debugging

Testing and debugging are techniques used to identify flaws in algorithms and defects in code to be able to correct them. Test cases rely on providing specific input data to check whether a software program functions correctly and meets its designed requirements. Test cases need to be identified to drive tests. If a test associated with a test case fails, debugging needs to be conducted to identify the source of the problem and correct it. In other words, debugging is about locating and fixing defects (i.e., bugs) in algorithms and processes to make them behave as expected. In programming, everyone makes mistakes, they are part of the learning process. The important thing is to identify the mistake and work out how to overcome it. There are those who feel that the deepest learning takes place when mistakes are made.

In the jam sandwich algorithm, testing can be facilitated by taking turns to play the role of the programmer who gives instructions as well as the robot. If you are a programmer, your job is to read out the instructions and follow each step. You can choose to follow your pseudocode or your flowchart. Each instruction becomes a test case, and the test succeeds if the robot can follow every instruction exactly and successfully. In the alternative, you will need to debug the instruction to identify the source of the problem and correct it. Table 2.2 can be used to record problems encountered and improvements that need to be made.

Test Case	Input	Problem	Expected Outcomes	Observed Outcomes	Improvement	Responsible User
1.						
2.						

Table 2.2 Sample Table for Recording Test Problems and Improvements

INDUSTRY SPOTLIGHT

DNA Sequencing

Computational thinking is important in every industry today. In DNA sequencing, computational thinking helps manage the massive and complex data involved. It starts by breaking the large DNA sequence into smaller pieces. Then, it involves identifying patterns or sequences within these pieces, which might indicate genetic information like the presence of certain genes. The focus is on the most relevant parts of the sequence, discarding unnecessary data to concentrate on potentially significant genetic regions. Finally, refined algorithms process and reconstruct the original sequence to identify genetic variations. This approach is used for efficiently handling massive datasets in DNA sequencing and extracting meaningful insights. The parts of computational thinking (CT) can be identified and highlighted in the process of DNA sequencing, a complex task within the field of genomics:

- Decomposition: Break down the DNA sequencing process into specific steps such as sample collection and DNA extraction.
- Pattern recognition: Identify similarities in DNA sequences that could indicate genetic traits or anomalies.
- Abstraction: Focus on the essential parts of the genetic information that are relevant for the study at hand.
- Algorithms: Create step-by-step protocols for each part of the sequencing process.
- Logical thinking: Determine the most accurate methods for sequencing based on the type of sample and the required depth of sequence analysis.
- Evaluation: Assess the quality and accuracy of the sequencing data obtained.
- Debugging: Identify issues that may arise during the sequencing process.

Practical Computational Thinking Examples

Here are different real-life scenarios of practical applications of computational thinking with suggested solution approaches to provide problem-solving and decision-making:

- Organizing a city's recycling program to maximize efficiency. How can you ensure the most effective collection routes and times?
- Solution: Use a route optimization algorithm to analyze and plan the most efficient paths for collection trucks, considering factors like distance and traffic patterns.

- Planning the layout of a community garden to optimize space and sunlight exposure for different plant types. How do you decide where to plant each type of vegetable or flower?
- Solution: Employ a simulation algorithm that models sunlight patterns, plant growth rates, and space requirements to design a garden layout that maximizes space and plant health.

- Creating a schedule for a multistage music festival to minimize overlaps and ensure a smooth flow of audiences. How do you schedule the performances across different stages?
- Solution: Implement a scheduling algorithm that considers audience preferences, artist availability, and stage logistics to create a timetable that maximizes attendee satisfaction and minimizes conflicts.

- Determining the most efficient way to allocate computer resources in a cloud computing environment to handle varying user demands. How do you manage the computational load?
- Solution: Use load balancing algorithms to distribute tasks across servers dynamically, ensuring optimal resource utilization and maintaining system performance.

2.2 Architecting Solutions with Adaptive Design Reuse in Mind

Learning Objectives

By the end of this section, you will be able to:

- Describe how business solutions design heuristics and how patterns are used
- Discuss the role of enterprise architecture (EA) and related architecture domains
- Differentiate between enterprise and solution architecture

While computational thinking commonly employs a bottom-up strategy for crafting well-structured components, adaptive design reuse adopts a top-down methodology, emphasizing the creation and assembly of business solutions by combining existing design components. A **design component** is a reusable element within a larger system that serves a specific purpose. These components, akin to low-level solution building blocks, necessitate minimal modifications. This approach, often termed *computing in the large*, diverges from individual algorithmic instructions. Instead of composing instructions for the execution of specific actions through computational thinking's decomposition, adaptive design reuse identifies fitting components based on the articulated requirements of the business solution's potential users, commonly referred to as stakeholders. The method(s) used to identify these needs will be discussed in detail in Chapter 9 Software Engineering. While adaptive design reuse typically considers algorithmic designs as building blocks, it uses similar techniques (i.e., decomposition, logical thinking and pattern recognition, abstraction/generalization, componentization, testing, and debugging) to identify and reuse building blocks.

The structural designs that stem from the top-down adaptive design reuse approach to business solution design are referred to as *business solutions architecture models*. A business **solution architecture** is a structural design that is meant to address the needs of prospective solution users. When a structural design meets these needs, it is considered "architecturally sound" for the problem at hand. Furthermore, to accelerate the putting together of complete solutions, the adaptive design reuse approach relies on architectural models or component assemblies that were developed from previous designs. When the granularity of these architectural models is at the level of subsystems, they are referred to as system family architectures or architectural patterns. An **architectural pattern** is a reusable solution to a recurring problem in software architecture design. Chapter 10 Enterprise and Solution Architectures Management provides more detail about the various levels of patterns and how to organize them and bookkeep them into pattern catalogs to facilitate reuse.

Business Solutions Design Patterns

Business solutions are strategies/systems created to solve specific challenges in a business. They aim to make operations more efficient, improve decision-making, and contribute to overall success. Creating business solutions is a multifaceted and intricate process, demanding knowledge in different technological domains and the relevant business sector. A crucial step in this procedure is outlining the solution's architecture, serving as a master blueprint, and guiding the entire design and implementation process. A **blueprint** is a detailed plan or design that outlines the structure, components, and specifications of a building, product, system, or process.

Computational thinking and adaptive design reuse are simply methods used to bootstrap and/or accelerate the design of business solutions by providing a comprehensive set of methods for developing software solutions to business problems—for example, gathering solutions needs, building, and deploying turnkey solutions. These solutions will be discussed in detail in Chapter 9 Software Engineering.

As explained earlier, the computational thinking and adaptive design reuse approach only provide high-level techniques to help create or reuse components. As a result, one of the typical criticisms is that these approaches are too vague, as it is sometimes not clear how they differ from other forms of thought. This is why experts at applying these methods are usually seasoned architects who can instinctively recognize when

various types of components may be reused. When you ask Thoughtworks' enterprise architecture expert Martin Fowler why a particular business solution architecture is sound in a given context, he will often reply that it simply "smells good" to him.

Rather than looking at computational thinking and adaptive design reuse as a best practice set of techniques, it is best to consider them as process patterns that may be applied and adapted to help create a business solution **architecture model**, which represents the content of the application architecture. Process patterns may, in turn, leverage other process patterns that are more focused. When process patterns that are an inherent part of the process of creating a work product become rules of thumb, they are referred to as heuristics.

INDUSTRY SPOTLIGHT

Case Study: Making Online Shopping Easier with Smart Design

Imagine an online store called SwiftShop. They had a problem. Even though they had lots of great products to buy, people were leaving their website without buying anything. It was like having a store full of customers who walked in and out without buying anything! The business challenge is SwiftShop wanted to make shopping on their website easier and more fun. They wanted people to stay longer, buy more, and come back again and again. SwiftShop decided to use smart design tricks to fix their website and make it super easy to shop. Here's how they did it:

- User interface (UI) design patterns: Breadcrumb navigation: SwiftShop added breadcrumb trails so shoppers could easily see where they were on the website and find their way back if they got lost.
- Progress indicators: During checkout, SwiftShop put in progress bars so shoppers knew how far along they were in the buying process.
- Information architecture (IA) design patterns: Card sorting: SwiftShop asked real shoppers to help organize their products. By sorting cards and asking people how they'd group things, SwiftShop made it easier to find what customers were looking for.
- Faceted search: SwiftShop added filters so shoppers could narrow down their searches by attributes like price, size, and brand.
- Interaction design (IxD) patterns: One-click purchase: To speed up the buying process, SwiftShop let registered users buy with just one click.
- Personalized recommendations: SwiftShop used machine learning algorithms to suggest products that customers might like based on what they've bought before.
- Results: After making these changes, SwiftShop saw great results:
  - More people buying: With the new features, more people ended up buying products from SwiftShop.
  - Happier shoppers: Customers spent more time on the website, which meant they liked shopping there more.
  - More repeat customers: Because it was easier to shop, people kept coming back to SwiftShop again and again.

So, by using smart design tricks like breadcrumb navigation, progress indicators, card sorting, faceted search, one-click purchase, and personalized recommendations, SwiftShop made their online store a better place to shop.

Layering and Componentization

Two heuristics are inherent to the design of business solutions and the creation of business solution architectures. These heuristics are known as layering and componentization and can be thought of as design approaches followed with an intent of architectural concerns.

Componentization has already been introduced as a computational thinking and adaptive design reuse technique.[1] Layering in business solution architecture involves creating distinct layers that abstract specific aspects of the overall architecture. This **heuristic** enables the separation of concerns between layers and improves modularity. Each layer comprises components, and a layer may consist of multiple components. This hierarchical structure helps organize and separate different functionalities or responsibilities within the architecture, making it easier to manage and understand.

The layering strategy is based on the concept of separating different concerns. This method structures software design into distinct, stacked layers, with each layer tasked with specific functions. Commonly, business solution architectures are organized into three main layers: the presentation, business logic, and data management layers. The **presentation layer** is the user's touchpoint, handling the user interface (UI) and delivering the **user experience (UX)**, which is the overall experience that a person has when interacting with a product, service, or system. The **business logic layer** holds the business logic for the business solution or application, separating the UI/UX from the business-centric computations. This separation affords the flexibility to adjust business operations as needs evolve without disrupting other system components. Although not tied to a specific domain, the **data management layer** is responsible for interacting with persistent storage systems like databases and various data processing mechanisms. In a layered architecture, information and commands flow through each layer, enhancing the design's abstraction level, which denotes the granularity or detail at which a system is represented. Despite its structured approach and abstraction benefits, software designed in this layered manner might lean toward a monolithic build, potentially making modifications challenging. Layered architecture can lead to tightly connected layers in software, making it difficult to change one part without affecting others. This tight connection can also make it harder to update or expand the software.

A **monolithic structure** is a type of system or application where all the parts are closely combined into one single unit. This setup means that everything from the user interface to data handling and processing is interconnected within one big software application. While this can make the system easier to set up and manage initially, it also means that changing one part can require changes throughout the whole system, making it less flexible. As illustrated in Figure 2.10, there are many more recent variations of layered architectures that also provide complementary capabilities, such as tiered architectures, service-oriented architectures (SOA), and microservices architectures.[2]

1 Componentization was used initially in component-based business solution architectures that were derived from the Object Management Group's Object Management Architecture (OMA), including CORBA 3, JEE, DCOM, and COM+.

2 Per an article written by Thoughtworks Martin Fowler in 2014, the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies.

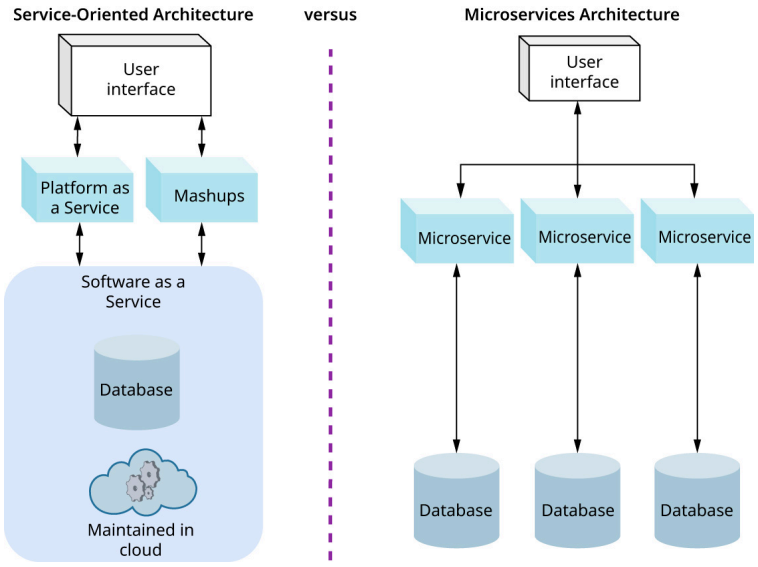


Figure 2.10 The different layered architectures include tiered architectures, service-oriented architectures (SOA), and microservices architectures. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Detailed coverage of these specific architectures is not part of the scope of this chapter. They will be discussed in more detail in multiple later chapters. It's fundamental to understand that each software architecture model is crafted to address the key challenges found in its preceding models. Being well-versed in various architectural approaches equips you to devise a robust and effective architecture for your specific project. While no software architecture can claim absolute perfection, an approach can be deemed highly suitable or "relatively perfect" if it aligns well with the specific requirements and goals of your current project.

Enterprise-Level Architecture Domains

Enterprise-level architecture encompasses various domains that define the structure, components, and operations of an entire organization. These domains provide a comprehensive framework for managing an enterprise.

Up to this point, our efforts have been centered on employing the adaptive design reuse methodology to construct business solution architectures tailored to individual projects. Within this scope, we aim to facilitate the conversion of solution requirements into a cohesive solution concept, comprehensive business and IT system outlines, and a collection of implementation activities, essentially forming the project plan. However, because the adaptive design reuse approach is top-down, it is possible to start applying it at a higher level. The enterprise level is typically the highest level of an organization, and it covers all strategic and tactical functions. An enterprise often spans multiple organizations.

Enterprise architecture (EA) emerged at the beginning of the information age in the 1970s, and various enterprise architecture frameworks (EAFs) were developed and used over time. The Open Group Architecture Framework (TOGAF) is one such framework. According to TOGAF, an **EA domain** represents business, data, application, and technology architectures. While specific frameworks may vary, Table 2.3 illustrates the common enterprise-level architecture domains.

Architecture	Purpose	Components
Business architecture	Defines the organization's business strategy, goals, processes, and functions	Business models, processes, capabilities, and organizational structure
Information architecture	Manages data and information assets	Data models, information flow diagrams, data governance, data standards, and metadata
Application architecture	Designs and organizes software applications	Application portfolio, application integration, and interface design
Technology architecture	Specifies the hardware, software, and technology infrastructure	Servers, networks, databases, cloud services, security protocols
Security architecture	Ensures the protection of information assets, systems, and networks	Security policies, access controls, and encryption mechanisms
Integration architecture	Facilitates seamless communication and data exchange	Middleware, messaging systems, and integration patterns
Process architecture	Defines and optimizes business processes	Process models, workflow diagrams, and performance metrics

Table 2.3 Common Enterprise-Level Architecture Domains

EA adopts a holistic perspective, treating the enterprise as a unified system or a system of systems. Its primary aim is to integrate disparate processes, both manual and automated, into a cohesive environment that is responsive to change and aligned with the enterprise's business strategy. EA involves designing comprehensive systems to support business processes, going beyond individual software or technology systems. By standardizing and integrating fundamental processes across various business divisions, EA facilitates enterprise-wide alignment and optimization of operations.

Solution architecture complements EA by tailoring specific system solutions to meet defined business and technological needs. While EA focuses on defining the present situation and future goals of the organization, solution architecture ensures the timely availability of suitable systems to fulfill business requirements.

TOGAF architectures promote the reuse of building blocks but lack a prescriptive method for managing them. Adaptive design reuse becomes valuable in this context, involving understanding enterprise architectures and adapting preexisting models and implementations. Detailed management of EA, solution architectures, and leveraging EAFs is explored further in Chapter 10 Enterprise and Solution Architectures Management.

Enterprise Business Architecture and Model

The enterprise business architecture (EBA) is a comprehensive framework that defines the structure and operation of an entire organization. It is related to corporate business and the documents and diagrams that describe the architectural structure of that business. It characterizes the processes, organization, and location aspects of the EBA at hand. EBAs usually consist of different business capabilities grouped into categories, as illustrated in Figure 2.11. In this example, sample categories include product, sales, and service. The business capabilities shown under the various categories typically have business models of their own that are part of the overall organization's business model. Note that it is clear from this diagram that the layering and

componentization heuristics play an important role in structuring capability models.



Figure 2.11 The EBA framework categories, including product, sales, and service, support the business organizations. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Business architecture modeling helps extract the overall business model of an organization's capability, which includes the set of underlying processes necessary to operate the various capabilities of the organization.

A **business model** is a framework that outlines how a business creates, delivers, and captures value. It covers the way a company generates revenue, identifies its target audience, and defines the products/services it offers. The business model includes the organization, location, and process model, as described in the following sections. It may include additional models, such as the financial model, which will not be covered here. To illustrate what a business model entails practically, we will focus on a typical financial instruments trading capability, which could be one of the capabilities in the overall business model of an organization operating in the global markets industry. The organization, location, and process models are typically represented at a logical level of abstraction, which is the most detailed representation for these types of models. Note again that layering and componentization heuristics play an important role in structuring these models.

Organizational Model

The organizational model is the structure and design of an organization, outlining how roles, responsibilities, and relationships are defined. It provides a framework for understanding how various components within the organization interact and collaborate to support its functions. In addition, it offers clear definitions of roles and a matrix mapping of processes to roles.

Role definitions encapsulate a set of functional capabilities essential for an individual to manage one or several distinct business processes, as shown in Table 2.4. It is common for an individual to fulfill multiple roles. The process/role matrix delineates which business roles are accountable for particular business processes. Typically structured at the elementary business process level, this matrix can also be extended to more overarching levels when beneficial.

Role Title	Responsibilities	Reporting
DBA (database administrator)	Managing and monitoring the database activities	IT manager or VP of IT
CIO (chief information officer)	Overall IT leadership	CEO (chief executive officer)
HR employee	Human resources operations	HR manager

Table 2.4 Organizational Roles

In general, the role matrix is a visual representation or chart that outlines the various roles within an organization and their respective responsibilities. It helps in clarifying and communicating the distribution of

tasks and authorities among different positions or departments.

In a financial instruments trading business model, various roles contribute to the overall functioning of the organization, as shown in Table 2.5.

Title	Roles and Responsibilities
Traders	• Manage trading portfolios • Develop strategies to maximize profits and manage risk
Sales team	• Build and maintain client relationships • Understand client needs and offer suitable products and services
Back-office operations[3]	• Process and settle trades • Handle trade confirmation and reconciliation
Risk managers	• Monitor and manage risk exposure • Ensure compliance with risk management policies
Legal advisors	• Provide legal advice and services • Ensure compliance with laws and regulations
Technology professionals	• Develop and maintain trading systems and IT infrastructure • Ensure the security and efficiency of technology resources
Human resources	• Manage recruitment, training, and employee relations • Ensure the organization is staffed with skilled and motivated employees

Table 2.5 Roles and Responsibilities within a Financial Organization

In addition to these titles, there are two primary management layers: upper management and systems management. Upper management is tasked with the supervision of various sectors within the organization, encompassing numerous business units, operations departments, and other key areas. This tier includes roles such as division managers, group managers, and risk managers, who collectively ensure the smooth operation and strategic direction of the business. On the other hand, systems management is dedicated to the routine functioning of the trading system. This includes positions like site manager, systems maintenance manager, content manager, and help desk personnel, all of which collaborate daily to ensure the trading system is operational, well-maintained, and user-friendly.

Process Model

The business process is a series of interrelated tasks, activities, or steps performed in a coordinated manner within an organization to achieve a specific business goal. The business process model can be encapsulated within a framework of business process hierarchies. These hierarchies visually illustrate the outcome of breaking down complex processes, a method known as process decomposition. This decomposition is carried out until it reaches the **elementary business process (EBP)**, which is a fundamental and indivisible activity

3 Often referred to as "back-office personnel." A distinction may be necessary between back-office and front-office functionality. Support—Front-office operations: Typically work alongside traders and salespeople. They facilitate telephone calls, process trades, and resolve trading and sales/customer issues. Front-office personnel would consist of assistant traders and salespeople.

within a business that is not further subdivided into smaller processes (i.e., the smallest unit of business activity). Figure 2.12 illustrates an example of business process modeling for creating an order.

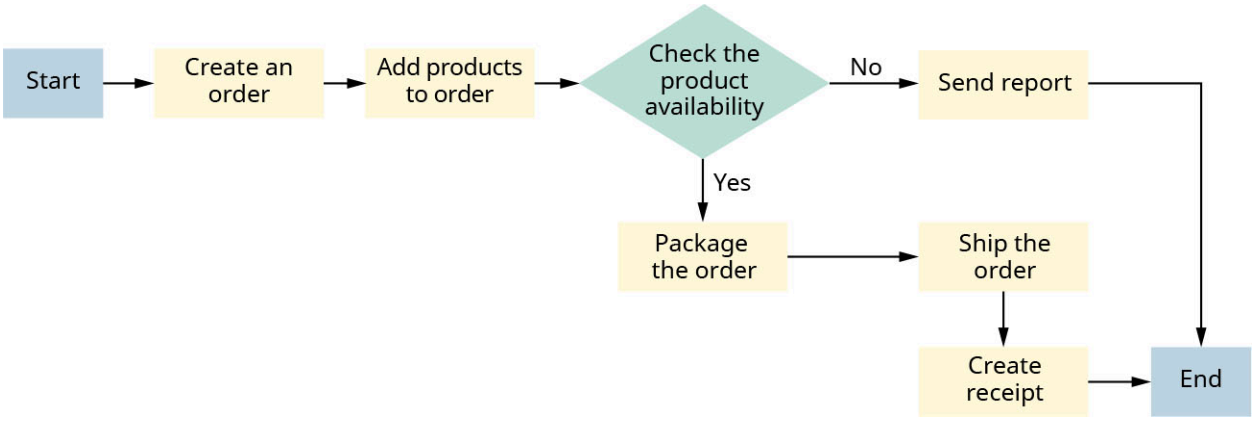


Figure 2.12 This flowchart outlines the business process model for creating an order, checking availability, shipping, and payments. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

EBPs possess distinct characteristics that make them noteworthy in the context of business operations. They are considered significant and candidates for in-depth analysis. EBPs are typically associated with the actions of a single user, emphasizing a focused responsibility within the organization. Furthermore, these processes may encompass both manual and automated activities, reflecting the diverse nature of contemporary business workflows. User involvement in EBPs is concentrated at a specific point in time, streamlining the temporal aspect of these activities. The preservation of crucial business distinctions between processes ensures clarity and coherence. Finally, EBPs aim to leave the conceptual entity model in a consistent state, contributing to data integrity and maintaining a reliable representation of the organization's entities and relationships where the **entity model** represents the various objects or concepts and their relationships within a system. A **process map** displays the order of chosen processes from the process hierarchies, highlighting their connection to the roles responsible for executing them. A **business process hierarchy** organizes a company's activities from broad, general processes down to specific tasks, making it easier to manage and improve how the business operates. Figure 2.13 illustrates a high-level business process hierarchy that could be used for any manufacturing business process.

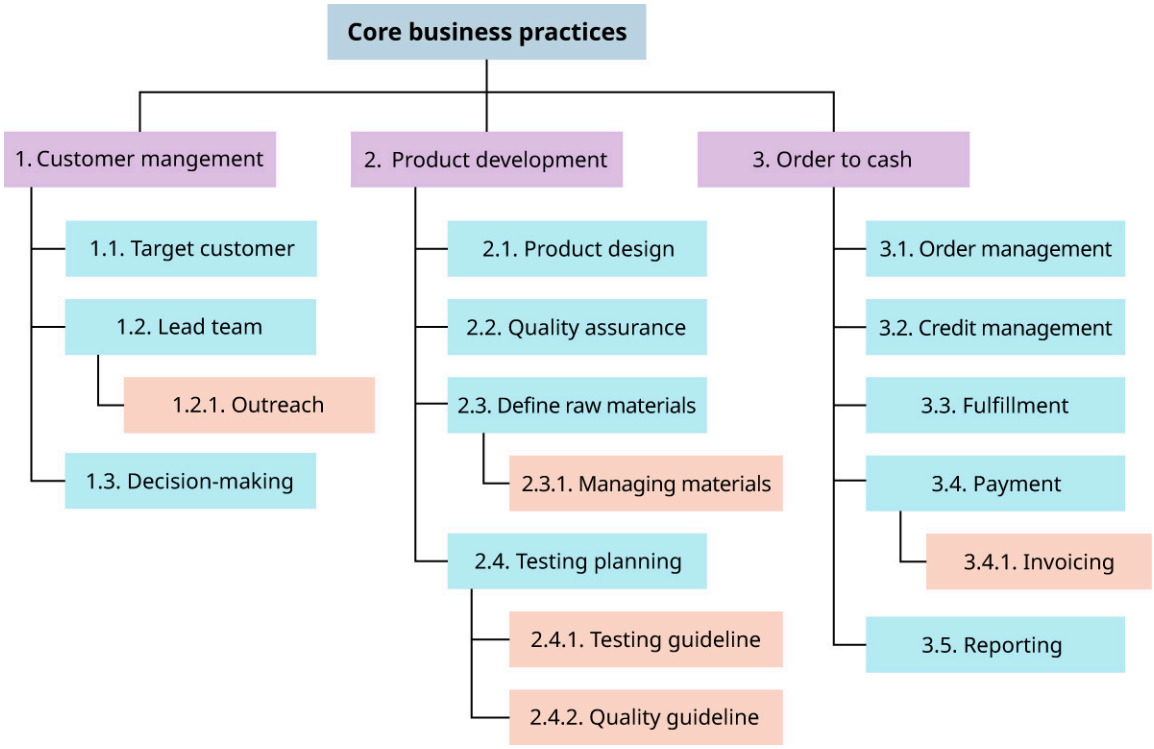


Figure 2.13 The trading business process hierarchy includes activities such as customer management, product development, quality control, order fulfillment, and customer payments. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

At the top of the hierarchy, there are three core business processes:

1. Customer management: This process plays a central role, interacting with various activities designed to acquire, retain, and develop customer relationships. It focuses on all aspects of customer interaction. There are three subsections as follows:
    - 1.1. Target customer: Customers first discover a company and its offerings through various marketing efforts. They then evaluate these offerings to decide if they meet their needs, leading to a purchase decision. After buying, customers may require support, which is provided to ensure satisfaction. A positive experience can turn these customers into loyal buyers and advocates for the company and maintain strong customer relationships.
    - 1.2. Lead team: This involves identifying potential leads and assessing their likelihood to purchase. It includes:
        - 1.2.1. Outreach: This includes reaching out to customers (current and new) with promotions and advertisements.
    - 1.3. Decision-making: This step focuses on the creation and evaluation of product prototypes to ensure they meet the required standards before launch.
2. Product development: This encompasses the entire life cycle of a product from initial idea generation through design and development to launch. There are four subsections:
    - 2.1. Product design: This focuses on the creation and evaluation of product designs to ensure they meet the required standards before testing.
    - 2.2. Quality assurance: QA involves checking the quality of the design and matching it with the listed requirements.
    - 2.3. Define raw materials: Raw materials are the basic substances needed to make products. These can be things taken from nature, like wood, oil, or metals, which are used in making everything from buildings to clothes and food. The type of raw material used affects how products are made, their quality, and how much they cost. It includes:
        - 2.3.1. Managing materials: Managing materials well is important for businesses to make sure they

have enough to meet demand, keep production running smoothly, and reduce waste, making the whole process more efficient and sustainable.

- 2.4. Testing planning: Writing detailed instructions for setting up and conducting usability tests, including how to select participants and record feedback, is important and includes:
  - 2.4.1. Testing guidelines: Rules that help ensure a product or service works well and is safe before it's made available or used. These rules guide how to test the item, what tools to use, and what problems to look for, aiming to fix any issues found. The main goal is to make sure the product does what it's supposed to do and meets quality standards.
  - 2.4.2. Quality guidelines: These are rules that help ensure products or services are consistently good and meet customers' needs. By following these guidelines, companies aim to make their customers happy, reduce mistakes, and work more efficiently. This involves regularly checking and improving how things are done to make sure they meet high standards.

3. Order to cash: This is a comprehensive business flow that starts when a customer places an order and ends when the payment is received and recorded:
   - 3.1. Order management: The process begins when a customer places an order through a chosen method, such as an online platform, phone call, or sales representative.
   - 3.2. Credit management: A credit check is performed to ensure the customer has the necessary credit to make the purchase. If the customer passes the credit check, the order is approved for processing. Otherwise, it may be held until payment is secured or canceled.
   - 3.3. Fulfillment: The required products are allocated from inventory. If products are not available, this may trigger a back-order or manufacturing process.
   - 3.4. Payment: The customer makes a payment using one of the accepted payment methods.
     - 3.4.1. Invoicing: This step will start if the customer pays with a purchase order (i.e., the customer may use different payment methods such as credit card or money transfer). Once the order is shipped, an invoice is generated and sent to the customer, detailing the amount paid and/or due for the products or services provided.
   - 3.5. Reporting: Reports are generated to analyze the sales volume, payment collection efficiency, and customer buying patterns.

EBP definitions include brief explanations of the activities within each process. These descriptions, when paired with the appropriate process flow, serve as a foundation for progressing to intricate design stages and aid in the ongoing creation of functional specifications. Furthermore, they highlight any presumptions established during the architectural development, delineate rules for decisions that require manual intervention, and, where relevant, provide cross-referencing details to ensure compatibility with the functional demands of the application being developed.

The assignment of numbers to EBPs mirrors their placement within hierarchical structures, enabling their linkage to various work products throughout the business, organizational, and location-specific domains, along with pertinent models.

Process map diagrams illustrate the order of chosen processes from the process hierarchies, focusing on their connection to the roles responsible for executing them. The process maps specifically highlight key processes that hold particular business importance. Although every process featured in a process map is also included in the business process hierarchy, not all processes in the hierarchy are depicted in the process maps. Process flow diagrams can depict various elements, such as the business events triggering a process, outcomes of completing a process, the processes themselves, the sequence of process steps, interruptions in the process flow, possibilities for repetition, choices, exclusivity among processes, and any additional notes. Figure 2.14 illustrates a process map diagram for an order to cash business process.

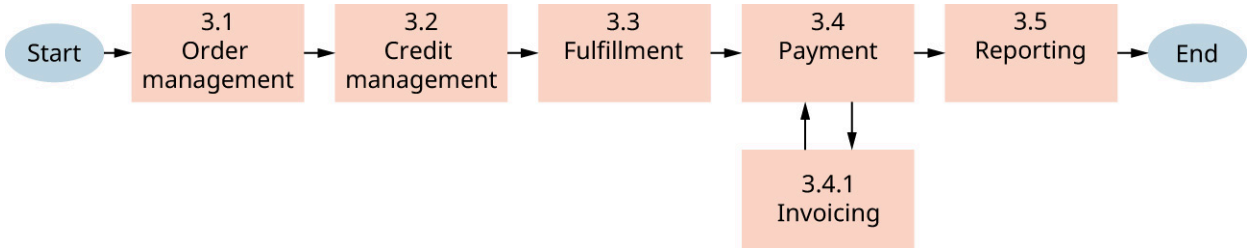


Figure 2.14 The order to cash business process flows from order management through product fulfillment and payment. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Figure 2.15 illustrates the process of managing orders within a business. It begins with order management, where orders are received and verified for accuracy. Next is credit management, which assesses the customer's credit to ensure they can fulfill payment requirements. The process then moves to fulfillment, where the order is prepared and shipped. Then, during payment, the customer is invoiced and payment is processed. Invoicing is a detailed part of this step, where an invoice is generated and sent to the customer. The final stage is reporting, where the business generates reports on sales and financial transactions. This process ends once all steps are completed, ensuring each order is handled efficiently from start to finish.

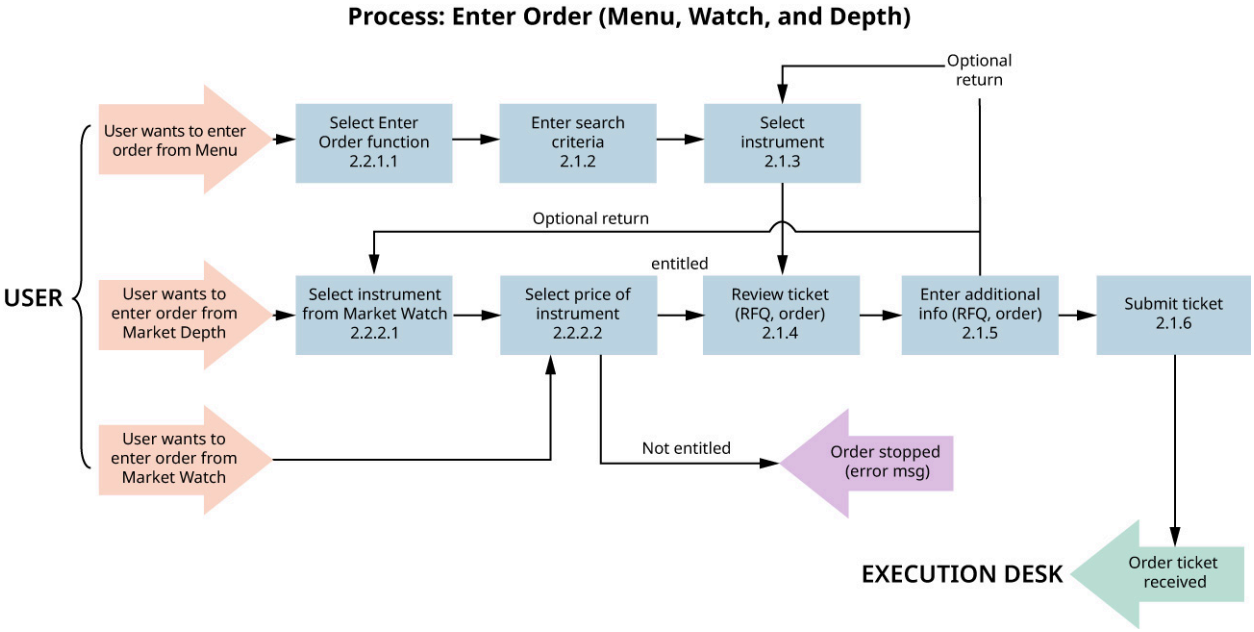


Figure 2.15 This sample trading business model represents a process map diagram for the enter order process. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Location Model

A location model refers to a set of rules used to analyze and make decisions related to the positioning of entities, activities, or resources. The conceptual location model shows how business processes will be distributed geographically. Within the location model, definitions of location types specify the identification of locations based on both their type and general geographic area. Additionally, the location model contains a matrix illustrating the relationship between processes and locations, indicating the specific location types where each process takes place.

Enterprise Technical Architecture

The enterprise technology architecture (ETA) is a comprehensive framework that defines the structure, components, and interrelationships of an organization's technology systems to support its business processes and objectives. In general, the ETA guides the development and support of an organization's information systems[4] and technology infrastructure. Therefore, it characterizes the organization's application, data, and technology infrastructure architectures and describes their models. The technology (virtual infrastructure) supports the execution of information systems services, which in turn support business functions and related services. Application, data, and technology infrastructure architecture models may be described via blueprints at different levels of abstraction, including presentation, conceptual, logical, and physical. Layering and componentization heuristics play an important role in structuring these models.

The abstraction level indicates the extent to which a design is distanced from tangible technological details. This level of abstraction can differ across various architectural fields, but four main levels are widely recognized:

1. Presentation: At this level, architecture is simplified into a form to streamline the communication of essential ideas, particularly to business executives. Distilling complex architectural diagrams into more straightforward, high-level visuals ensures that the core messages are conveyed effectively.
2. Conceptual: This level offers a more structured view of architecture, deliberately omitting many specifics. The rationale for not including certain details is that they may not be relevant to the diagram's intended purpose or are yet to be decided.
3. Logical: The architecture is depicted in detail but remains uncertain of specific technologies. It aims to give a full outline without committing to the particular technologies that will be employed for the final build.
4. Physical: This level focuses on the actual technological specifics used or to be used in the architecture's realization, making it the most concrete representation of the four.

CONCEPTS IN PRACTICE

Manufacturing Industry

Business operations are commonly articulated through a *value chain* framework, a concept originating from manufacturing practices. Analogous to the manufacturing industry where raw materials like steel are transformed into various components, each step in the value chain plays a role in creating a finished product, such as a car. In our approach, we've utilized the componentization heuristic to break down business operations into distinct steps constituting the elements of a value chain. Additionally, the layering heuristic has been applied to structure the value chain as layers of elements. Each layer in this model relies on the output of the preceding layer to perform its specific function. This methodology enhances the understanding of how value is sequentially added throughout the business process.

Application Architecture Model

The **application architecture** is a subset of the enterprise solution architecture that includes a process to architect and design the application architecture as well as the actual application architecture model, which represents the content of the application architecture. Application architecture helps organizations decide how to invest in new software and systems. It makes sure that any new software being looked at, designed, or put into use can work well with the systems the organization already has. This includes brand-new software, updates to old software, making old software better, and buying and updating software packages.

4 Information systems architectures combine the use of application and data architecture details and specify control and data flow necessary to operate the systems.

Figure 2.16 illustrates a conceptual application architecture model of the sample trading business model that was introduced earlier in this section. The goal of a conceptual application architecture is to demonstrate how members of the organization interact with the application architecture from different locations when invoking business processes. This example illustrates three distinct offices accommodating a diverse range of users, including management, trading, IT, and others.

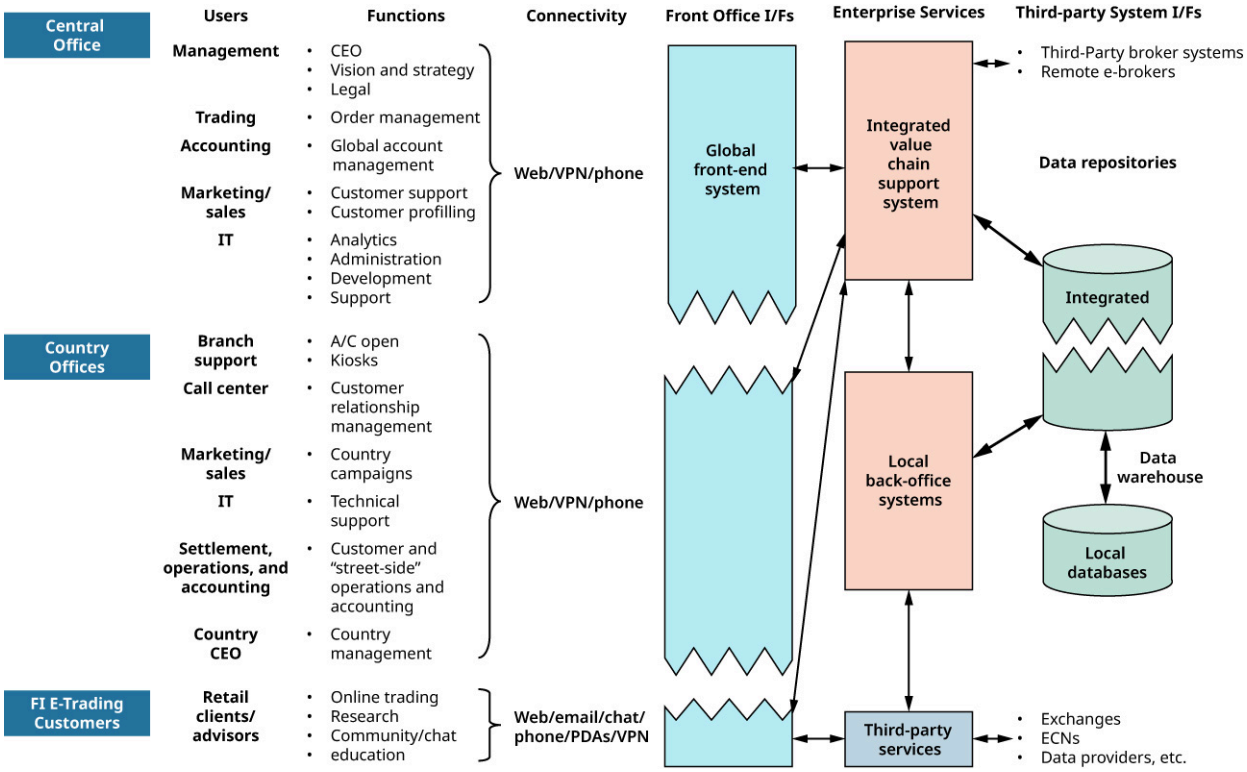


Figure 2.16 A conceptual trading application architecture moves through various levels, including users, functions, enterprise services, and third-party systems. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Figure 2.17 takes the conceptual trading application architecture a bit further and describes what one of the alternative application architectures may look like at the logical level. The goal of the logical application architecture is to identify the various functional blocks within the architecture without getting into the details of how they connect with and/or use each other.

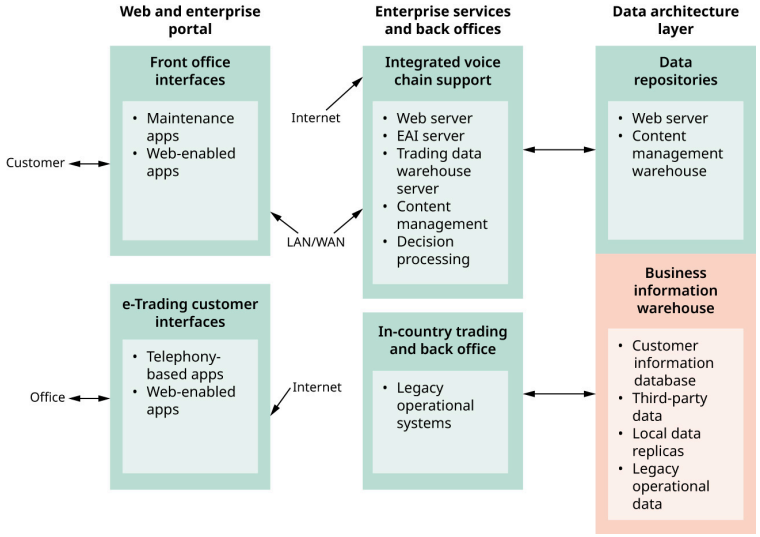


Figure 2.17 The logical trading application architecture illustrates each function block for the processing of customer orders. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

THINK IT THROUGH

Enterprise Business Architecture

What is the mechanism that prompts the creation of organization, location, and process domains to describe an (enterprise) business architecture? How does decomposition help describe (enterprise) technology architectures?

Hint: The organization domain focuses on the structure, roles, responsibilities, and relationships within the enterprise. The location domain deals with the physical or virtual places where business activities take place. The process domain delves into the business processes that drive the value chain and operational activities.

The process involves dividing a system or architecture into smaller, more discrete elements, which aids in analysis, understanding, and effective communication. Technology architectures in enterprises can be highly complex, involving numerous components and interdependencies. Decomposition simplifies this complexity by breaking down the architecture into smaller, comprehensible pieces. Each component can then be analyzed and understood independently.

Figure 2.18 is a simplified view of the logical trading application architecture model that provides callouts to explain what the various functions are meant to accomplish. It is good practice to document the functions in that way. It is also good practice to provide an additional diagram (not included here) that includes callouts identifying the actual third-party technologies that are used to implement the various functions.

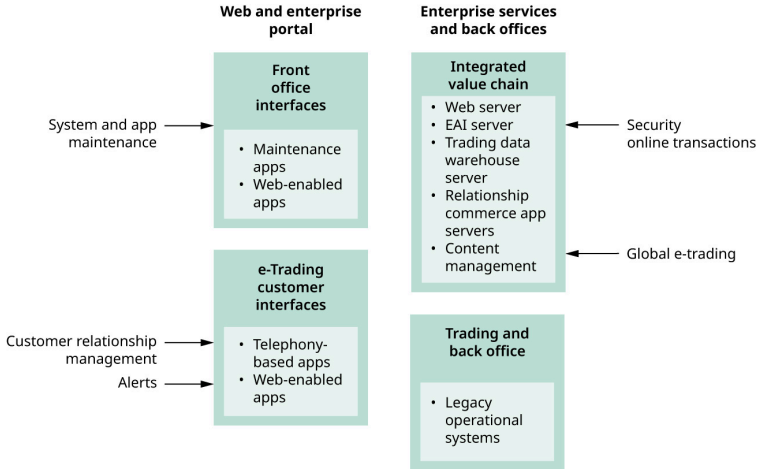


Figure 2.18 This logical trading application architecture includes function callouts that identify the technologies used for this process. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Data Architecture Model

A **data architecture model** is a conceptual framework that outlines how an organization structures, organizes, and manages its data assets. Data architecture forms a component of the broader enterprise solution architecture. It encompasses the design process for both information and actual data architecture models, representing the content within the data architecture. This framework aids organizations in strategically planning investments in data management solutions and associated systems. The evaluated, designed, and delivered data management solutions must seamlessly coexist with established ones, managing newly developed databases as well as legacy database extensions.

In general, information can be extracted from raw data, knowledge can be gleaned from information, and wisdom can be obtained from knowledge. Most enterprises refer to the relationship between data, information, knowledge, and wisdom as the *pyramid of knowledge*. A wealth of additional information related

to data management and the pyramid of knowledge is provided in Chapter 8 Data Management. Information modeling is the process used to describe the metadata necessary to understand the data, processes, and rules that are relevant to the enterprise, as illustrated in Figure 2.19.

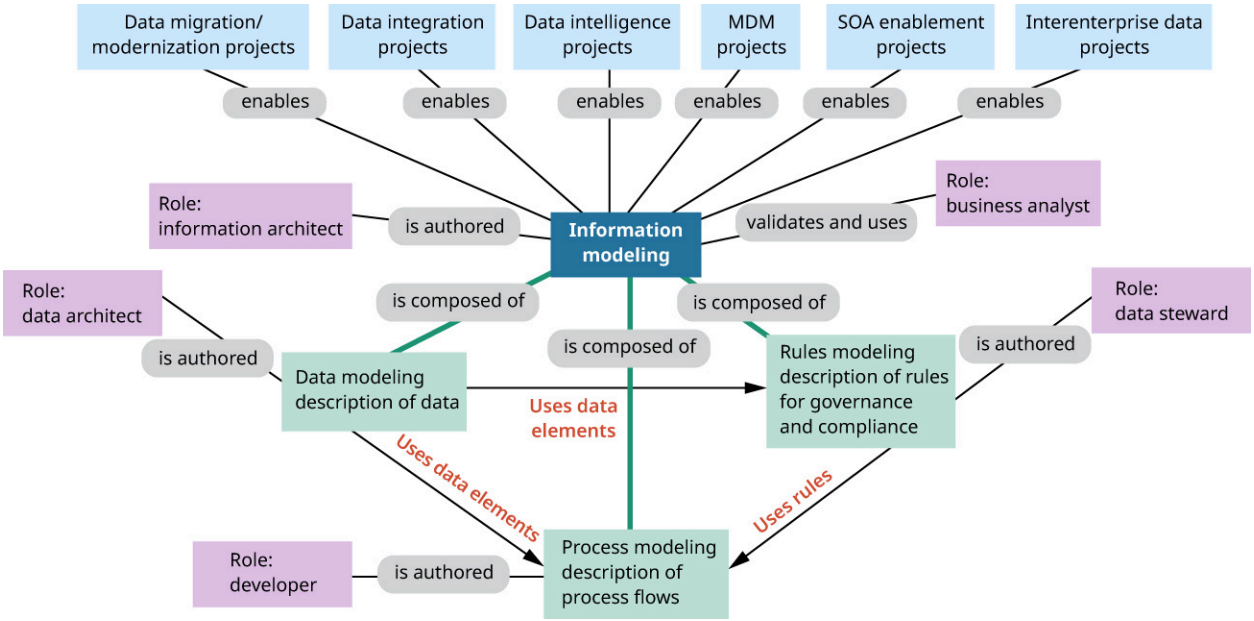


Figure 2.19 Examining and learning from data is integral to an organization's success, as outlined in this role of information modeling figure. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

In the collaborative process of **data modeling**, IT and business stakeholders establish a shared understanding of essential business terms, known as entities, which typically end up being represented as tables that contain data in a relational database management system. This involves defining the attributes that characterize these terms and establishing the relationships between them. The capability to uphold and document the data model is integral to an organization's capacity to address varied data acquisition needs across critical business projects. In essence, a well-maintained data model serves as a foundational element for ensuring coherence and effectiveness in managing diverse data requirements.

Figure 2.20 is an example of an enterprise-level conceptual data architecture for an insurance company. The goal of the enterprise conceptual data architecture is to illustrate the various types of data repositories and the way data is collected and managed within the enterprise.

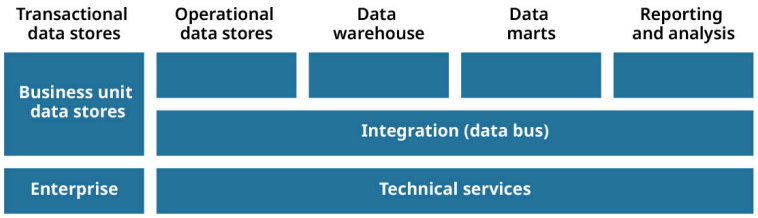


Figure 2.20 The enterprise conceptual data architecture for a fictitious insurance company highlights the different ways in which data is handled and managed. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Concerning the data that flows through the application architecture, there may be many types of data (e.g., relational and NoSQL) and various ways to represent the corresponding data architecture models. A relational database organizes data into tables where each row represents a unique record and each column represents a property of that record. It uses a language called SQL to manage and query the data. NoSQL databases are designed to store and manage data that doesn't fit neatly into tables and can handle various types of data models like documents or graphs.

INDUSTRY SPOTLIGHT

Design Reuse

Adaptive design reuse plays a crucial role across various industries today, including the health-care sector, where its impact is profoundly significant. Understanding and leveraging adaptive design reuse in health care can lead to innovative solutions for complex medical problems, enhancing patient care and treatment outcomes. One prime example of its application is in the design of artificial valves that can replace natural heart valves in people.

Heart valve disease is a condition where one or more valves in the heart do not function properly, leading to disrupted blood flow. The traditional solution involves surgical replacement with prosthetic valves, which can be derived from biological sources or made from synthetic materials. Adaptive design reuse in this context refers to the innovative process of designing these prosthetic heart valves by repurposing existing materials, technologies, and design principles from within or outside the medical field. This approach can accelerate the development of more effective, durable, and safer heart valve replacements.

Infrastructure Architecture (Infrastructure Pillars)

Technology architecture is a fundamental component of enterprise architecture, supported by four main pillars: compute, memory, storage, and network. It outlines the organization and functionality of an enterprise's solutions or system's technology framework. This encompasses the configuration of client and server hardware, the applications operating on this hardware, the services these applications provide, and the protocols and networks facilitating communication between applications and hardware components. It's important to distinguish technology architecture from system architecture. The **system architecture** deals with applications and data, how they are related to each other, and what business process they support together. The **technical architecture** includes the software and hardware capabilities to fully enable application and data services.

Refer to Figure 10.36 for an example of an enterprise-level conceptual technology architecture for a fictitious company. The goal of the enterprise conceptual technology architecture is to illustrate the various types of hardware components that are part of the enterprise infrastructure and the way they are laid out at a high level.

GLOBAL ISSUES IN TECHNOLOGY

Design Reuse Broader Impacts

Focusing on reusing existing designs and technologies can save time and money, but it might also limit new ideas and innovations because designers could stick too closely to what's already been done. This approach can have several effects:

- Socially, it might not meet the needs of all users, especially if the technology doesn't consider different cultures or lifestyles, leading to some people being left out.
- Ethically, there's a question about fairness and whether technology serves everyone equally, as relying on old designs may not address current or future challenges well.
- Environmentally, while using existing designs could reduce waste and save resources, it might also keep using outdated, less eco-friendly technologies instead of developing cleaner, more efficient options.
- Economically, countries that already have a lot of designs and technologies could get ahead because they have more to reuse, making it harder for countries with fewer resources to catch up or compete.

While reusing designs has its benefits, it's important to also think about these broader impacts and strive for a balance between recycling old ideas and creating new ones to make sure technology keeps improving in a way that's good for everyone.

Figure 2.21 illustrates a possible physical architecture for the sample trading business model that was introduced earlier in this section. This diagram depicts the layout of the actual hardware components that make up the infrastructure of the trading solution. It also delineates where the functional blocks of the application architecture are physically deployed. Note that this physical technology architecture leverages the layout and components of the enterprise application architecture illustrated previously at the conceptual level.

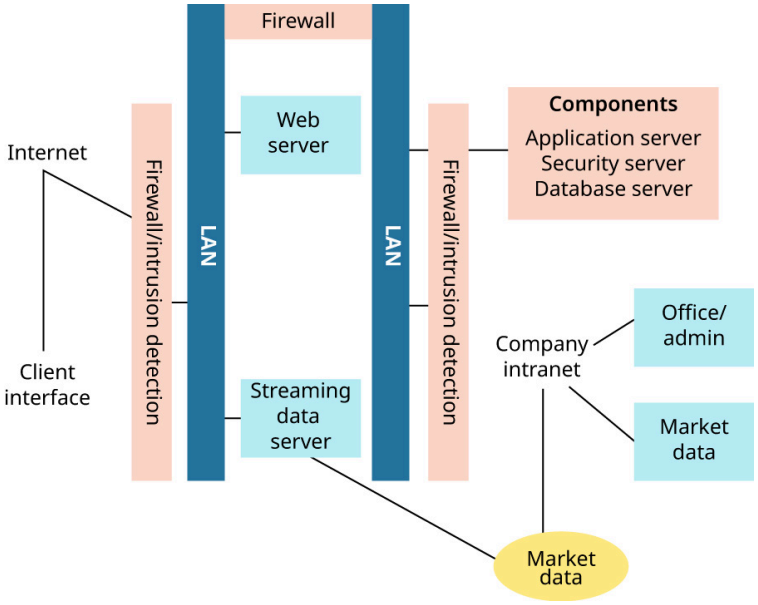


Figure 2.21 The physical trading application architecture highlights the use of hardware to meet the organization's goals. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

An alternative physical application architecture for the trading solution is shown in Figure 2.22. The diagram does not delineate where the functional blocks of the application architecture are physically deployed.

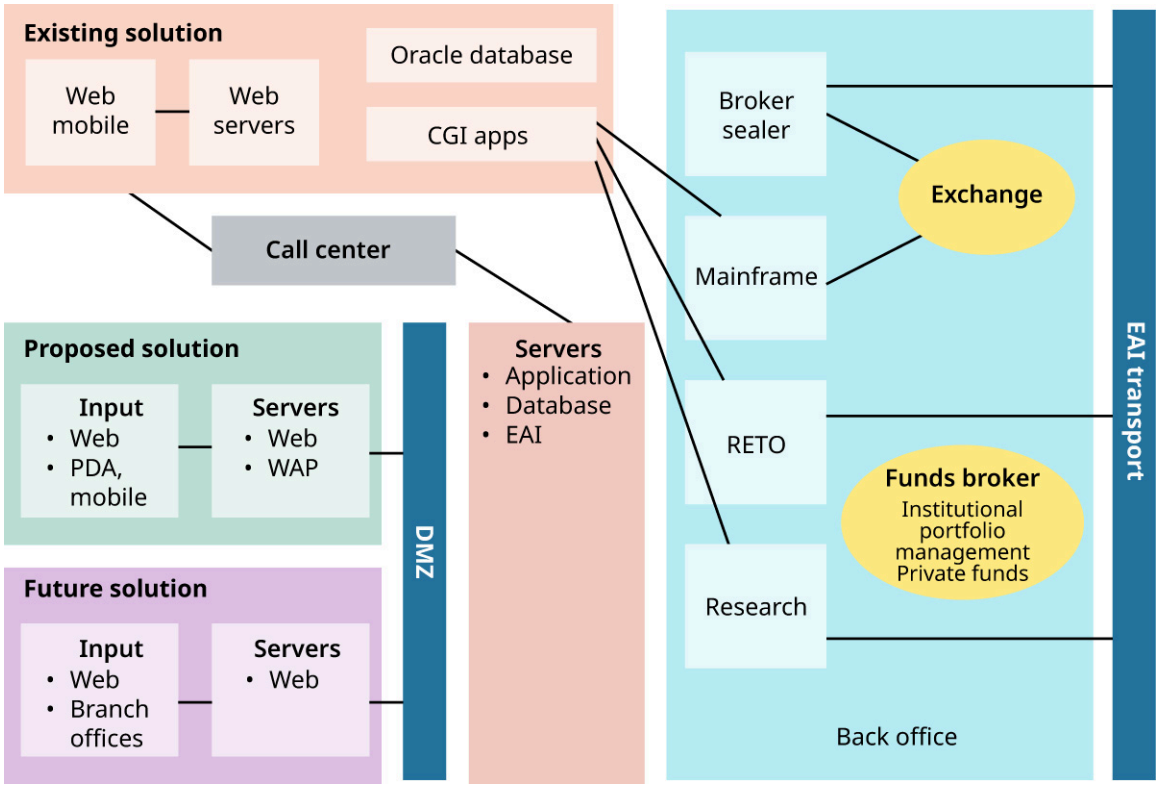


Figure 2.22 This alternative physical trading application architecture outlines possible changes from the existing web solution. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Information Systems Architecture View

Architecture views are representations of the overall system design that matter to different stakeholders. In addition to the application, data, and technology architecture models, IT architects create specific views to communicate and ensure that the system meets the needs of various stakeholders. An architecture is typically conveyed through one or more architecture models that collectively offer a clear description of the system's structure. A singular, all-encompassing model is often too intricate to be easily grasped, displaying every intricate relationship among diverse business and technical elements.

Just as an architect designs different aspects of a building, such as wiring diagrams for electricians, floor plans for owners, and elevations for planners to address their unique needs, the architecture of an information system is similarly broken down into various views for effective communication among its stakeholders. In the IT world, for example, an architect might create specific views of the system's physical layout and security measures. These tailored views ensure that stakeholders, each with their own concerns and areas of focus, have a clear understanding of the system's components relevant to their interests.

From Enterprise to Solution Architecture

After identifying business and technical characteristics through the diagrams discussed in the previous section, solution models can be developed, and implementations can be created. This involves constructing new components as necessary and combining them with reusable design components obtained from a pattern catalog. If implementations of these reusable components already exist and can be customized, the implementation of the solution becomes much faster. This approach helps avoid reinventing the wheel and developing software components or systems that already exist, focusing instead on assembling existing components for efficiency.

Breadth of Applicability of Models

Figure 2.23 illustrates the key dimensions for representing and categorizing architecture models,

accompanying diagrams, and related patterns. These architecture domains align with the TOGAF standard, as discussed earlier. The diagram also illustrates the levels of abstraction to characterize various architectural models. Additionally, it introduces the architecture scope as another dimension, classifying the models' breadth of applicability at the enterprise, portfolio, or project level. The **architecture scope** is the extent and boundaries within which architectural considerations, decisions, and solutions apply.

<table>
<tr><td colspan="4">To address the concerns of the following stakeholders...</td></tr>
<tr><td>Users, planners, business management</td><td>Database designers and administrators, system engineers</td><td>System and software engineers</td><td>Acquirers, operators, administrators, and managers</td></tr>
<tr><td colspan="4">... the following views may be developed</td></tr>
<tr><td>Business architecture views</td><td>Data architecture views</td><td>Applications architecture views</td><td>Technology architecture views</td></tr>
<tr><td>Business function view</td><td rowspan="5">Data entity view</td><td rowspan="5">Software engineering view</td><td rowspan="4">Networked computing/ hardware view</td></tr>
<tr><td>Business services view</td></tr>
<tr><td>Business process view</td></tr>
<tr><td>Business information view</td></tr>
<tr><td>Business locations view</td><td rowspan="3">Communications engineering view</td></tr>
<tr><td>Business logistics view</td><td rowspan="4">Data flow view (organization data use)</td><td rowspan="4">Applications interoperability view</td></tr>
<tr><td>People view (organization chart)</td><td rowspan="3">Processing view</td></tr>
<tr><td>Workflow view</td></tr>
<tr><td>Usability view</td></tr>
<tr><td>Business strategy and goals view</td><td rowspan="5">Logical data view</td><td rowspan="5">Software distribution view</td><td>Cost view</td></tr>
<tr><td>Business objectives view</td><td rowspan="3">Standards view</td></tr>
<tr><td>Business rules view</td></tr>
<tr><td>Business events view</td></tr>
<tr><td>Business performance view</td></tr>
<tr><td></td><td colspan="3">System engineering view</td></tr>
<tr><td colspan="4">Enterprise security view</td></tr>
<tr><td colspan="4">Enterprise manageability view</td></tr>
<tr><td colspan="4">Enterprise quality of service view</td></tr>
<tr><td colspan="4">Enterprise mobility view</td></tr>
</table>

Figure 2.23 TOGAF architectural dimensions include various levels of abstraction. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Portfolio- or domain-level architectures usually concentrate on collections of solutions and projects associated with a specific business unit, like marketing or sales in a large organization. In contrast, project- or system-level architecture is geared toward individual solutions and projects within those business units. Defining the

scope of a model is crucial because there exists a direct relationship between the scope and the level of detail achievable in a blueprint. This is due to the necessity for increased generalization, such as simplification, feature selection, and grouping, as the blueprint's scope expands.

TECHNOLOGY IN EVERYDAY LIFE

Adaptive Design Reuse: Eco-Friendly Homes from Recycled Materials

Adaptive design reuse can significantly benefit people in everyday life by promoting efficiency, sustainability, and improved user experiences. Imagine a neighborhood called EcoHomes, where all the houses are built using old materials from buildings that were taken down or left unused. It is all about making new homes without needing to produce or buy more materials, which helps the environment. In EcoHomes, architects and builders take things like bricks, glass, and wood from old sites and use them to build new, modern houses. For example, wooden beams from an old barn become part of the living room in a new house, adding a cool, old-time feel to a modern design. Windows from an old office building let in lots of sunlight, cutting down on the need for electric lights. EcoHomes is a hit because it shows how reusing building materials can save money and help the planet. The people living there have lower energy bills and are proud of their unique, eco-friendly homes. This story shows how using what we already have in new ways can make a big difference for our wallets and the world.

2.3 Evolving Architectures into Useable Products

Learning Objectives

By the end of this section, you will be able to:

- Analyze similarities between architectures and apply patterns
- Discuss how to accelerate the creation of applications

The combination of top-down, adaptive design reuse, and bottom-up, computational thinking, optimizes modern software development. This blend allows software developers to find a middle ground by adapting and assembling existing components, minimizing the need for developing entirely new software. A clear example of this cooperation is evident in modern websites, where the Model-View-Controller architectural pattern is widely employed. The **Model-View-Controller (MVC)** is a software architectural pattern commonly used in the design of interactive applications, providing a systematic way to organize and structure code. The pattern separates an application into three interconnected components: model, view, and controller. The *model* represents the application's data structure and business logic, managing data and rules. The *view* is responsible for displaying the user interface; it shows data to the user and sends user commands to the controller. The *controller* serves as an intermediary between the model and the view. It processes user input received from the view, interacts with the model to retrieve or update data, and determines the appropriate view for presenting the response. Many practical web application frameworks, such as Django, have already implemented the MVC pattern. In this setup, the application is divided into three parts: the model handles the data structure, the view displays the data on web pages, and the controller manages the business logic, facilitating interaction between the model and the view. Adding a broker pattern to MVC architectures can improve the system's scalability and flexibility when applicable and/or necessary. The broker acts as a middleman that manages communication between different parts of the application, helping to handle more data and complex operations efficiently.

Leveraging these existing frameworks enables developers to concentrate on crafting the specific logic relevant to the website rather than reinventing the wheel. The beauty of this approach lies in the ability to swiftly piece together solutions by extending and adapting the available frameworks. By doing so, developers streamline the development process, enhance efficiency, and capitalize on the collective wisdom embedded in proven

frameworks, thereby fostering innovation in a more focused and resource-efficient manner.

Leveraging Architectural Similarities and Applying Patterns

The adaptive design reuse approach is a strategy in software development that emphasizes the efficient reuse of existing design solutions to create new systems or applications. The beauty of the adaptive design reuse approach is that the business solution architecture model helps create abstract representations of real systems. Therefore, if there exist tangible realizations of the various components that are part of these abstract representations, it is possible to implement the model and create specialized running business solutions from it.

A **solutions continuum** is a strategy where existing solutions, components, or patterns are leveraged and adapted for use in different contexts. Figure 2.24 illustrates the TOGAF model of reuse that is referred to as the solutions continuum. As mentioned earlier, TOGAF does not provide a prescriptive approach to creating and/or managing a catalog of patterns. However, various pattern repositories are available on the Internet and the adaptive design technique can be used to avoid having to reinvent the wheel when architectural patterns and related component implementations exist and can be customized and assembled with new components. More information on this topic is provided in Chapter 10 Enterprise and Solution Architectures Management.

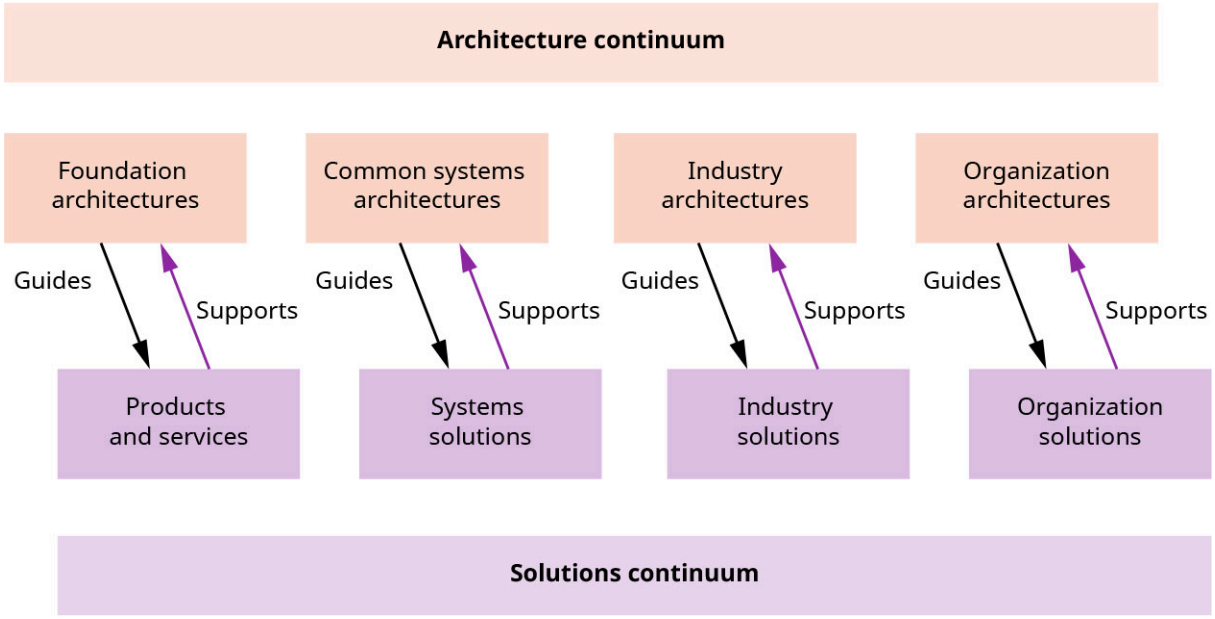


Figure 2.24 This TOGAF solutions continuum illustrates how each architecture guides and supports the others. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

As illustrated in Figure 2.25, the TOGAF solutions continuum offers a limited set of dimensions. It serves as a guideline, and The Open Group allows interested parties to enhance the model by incorporating additional dimensions that are relevant to their specific needs.

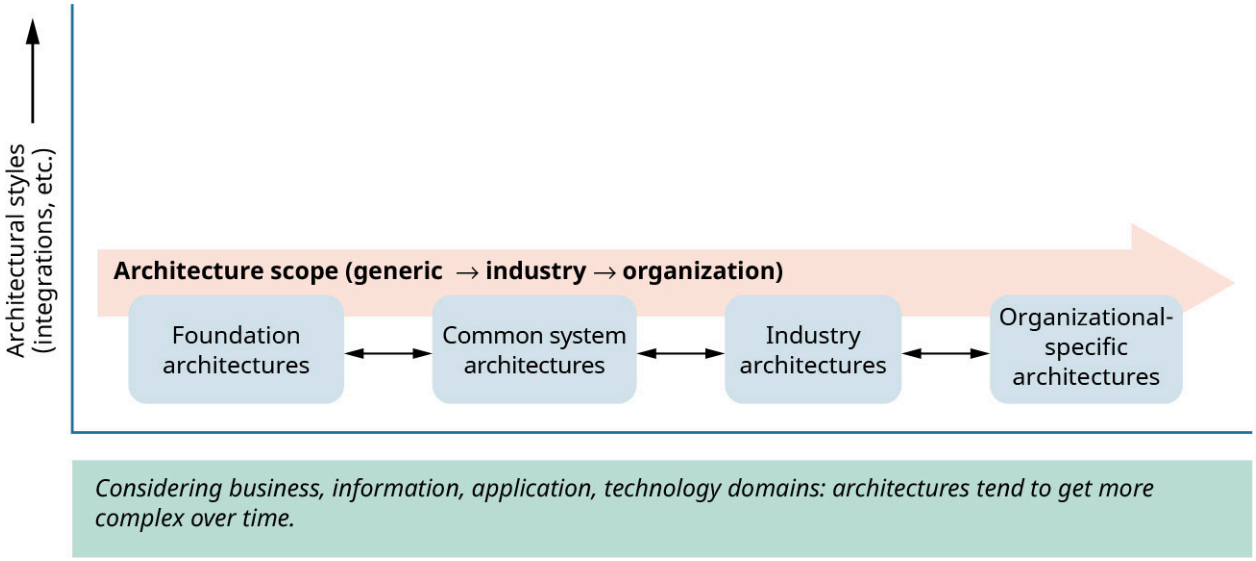


Figure 2.25 The TOGAF architecture continuum can suggest extensions but may only be able to focus on one aspect at a time. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Accelerating the Creation of Mainstream Business Solutions

To illustrate the power of architectural design and adaptive design reuse, various designs used in mainstream business solutions are surveyed, followed by explanations as to how corresponding turnkey solutions can be derived from these models. Several subsequent chapters of the book elaborate on building-related solutions.

CONCEPTS IN PRACTICE

Object Management Architecture (OMA)

Organizations like the Object Management Group (OMG) create foundational and common system architectures that may be used across industries. An example is the Object Management Architecture (OMA), which is a foundation for developing architectures as building blocks. It then elaborates in providing Object Services, Horizontal Facilities, and Vertical Facilities as subcomponents to help classify common system architectures that may be used to assemble a complete OMA-centric architecture. It is then the responsibility of the various industries to establish standard architectures that may be leveraged by organizations that operate in these industries. Finally, organizations benefit from being able to leverage foundational, common systems and industry architectures to develop their own proprietary architectures. Based on the models of the various architectures that organizations may use and assuming there exists solutions for them, organizations can develop their own solutions faster by reusing and customizing existing solution components instead of reinventing the wheel. This is actually how the TOGAF solution continuum applies adaptive design reuse.

Responsive Web 2.0 Business Solutions

World Wide Web Consortium (W3C) is an international community that develops guidelines to ensure the long-term growth and accessibility of the World Wide Web. **Web 2.0** is the second generation of the World Wide Web when we shift from static web pages to dynamic content. **Web 3.0** is the third generation of the World Wide Web and represents a vision for the future of the Internet characterized by advanced technologies. Most modern websites rely on the Web 2.0 architectural model set forth by W3C. A sample logical application architecture model is illustrated in Figure 2.26.

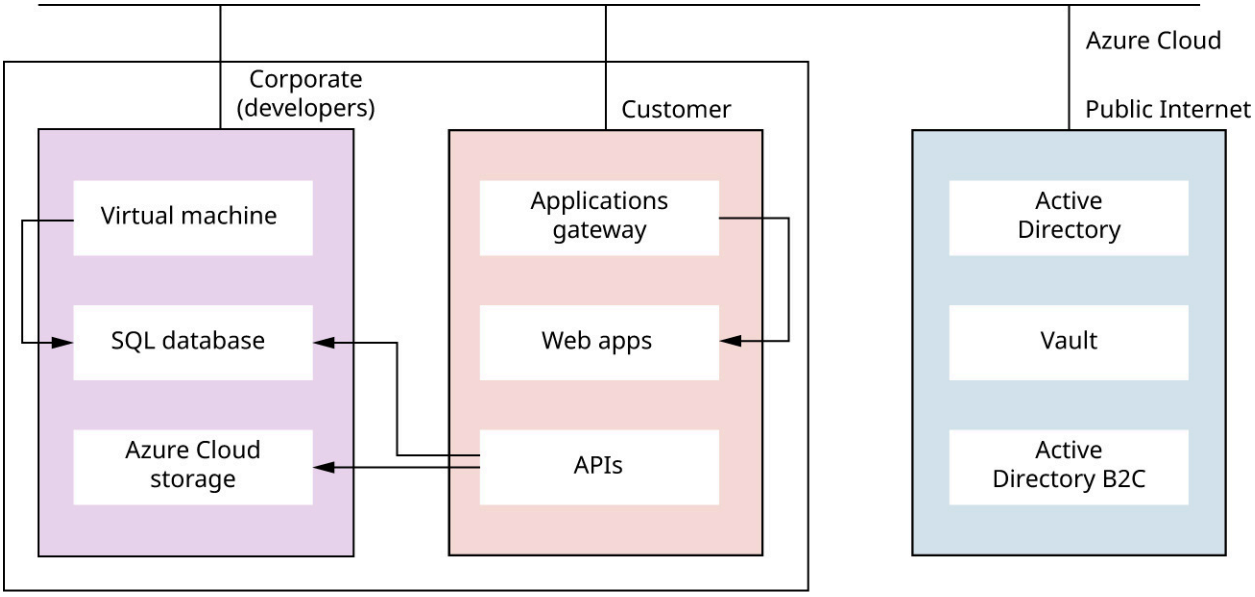


Figure 2.26 The logical application architecture of Microsoft Azure-hosted web applications allows for responsive web and mobile solutions for users. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

In this case, the model leverages the various components available on the Microsoft Azure Cloud. **Microsoft Azure** is a comprehensive cloud computing platform provided by Microsoft. Azure is designed to help organizations build, deploy, and manage applications and services through a global network of data centers. Azure provides streamlined development capabilities under its DevOps offering to make it very easy to develop and quickly deploy websites on the Azure platform using mainstream web application frameworks (e.g., ASP.Net, PhP, Java). DevOps is an Agile Software Engineering tools-driven approach that focuses on developing software and deploying it into operation.

Many of the support components required to support website implementations are readily available on Microsoft Azure and other systems that provide reusable components for responsible web design. It is easy to evolve the model shown below into a running website. A **web application framework** has built-in support for architectural patterns that make it easy to extend the framework and use plug-ins to implement commercial-grade websites in a reasonable amount of time. They also support the use of web frameworks that make it possible to build a responsive web application that makes the functionality available on the desktop version of the application seamlessly available on a mobile device. In addition to these capabilities, the adaptive design reuse approach may be used to create the custom part of the web application. More information related to the development of web solutions is provided in Chapter 9 Software Engineering, Chapter 10 Enterprise and Solution Architectures Management, and Chapter 11 Web Applications Development.

THINK IT THROUGH

Architectural Similarities

What is one of the mechanisms that makes it possible to compare architectural similarities between two solutions at different levels?

Native Mobile Business Solutions

A **web application (web app)** is a software application that is accessed and interacted with through a web browser over the Internet. Many web-based solutions leverage the inherent capabilities of mobile devices, offering web apps tailored for various types of phones in addition to responsive websites. Numerous frameworks exist to facilitate the development of native web apps, streamlining the process of creating

applications that can run seamlessly on different mobile platforms. These frameworks often provide a unified and efficient approach to building cross-platform mobile applications, ensuring a consistent user experience across various devices.

In certain frameworks and development environments, React Native UI component libraries can be leveraged to, port web apps to mobile devices. Examples include React Native support for Android apps using the Android Studio (Android Studio provides a comprehensive environment for developing, testing, and debugging Android apps) or iPhone web app using XCode IDEs (Xcode is an integrated development environment [IDE] developed by Apple for macOS that offers a suite of tools for building software for Apple platforms, including macOS, iOS, watchOS, and tvOS). Figure 2.27 illustrates the logical application architecture of mobile web apps that use React Native. In addition to these capabilities, the adaptive design reuse approach may be used to create the custom part of the native web app. More information related to the development of native web app solutions is provided in Chapter 9 Software Engineering, Chapter 10 Enterprise and Solution Architectures Management, and Chapter 11 Web Applications Development.

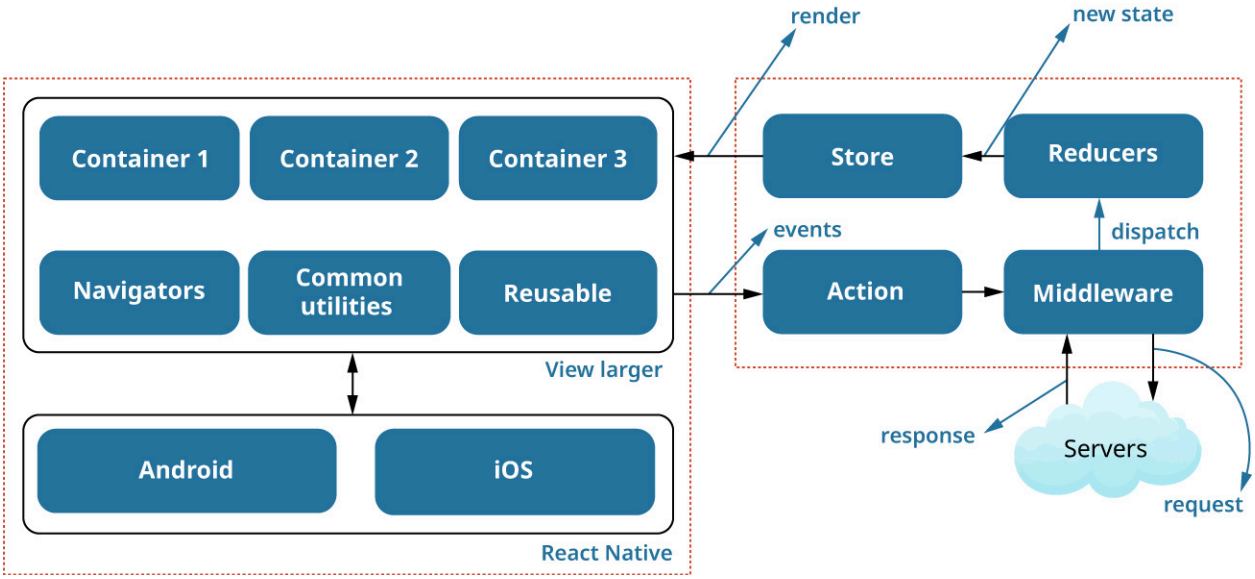


Figure 2.27 The logical application architecture of React Native mobile web apps shows the back-end processes that allow both Android and IOS customers to use the same application. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Native Mobile Business Examples

Native mobile apps are designed specifically for mobile operating systems, providing optimal performance and a seamless user experience.

- WhatsApp: WhatsApp is a native mobile app designed specifically for iOS and Android platforms. It directly accesses the hardware of the device, such as the GPS, camera, and microphone, which allows for features like real-time location sharing, voice and video calls, and media sharing.
- Instagram: Instagram is a photo- and video-sharing app. Native development helps Instagram manage high-quality media content efficiently, apply real-time filters, and smoothly handle in-app animations.
- Uber Eats: Uber Eats is a food-delivery service that operates as a native app on mobile devices. Being native allows the app to use device-specific features, such as GPS for tracking the delivery person's location in real time.
- Spotify: Spotify uses its native app to deliver personalized music and podcast streaming services. The app's native nature allows it to integrate closely with the device's hardware, offering features like offline downloading, low-latency streaming, and background play.

Web 3.0 Business Solutions

The secure and transparent way of recording transactions that uses a chain of blocks, each storing a list of encrypted transactions is called **blockchain**. Once a block is full, it is linked to the previous one, forming a chain. Blockchain technology decentralizes processing to ensure the integrity of transactions across multiple computer nodes. This ensures that no single computer node gets assigned to processing transactions repeatedly, thereby preventing possible fraudulent modifications of transactions. A **smart contract** is an automated agreement written in code that runs on blockchain technology. They enforce contract terms automatically when specific conditions are met, removing the need for intermediaries and ensuring security. The use of blockchain smart contracts within web applications is becoming more popular. The logical application architecture model in Figure 2.28 illustrates how this is made possible by creating hybrid Web 2.0 websites that interact with Web 3.0 smart contracts.

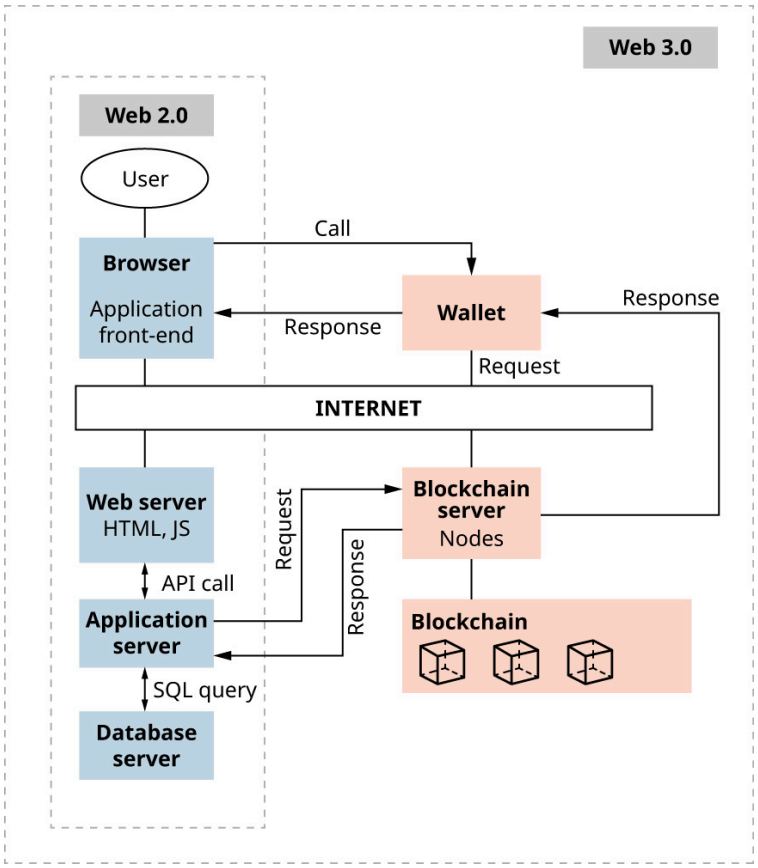


Figure 2.28 The flowchart show the logical application architecture of a Web 2.0 and Web 3.0 Website. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Building these types of business solutions is greatly facilitated by the use of the **Ethereum platform**, an open-source blockchain platform that enables the creation and execution of smart contracts and decentralized applications, or Cloud blockchain platforms provided by one of the Cloud service providers such as Amazon AWS, Google GCP, IBM Cloud, Consensys, Oracle Cloud, and others. These platforms provide frameworks and APIs that make it easy to develop and deploy smart contracts. The Web 2.0 portion of the website can leverage the frameworks mentioned earlier. In addition to these capabilities, the adaptive design reuse approach may be used to create the custom part of the Web 3.0 application. More information related to the development of Web 3.0 solutions is provided in Chapter 9 Software Engineering, Chapter 10 Enterprise and Solution Architectures Management, and Chapter 13 Hybrid Multicloud Digital Solutions Development.

Cloud-Native Business Solutions

A way of building software by breaking it into small, independent pieces where each piece, or service, does a

specific job and works on its own is called **microservices**. A large number of businesses have been migrating their legacy business solutions to the cloud to take advantage of microservices that are designed around specific business functions and can be deployed independently using automated deployment systems. Figure 2.29 illustrates how secure, managed, and monetized APIs that are critical for a digital enterprise can be created by leveraging a combination of API-led integration frameworks and cloud-native technologies. The use of such frameworks and technologies helps streamline the migration of legacy business solutions. The process of **migrating legacy business solutions** means upgrading or replacing old systems with newer, more efficient ones. In addition to these capabilities, the adaptive design reuse approach may be used to create the custom part of the cloud-native applications. More information related to the development of cloud-native solutions is provided in Chapter 9 Software Engineering, Chapter 10 Enterprise and Solution Architectures Management, and Chapter 12 Cloud-Native Applications Development.

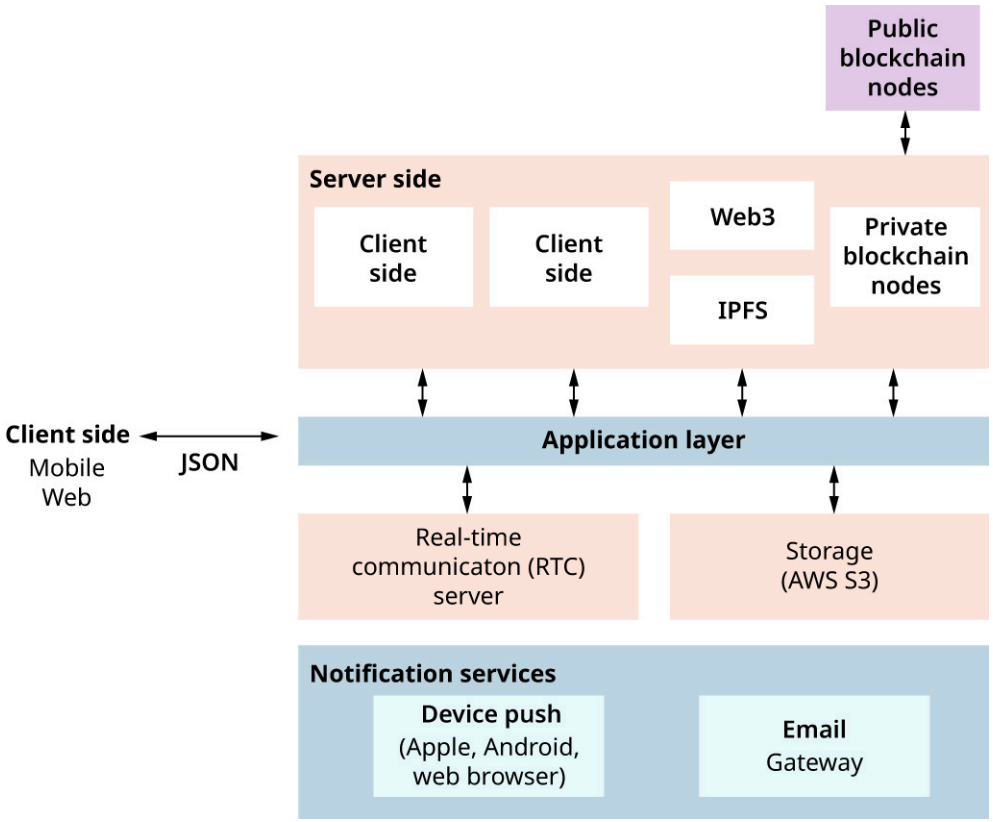


Figure 2.29 The cloud-native application architecture view of a digital enterprise shows both client-side and server-side processes through each layer. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Figure 2.29 illustrates the architecture of a digital platform that combines web and mobile applications with blockchain technology. On the client side, users interact with the platform through web dashboards and mobile apps, which communicate with the server using JSON and handle notifications via services like Firebase, Huawei, and Apple. The server side includes an API layer that processes these requests, a caching layer to improve performance, and a back-end logic layer responsible for application logic, backups, and analytics. The architecture also features integration with public blockchain networks for enhanced security and transparency, and it supports various notification services to keep users informed.

GLOBAL ISSUES IN TECHNOLOGY

Scale Transformations

The approach that consists of reinventing, rethinking, and rewiring solutions in various industries seems to

favor countries that have the means to perform broadscale transformations. This may have ethical, social, and economic implications in other parts of the world.

Consider how advanced countries are rapidly adopting electric cars. They have the resources to reinvent transportation by investing in electric vehicle (EV) technology, rethinking their energy use to reduce pollution, and rewiring their infrastructure to support EV charging stations. This shift toward electric cars is more challenging in less wealthy countries due to the high costs of EVs and the lack of charging infrastructure. As a result, these countries may continue to depend on older, more polluting vehicles, facing both environmental and economic disadvantages.

Innovative Cloud Mashups

Innovative cloud mashups refer to creative combinations of different innovative business solutions that leverage disruptive technologies such as IoT, big data analytics, machine learning, blockchain, and others that can be quickly assembled today as hybrid cloud applications. A **hybrid cloud application** combines the benefits of both private and public clouds, allowing organizations to optimize their infrastructure based on specific requirements.

Internet of Things (IoT) refers to the network of physical devices embedded with sensors, software, and connectivity, enabling them to collect and exchange data. The process of examining, processing, and extracting valuable insights from large datasets is called **big data analytics**. Developing algorithms that enable computers to learn from data and make decisions without explicit programming is called **machine learning**. This is made possible by creating mashups of platform services provided by various public cloud vendors to gain access to these disruptive technologies.

Figure 2.30 and Figure 2.31 illustrate models of solutions that are used today to support a variety of mobile health (MHealth), body area networks (BANs), emotions monitoring, and social media applications. In addition to the capabilities provided by the Big Clouds, the adaptive design reuse approach may be used to create the custom part of these hybrid solutions. Google Maps and Zillow are prime examples of applications that utilize location-based data to deliver valuable services. A GPS device identifies a user's location, and that information flows through the central network. Apps then display this data in a user-friendly manner, connecting users with real-time geographic information in Google Maps or housing market details in Zillow. The integration of GPS with other IoT systems allows for the seamless presentation of customized, location-specific content to enhance the user experience. More information related to the development of web solutions is provided in Chapter 9 Software Engineering, Chapter 10 Enterprise and Solution Architectures Management, and Chapter 13 Hybrid Multicloud Digital Solutions Development.

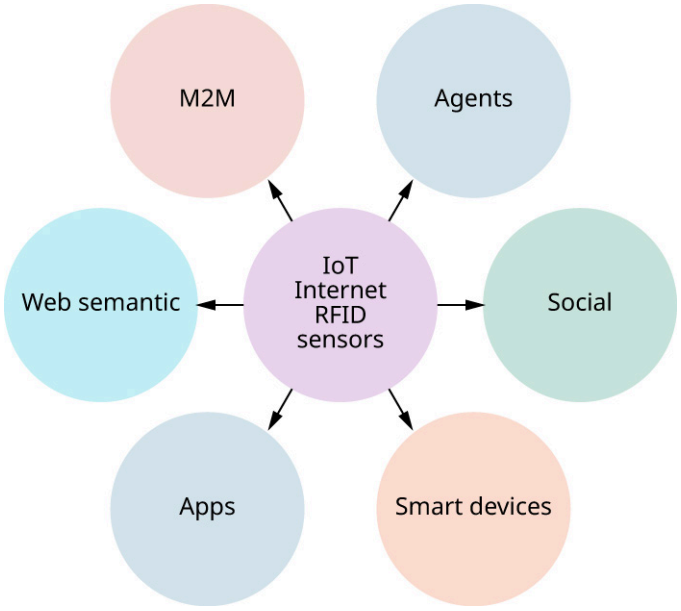


Figure 2.30 IoT devices use sensors, applications, and connectivity to interact, collect, and exchange data. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

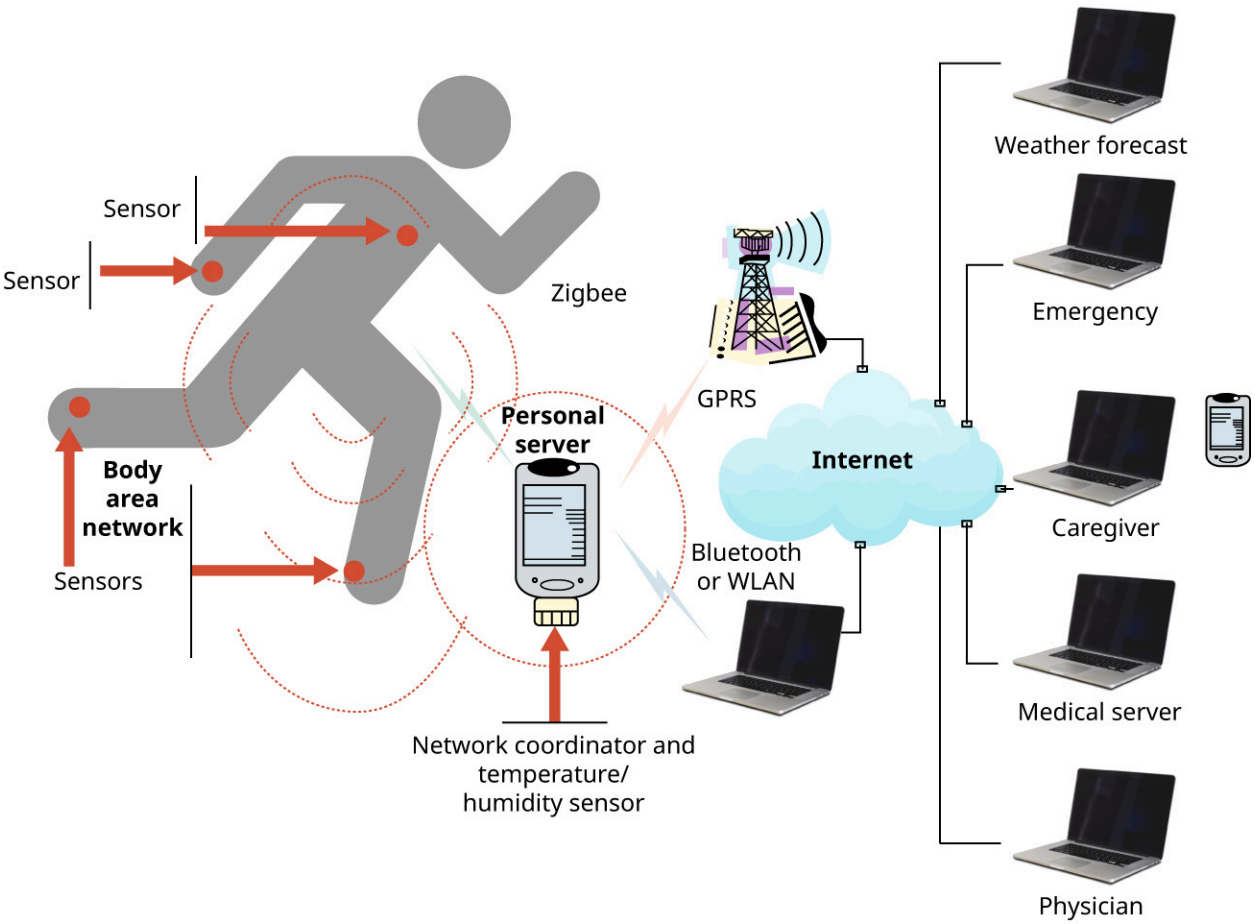


Figure 2.31 This diagram depicts an architecture of a body area network. (credit: modification of "Body Area Network" by "Jtel"/Wikimedia Commons, Public Domain)

Web 3.0 enables businesses to create more personalized and predictive services for users, fostering greater trust and engagement by giving users control over their own data. For companies, this translates to new opportunities for collaboration, innovation, and reaching consumers directly without intermediaries, ultimately

driving more efficient business models and creating value in ways that were not possible with earlier web technologies.

Chapter Review

Key Terms

abstraction simplified representation of complex systems or phenomena
application architecture subset of the enterprise solution architecture; includes a process to architect and design the application architecture as well as the actual application architecture model, which represents the content of the application architecture
architectural pattern reusable solution to a recurring problem in software architecture design
architecture model represents the content of the application architecture
architecture scope extent and boundaries within which architectural considerations, decisions, and solutions apply
automation using a program or computer application to perform repetitive tasks or calculations
big data analytics process of examining, processing, and extracting valuable insights from large datasets
blockchain secure and transparent way of recording transactions; uses a chain of blocks, each storing a list of transactions
blueprint detailed plan or design that outlines the structure, components, and specifications of a building, product, system, or process
business logic layer holds the business logic for the business solution or application
business model framework that outlines how a business creates, delivers, and captures value
business process hierarchy organizes a company's activities from broad, general processes down to specific tasks, making it easier to manage and improve how the business operates
computational thinking problem-solving and cognitive process rooted in principles derived from computer science that involves breaking down complex problems into smaller, more manageable parts and devising systematic approaches to solve them
data architecture model conceptual framework that outlines how an organization structures, organizes, and manages its data assets
data management layer responsible for interacting with persistent storage systems like databases and various data processing mechanisms
data modeling collaborative process wherein IT and business stakeholders establish a shared understanding of essential business terms, known as entities, which typically end up being represented as tables that contain data in a relational database management system
debugging finding and fixing of issues in code
decomposition solving of a complex problem by breaking it up into smaller, more manageable tasks
design component reusable element within a larger system that serves a specific purpose
EA domain represents business, data, application, and technology architectures
elementary business process (EBP) fundamental and indivisible activity within a business that is not further subdivided into smaller processes
entity model represents the various objects or concepts and their relationships within a system
Ethereum platform open-source blockchain platform that enables the creation and execution of smart contracts and decentralized applications
flowchart method for showing the flow and direction of decisions in a visual way using a diagram
function set of commands that can be repeatedly executed
heuristic form of a pattern that is well-known and considered a rule of thumb
hybrid cloud application combines the benefits of both private and public clouds, allowing organizations to optimize their infrastructure based on specific requirements
Internet of Things (IoT) network of physical devices embedded with sensors, software, and connectivity, enabling them to collect and exchange data
machine learning developing algorithms that enable computers to learn from data and make decisions without programming

microservices way of building software by breaking it into small, independent pieces; each piece, or service, does a specific job and works on its own
Microsoft Azure comprehensive cloud computing platform provided by Microsoft
migrating legacy business solutions upgrading or replacing old systems with newer, more efficient ones
Model-View-Controller (MVC) software architectural pattern commonly used in the design of interactive applications, providing a systematic way to organize and structure code
monolithic structure system or application architecture where all the components are tightly integrated into a single unit
presentation layer user's touchpoint, handling the user interface (UI) and delivering the user experience (UX), which encapsulates the overall feel and interaction a person has with a system or service
process map displays the order of chosen processes from the process hierarchies, highlighting their connection to the roles responsible for executing them
pseudocode outline of the logic of algorithms using a combination of language and high-level programming concepts
recursion programming and mathematical concept where a function calls itself during its execution
smart contract automated agreement written in code that runs on blockchain technology
solution architecture structural design that is meant to address the needs of prospective solution users
solutions continuum strategy where existing solutions, components, or patterns are leveraged and adapted for use in different contexts
system architecture deals with application and data, and how they are related to each other and what business process they support together
technical architecture includes the software and hardware capabilities to fully enable application and data services
user experience (UX) overall experience that a person has when interacting with a product, service, or system
Web 2.0 second generation of the World Wide Web when we shift from static web pages to dynamic content
Web 3.0 third generation of the World Wide Web and represents a vision for the future of the Internet characterized by advanced technologies
web application (web app) software application that is accessed and interacted with through a web browser over the Internet
web application framework built-in support for architectural patterns that make it easy to extend the framework and use plug-ins to implement commercial-grade websites in a reasonable amount of time
World Wide Web Consortium (W3C) international community that develops guidelines to ensure the long-term growth and accessibility of the World Wide Web

Summary

2.1 Computational Thinking

- Complex problems are situations that are difficult because they involve many different parts or factors.
- Computational thinking means breaking these problems into smaller parts, understanding how these parts relate to each other, and then coming up with effective strategies or steps to solve each part.
- Computational thinking is a set of tools or strategies for solving (and learning how to solve) complex problems that relate to mathematical thinking in its use of abstraction, decomposition, measurement, and modeling.
- Characterization of computational thinking is the three As: abstraction, automation, and analysis.
- Decomposition is a fundamental concept in computational thinking, representing the process of systematically breaking down a complex problem or system into smaller, more manageable parts or subproblems.
- Logical thinking and pattern recognition are computational thinking techniques that involve the process of identifying similarities among and within problems.
- Abstraction is a computational thinking technique that centers on focusing on important information

while ignoring irrelevant details.
- Algorithms are detailed sets of instructions to solve a problem step-by-step.
- Testing and debugging is about finding and fixing mistakes in the step-by-step instructions or algorithms used to solve a problem.

2.2 Architecting Solutions with Adaptive Design Reuse in Mind

- Computational thinking commonly employs a bottom-up strategy for crafting well-structured components.
- A business solution architecture is a structural design that is meant to address the needs of prospective solution users.
- Business solutions are strategies/systems created to solve specific challenges in a business. Designing business solutions can be described as a complex systemic process that requires expertise in various spheres of technology as well as the concerned business. A blueprint is a detailed plan or design that outlines the structure, components, and specifications of a building, product, system, or process.
- Two heuristics are inherent to the design of business solutions and the creation of business solution architectures. Layering in business solution architecture involves creating distinct layers that abstract specific aspects of the overall architecture. The layering approach relies on the principle of separation of concerns. The presentation layer holds the user interface (UI) that interacts with the outside world.
- User experience (UX) refers to the overall experience that a person has when interacting with a product, service, or system.
- A monolithic structure is a system or application architecture where all the components are tightly integrated into a single unit.
- Enterprise-level architecture encompasses various domains that define the structure, components, and operations of an entire organization. Enterprise architecture (EA) views the enterprise as a system or a system of systems.
- The enterprise business architecture (EBA) is a comprehensive framework that defines the structure and operation of an entire organization. A business model is a framework that outlines how a business creates, delivers, and captures value. The organizational model is the structure and design of an organization, outlining how roles, responsibilities, and relationships are defined.
- The business process is a series of interrelated tasks, activities, or steps performed in a coordinated manner within an organization to achieve a specific business goal.
- Location model refers to a set of rules used to analyze and make decisions related to the positioning of entities, activities, or resources.
- The enterprise technology architecture (ETA) is a comprehensive framework that defines the structure, components, and interrelationships of an organization's technology systems to support its business processes and objectives.
- The application architecture is a subset of the enterprise solution architecture.
- A data architecture model is a conceptual framework that outlines how an organization structures, organizes, and manages its data assets.
- Data modeling is the collaborative process wherein IT and business stakeholders establish a shared understanding of essential business terms, known as entities.
- Architecture views are representations of the overall system design that matter to different stakeholders.

2.3 Evolving Architectures into Useable Products

- The combination of top-down, adaptive design reuse and bottom-up, computational thinking optimizes modern software development.
- Model-View-Controller (MVC) is a software architectural pattern commonly used in the design of interactive applications, providing a systematic way to organize and structure code.
- The adaptive design reuse approach is a strategy in software development that emphasizes the efficient reuse of existing design solutions to create new systems or applications.
- World Wide Web Consortium (W3C) is an international community that develops guidelines to ensure the

long-term growth and accessibility of the World Wide Web.

- Web 2.0 is the second generation of the World Wide Web when we shift from static web pages to dynamic content.
- Web 3.0 is the third generation of the World Wide Web and represents a vision for the future of the Internet characterized by advanced technologies.
- A web application (web app) refers to a software application that is accessed and interacted through a web browser over the Internet.
- Blockchain is a secure and transparent way of recording transactions. It uses a chain of blocks, each storing a list of transactions.
- Microservices is a way of building software by breaking it into small, independent pieces. Each piece, or service, does a specific job and works on its own.
- Migrating legacy business solutions means upgrading or replacing old systems with newer, more efficient ones.
- Innovative cloud mashups refer to creative combinations of different innovative business solutions that leverage disruptive technologies.

Review Questions

1. What term is a problem-solving and cognitive process rooted in principles derived from computer science that involves breaking down complex problems into smaller, more manageable parts and devising systematic approaches to solve them?
   a. abstraction
   b. decomposition
   c. computational thinking
   d. recursion

2. What shape in a flowchart represents a decision point?
   a. oval
   b. parallelogram
   c. rectangle
   d. diamond

3. What does pseudocode spell out in natural language?
   a. an algorithm
   b. a test case to debug
   c. a flowchart
   d. the programming language of choice

4. After a test case fails, what is the next step to determine the cause of the failure?

5. What are the key elements of CT that distinguish CT from other types of problem-solving strategies?

6. What is the primary difference between a heuristic and a pattern?
   a. A heuristic is a general rule used for quick problem-solving when an exact solution is not possible, while a pattern is a repeatable solution to a commonly occurring problem.
   b. A heuristic and a pattern are both specific guidelines used to achieve exact solutions in complex problems.
   c. A heuristic is used for creating new designs, whereas a pattern refers to repeating decorative motifs.
   d. There is no difference; both terms refer to specific scientific methods used in research.

7. What level of architecture is described as having a narrower scope, a detailed blueprint, and a lower level of abstraction?
   a. system architecture
   b. technical architecture

c. enterprise architecture
d. solution architecture

8. What level of architecture is described as having a wider scope, a vague plan for the entire organization, and a higher level of abstraction?
a. system architecture
b. technical architecture
c. enterprise architecture
d. solution architecture

9. What component holds the business logic for the business solution or application?
a. presentation layer
b. data management layer
c. business logic layer
d. business process hierarchy

10. What is the difference between data, information, knowledge, and wisdom?

11. Explain why an information system architecture is considered an architecture view in TOGAF.

12. Once architectural similarities have been identified between the architecture of a new problem and existing architectural solutions, what is required to apply these patterns effectively?
a. a comprehensive understanding of the new problem's requirements and constraints
b. the ability to modify existing patterns to fit the new problem's unique context
c. both a comprehensive understanding of the new problem's requirements and the ability to modify existing patterns
d. approval from a higher authority to use the identified patterns

13. In the Model-View-Controller, what layer is responsible for acting as an intermediary between two layers?
a. view
b. model
c. business logic
d. controller

14. What does Web 3.0 provide that Web 2.0 did not?
a. dynamic web pages as opposed to only static web pages
b. represents a vision for the future of the Internet characterized by advanced technologies
c. shifted from HTTP to HTTPS
d. based on JSON as opposed to HTML

15. A smart home with a thermostat, a refrigerator, and lights that all can be controlled remotely is an example of devices that can be described with what terminology?
a. Internet of Things (IoT)
b. machine learning
c. hybrid cloud application
d. solutions continuum

16. Why doesn't TOGAF provide prescriptive methods to create and manage repositories of architectural patterns?

17. What is a responsive web application?

18. What is a cloud mashup?

Conceptual Questions

1. Suppose you plan to meet with your friends at a location you are unfamiliar with. In what ways could you employ computational thinking to efficiently navigate and locate the meeting spot?
2. Explain how the pyramid of knowledge concept helps describe the learning progress you make when reading a textbook.
3. What are specific examples of business architecture similarities between two banks?
4. What are specific examples of technology architecture similarities between two banks?

Practice Exercises

1. Think of a complex problem—one that can be broken into many layers of smaller problems. Explain how computational thinking could help you develop a solution to your complex problem.
2. Look at the following pseudocode that describes an algorithm to make a peanut butter and jelly sandwich:
    a. Get the peanut butter.
    b. Get the jelly.
    c. Get the bread.
    d. Open the peanut butter jar.
    e. Open the jelly jar.
    f. Open the bread.
    g. Take out slice of bread.
    h. Take out another slice of bread.
    i. Dip the knife into the peanut butter.
    j. Spread the peanut butter on one slice of bread.
    k. Dip the knife into the jelly.
    l. Spread the jelly on the other slice of bread.
    m. Put the two slices of bread together.

    Write a new algorithm that utilizes abstraction to simplify the number of steps of the original algorithm and can be used as a pattern to make any sandwich.
3. Research what the Fibonacci number sequence is. Write the pseudocode to compute the *n*th number in the Fibonacci number sequence. Utilize recursion to model a pattern of computation.
4. Create a model that describes the business of running your daily life. Please note that this is not suggesting that you should run your life as a business. *Hint*: To answer this question, think about the various players, locations, and processes involved in your daily activities and create simple models that mimic the structure provided for the trading business model in the current chapter section.
5. Draw an application architecture diagram for a business solution that uses smart contracts for payment and transactions logging purposes. Feel free to leverage some of the figures from this chapter, rather than create something new.

Problem Set A

1. Create an algorithm to explain to a robot how to cross a street. Use computational thinking to break down the problem into smaller parts. Use the following information to guide your thinking.

Task	Decomposition	Pattern Recognition	Abstraction	Algorithm
Crossing the road	Vehicles, actions, decision	Identify the different considerations you can group together to form a pattern of what needs to be done.	Act out crossing the road. Do you do something differently from someone else?	Write your instructions in either pseudocode or as a flowchart.

2. Create an algorithm to explain how to bake a four-tiered wedding cake.
3. Reflect on what happens when you try to figure out driving directions from point A to point B.
4. Create an enterprise architecture business model for an insurance company that specializes in insuring home and car owners.
5. Create two alternative enterprise technology architecture models for the insurance company business model created in the previous question.
6. Draw an application architecture diagram for the Web 2.0 responsive website of a fictitious insurance company that focuses on home and car insurance and assume that the company also provides native apps to its customers in addition to the website.

Problem Set B

1. Create an algorithm to explain to a robot how to play a game of rock paper scissors. Use computational thinking to break down the problem into smaller parts. Use the following information to guide your thinking.

Task	Decomposition	Pattern Recognition	Abstraction	Algorithm
Rock, paper, scissors	Actions, choices, timings, winning conditions	Identify the different considerations you can group together to form a pattern of what needs to be done.	Play the game. Think of the actions you perform.	Write your instructions in either pseudocode or a flowchart.

2. Create an algorithm to show a robot how to play a game of tic-tac-toe. Use computational thinking to break down the problem into smaller parts. Use the following information to guide your thinking.

Task	Decomposition	Pattern Recognition	Abstraction	Algorithm
Tic-tac-toe	Moves that can be made, winning conditions	Identify the different considerations you can group together to form a pattern of what needs to be done.	Play against someone. What strategies do you use in order to win?	Write your instructions in either pseudocode or a flowchart.

3. Perform some research on the Internet to piece together enterprise architectures for as many industries as you can think of.

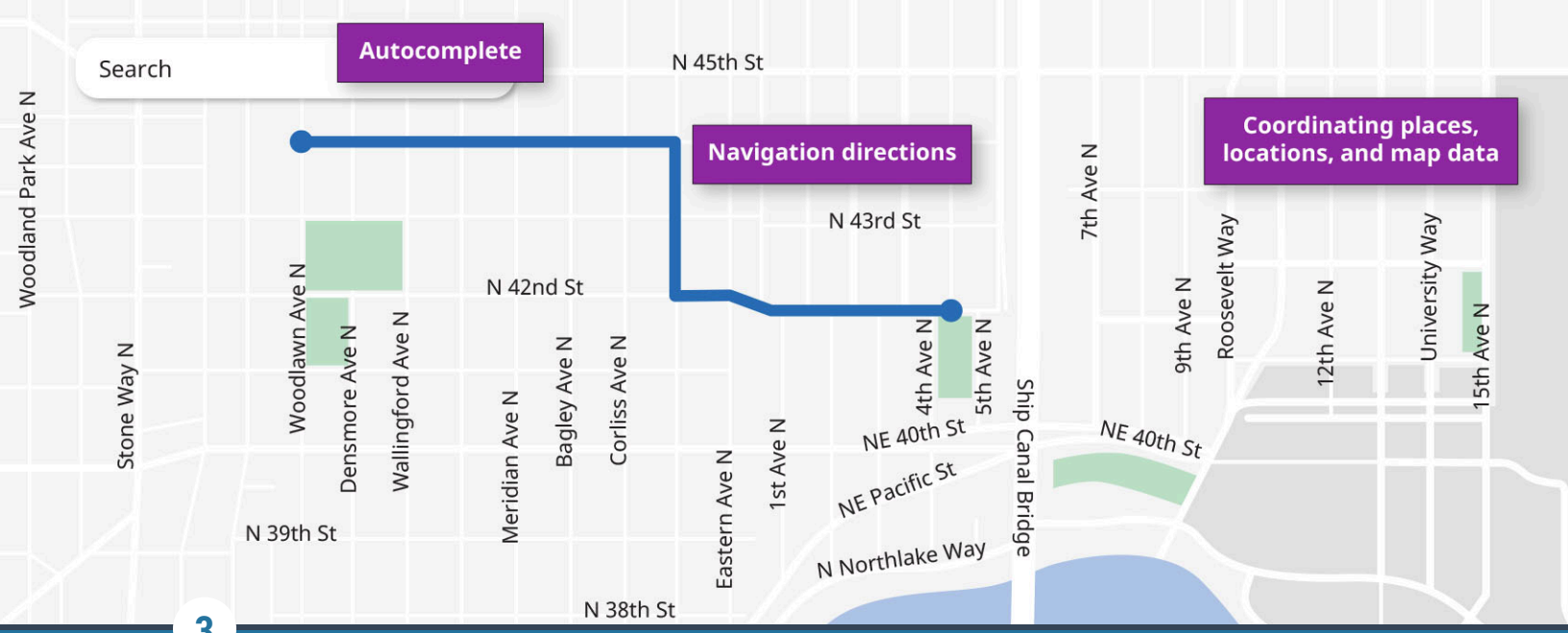
4. Draw a cloud-native application architecture diagram for the trading business and technical model documented in the previous section of this chapter.
5. A company wants to develop a business solution that takes pictures of the license plates of cars that drive too fast through intersections in a given city, sends tickets to the drivers, and manages ticket payments. Draw an innovative cloud mashup application architecture diagram for such a solution. Please note that IoT, machine learning, and blockchain PaaS services should be used as part of your design.
6. Document the architecture of a pattern catalog that could be used to provide access to solution architecture diagrams that would help accelerate the creation of mainstream business solutions.

Thought Provokers

1. Consider TechWorks, which is 100% committed to leveraging innovative technologies as a business growth facilitator. Describe how it can best use computational thinking to create products or services that can generate business. Give precise examples and explain how the start-up would be able to scale the resulting business (i.e., keep sustaining the cost of doing business while increasing its number of customers). *Hint*: Some companies leverage an incubation arm to come up with innovative ideas and then accelerate the process of developing these ideas into practical solutions via a solution accelerator.
2. Consider our start-up company that is 100% committed to leveraging innovative technologies as a business growth facilitator. Describe how it can best use adaptive design reuse to create products or services that can generate business. Give precise examples and explain how the start-up would be able to scale the resulting business (i.e., keep sustaining the cost of doing business while increasing its number of customers)? *Hint*: The company may decide to sell reusable design models and their implementation from a proprietary catalog; it may also focus on providing consulting services to derive complete solutions from its proprietary models.
3. Consider our start-up company that is 100% committed to leveraging innovative technologies as a business growth facilitator. Describe how it can best leverage evolving architectures into usable products to create products or services that can generate business. Give precise examples and explain how the start-up would be able to scale the resulting business (i.e., keep sustaining the cost of doing business while increasing its number of customers).

Labs

1. Perform some research on the Internet to find examples of problem scenarios that computational thinking may help solve and create a catalog of problem scenarios. Then, elaborate and show practically how this catalog may be used to compare the scenarios and classify them so they may be used as part of your pattern discovery as you apply computational thinking to new problem scenarios.
2. Create an enterprise architecture capability model for a company of your choice using your research from problem set B. Then, expand one of the capabilities and provide business and technology architecture models for it; identify a project within the capability you expanded upon and provide a complete solution architecture for it.
3. Perform some research on the Internet to piece together additional solutions architecture diagrams for the various categories of mainstream solutions covered in this chapter section. This should include application, data, and technology diagrams.
4. Catalog additional types of solution architectures that may be used to accelerate the creation of mainstream business solutions.
5. Apply critical thinking strategies to develop a study plan for your current semester's courses, aiming to achieve an A or pass each course.



3

Data Structures and Algorithms

Figure 3.1 Online mapping applications represent places, locations, and map data while providing functionality to look around, search for places, and get navigation directions. The right combination of data structures to manage collections of places, locations, and map data along with efficient search and navigation algorithms will help optimize the experience of users trying to find their way through the map and will also make optimal use of computing resources. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; data source: OpenStreetMap under Open Database License; attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Chapter Outline

3.1 Introduction to Data Structures and Algorithms
3.2 Algorithm Design and Discovery
3.3 Formal Properties of Algorithms
3.4 Algorithmic Paradigms
3.5 Sample Algorithms by Problem
3.6 Computer Science Theory

Introduction

Online maps help people navigate a rapidly changing world. It was not long ago that maps were on paper and that knowledge came from non-digital, trusted sources. In this chapter, we will study how computer scientists design and analyze the foundational structures behind many of today's technologies. Data structures and algorithms are not only foundational to map apps, but also enable an amazing variety of other technologies too. From self-driving cars to inventory management to simulating the movement of galaxies to transferring data between computers—all these applications use data structures and algorithms to efficiently organize and process large amounts of information.

3.1 Introduction to Data Structures and Algorithms

Learning Objectives

By the end of this section, you will be able to:

- Understand the difference between algorithms and programs
- Relate data structures and abstract data types
- Select the data structure that is appropriate to solve a given problem practically

Computer science is the study of computers and computational systems that involve data representation and process automation. Owing to their historical roots as calculators, computers can easily represent numerical data. Calculators rely on algorithms to add, subtract, multiply, and divide numbers. But what about more complex data? How do computers represent complex objects like graphs, images, videos, or sentences? What complications arise when we represent or process data in certain ways? These are some of the foundational questions that computer scientists and programmers focus on when designing software and applications that we use to solve problems.

A **data type** determines how computers process data by defining the possible values for data and the possible functionality or operations on that data. For example, the integer data type is defined as values from a certain range of positive or negative whole numbers with functionality including addition, subtraction, multiplication, and division. The string data type is defined as a sequence of characters where each character can be a letter, digit, punctuation, or space, with functionalities that include adding or deleting a character from a string, concatenating strings, and comparing two strings based, for example, on their alphabetical order.

Data types like strings are an example of **abstraction**, the process of simplifying a concept in order to represent it in a computer. The string data type takes a complex concept like a sentence and represents it in terms of more basic data that a computer can work with. When a computer compares two strings, it is really comparing the individual numerical character codes (see Chapter 5 Hardware Realizations of Algorithms: Computer Systems Design) corresponding to each pair of characters within the two strings.

In this section, we will learn how to solve problems by choosing abstractions for complex data. We will see that just as our data grows more complex, so do our algorithms.

Introduction to Algorithms

An algorithm is a sequence of precise instructions that operate on data. We can think of recipes, plans, or instructions from our daily lives as examples of algorithms. Computers can only execute a finite pre-defined set of instructions exactly as instructed, which is why programming can feel like such a different way of communicating as compared to our natural human languages. A **program** is an implementation (realization) of an algorithm written in a formal programming language.

Although each programming language is different from all the others, there are still common ideas across all of them. Knowing just a few of these common ideas enables computer scientists to address a wide variety of problems without having to start from scratch every single time. For example, the abstraction of string data enables programmers to write programs that operate on human-readable letters, digits, punctuation, or spaces without having to determine how to delve into each of these concepts. Programming languages allow us to define abstractions for representing ideas in a computer (see Chapter 4 Linguistic Realization of Algorithms: Low-Level Programming Languages for more).

The study of data structures and algorithms focuses on identifying what is known as a **canonical algorithm**: a well-known algorithm that showcases design principles helpful across a wide variety of problems. In this chapter, rather than focusing on the programming details, we will instead focus on algorithms and the ideas behind them.

Understanding Data Structures

For many real-world problems, the ability to design an algorithm depends on how the data is represented. A **data structure** is a complex data type with two equally important parts:

1. a specific **representation** or way of organizing a collection of more than one **element**, which is an individual value or data point, and
2. a specific **functionality** or operations such as adding, retrieving, and removing elements.

In our previous example, a string is a data structure for representing sentences as a sequence of characters. It has specific functionality such as character insertion or deletion, string concatenation, and string comparison.

Although the values for complex data are often diverse, computer scientists have designed data structures so that they can be reused for other problems. For example, rather than designing a specialized data structure for sentences in every human language, we often use a single, universal string data structure to represent sentences, including characters from different languages in the same sentence. (We will later see some drawbacks of generalizing assumptions in the design of data structures and algorithms.) In addition, computers take time to execute algorithms, so computer scientists are concerned about efficiency in terms of how long an algorithm takes to compute a result.

Among the different types of universal data structures, computer scientists have found it helpful to categorize data structures according to their functionality without considering their specific representation. An **abstract data type (ADT)** consists of all data structures that share common functionality but differ in specific representation.

Common abstract data types for complex data follow, and list and set types are shown in Figure 3.2. We will discuss each abstract data type in more detail together with their data structure implementations.

- A **list** represents an ordered sequence of elements and allows adding, retrieving, and removing elements from any position in the list. Lists are indexed because they allow access to elements by referring to the element's **index**, which is the position or address for an element in the list. For example, a list can be used to represent a to-do list, where each item in the list is the next task to be completed in chronological order.
- A **set** represents an unordered collection of unique elements and allows adding, retrieving, and removing elements from the set. Sets typically offer less functionality than lists, but this reduction in functionality allows for more efficient data structure representations. For example, a set can be used to represent the names of all the places that you want to visit in the future.
- A **map** represents unordered associations between key-value pairs of elements, where each key can only appear once in the map. A map is also known as a dictionary since each term (key) has an associated definition (value). Maps are often used in combination with other data structures. For example, a map can be used to represent a travel wish list: each place that you want to visit in the future can be associated with the list of things that you want to do when you arrive at a given place.
- A **priority queue** represents a collection of elements where each element has an associated priority value. In addition to adding elements, priority queues focus on retrieving and removing the element with the highest priority. For example, a priority queue can be used to represent inpatient processing at a hospital emergency room: the patients with more urgent need for care may be prioritized and dealt with first.
- A **graph** represents binary relations among a collection of entities. More specifically, the entities are represented as vertices in the graph, and a directed or undirected edge is added between two vertices to represent the presence or absence of a certain relation. For example, a friendship graph can be used to represent the friendship relations between people, in which case an undirected edge is added between two persons if they are friends. Graphs allow operations such as adding vertices and edges, removing vertices and edges, and retrieving all edges adjacent to a given vertex.

List

Index	0	1	2	3	4	5	6	7
Data	22	39	45	62	69	79	90	98

Set

Index	0	1	2	3	4	5	6	7
Data	22	39	98	45	69	79	65	90

Index	0	1	2	3	4	5	6	7
Data	DC	Alabama	California	Wyoming	New York	Florida	Texas	Arizona

Figure 3.2 Lists and sets are common abstract data types used to represent complex data and can be in the form of integers or string data. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Selecting a Data Structure

Since data representation is a fundamental task in designing algorithms that solve problems, how do we select

data structures for a particular problem? Computer scientists apply a top-down approach.

1. Select an appropriate abstract data type by analyzing the problem to determine the necessary functionality and operations.
2. Select an appropriate data structure by quantifying the resource constraints (usually program running time) for each operation.

The primary concern is the data and the operations to be performed on them. Thinking back to simple data types like numbers, we focused on addition, subtraction, multiplication, and division as the basic operations. Likewise, to represent complex data types, we also focus on the operations that will most directly support our algorithms. After deciding on an abstract data type, we then choose a particular data structure that implements the abstract data type.

Linear Data Structures

If a problem can be solved with an ordered sequence of elements (e.g., numbers, payroll records, or text messages), the simplest approach might be to store them in a list. Some problems require that actions be performed in a strict chronological order, such as processing items in the order that they arrive or in the reverse order. In these situations, a **linear data structure**, which is a category of data structures where elements are ordered in a line, is appropriate. There are two possible implementations for the list abstract data type. The first, an **array list** (Figure 3.3), is a data structure that stores list elements next to each other in memory. The other is a **linked list** (Figure 3.4), which is a list data structure that does not necessarily store list elements next to each other, but instead works by maintaining, for each element, a link to the next element in the list. Both array lists and linked lists are linear data structures because their elements are organized in a line, one after the other. An advantage of array lists is that they allow (random) access to every element in the list in a single step. This is in sharp contrast with linked lists, which only supports "sequential access." On the other hand, linked lists support fast insertion and deletion operations, which array lists do not.

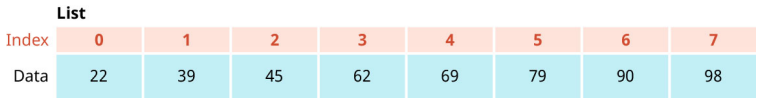


Figure 3.3 An array list stores elements next to each other in memory in the exact list order. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

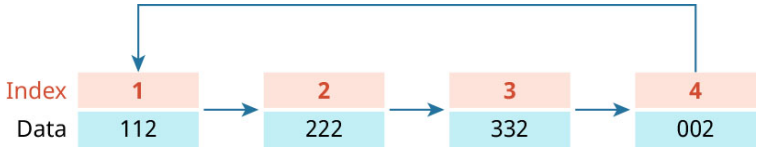


Figure 3.4 A linked list maintains a link for each element to the next element in the list. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Earlier, we introduced sets as offering less functionality than lists. Both array lists and linked lists can also implement the set abstract data type. Sets differ from lists in two ways: sets are unordered—so elements are not assigned specific positions—and sets only consist of unique elements. In addition to implementing sets, linear data structures can also implement the map, priority queue, and graph abstract data types. If linear data structures can cover such a wide range of abstract data types, why learn other data structures? In theory, any complex data can be represented with an array list or a linked list, although it may not be optimal, as we will explain further.

One drawback of relying only on linear data structures is related to the concept of efficiency. Even if linear data structures can solve any problem, we might prefer more specialized data structures that can solve fewer problems more efficiently, and help represent real world data arrangements more closely. This is particularly useful when we have large amounts of data like places or roads in an online map of the entire world. Linear data structures ultimately organize elements in a line, which is necessary for implementing lists but not necessary for other abstract data types. Other data structures specialize in implementing sets, maps, and

priority queues by organizing elements in a hierarchy rather than in a line.

Tree Data Structures

A **tree** is a hierarchical data structure. While there are many kinds of tree data structures, all of them share the same basic organizing structure: a **node** represents an element in a tree or graph. A node may or may not have a descendant. A **child node** is a descendant of another node. Often, the primary node is referred to as the "parent node." Trees maintain a hierarchy through parent-child relationships, which repeat from the **root node** at the top of the tree down to each **leaf node**, which is at the bottom of the tree and has no children. The height of a tree corresponds to the depth of the hierarchy of descendants. Figure 3.5 illustrates the structure and elements of a tree.

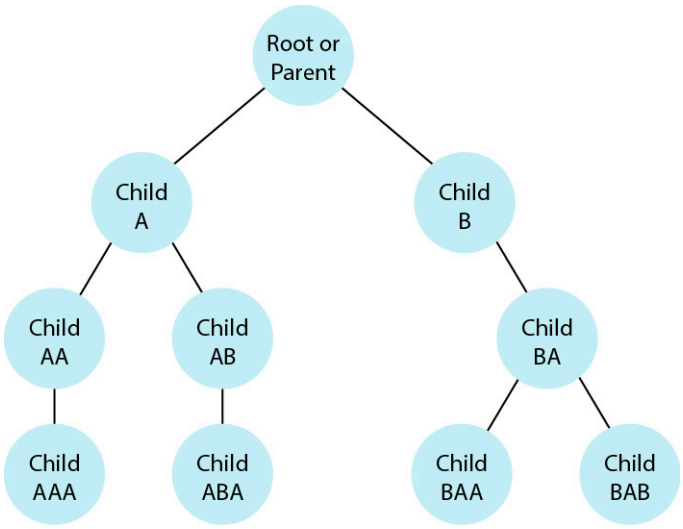


Figure 3.5 A tree is a hierarchical data structure with nodes where each node can have zero or more descendant child nodes. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Binary Search Trees

A **binary search tree** is a kind of tree data structure often used to implement sets and maps with the **binary tree property**, which requires that each node can have either zero, one, or two children, and the **search tree property**, which requires that elements in the tree are organized least-to-greatest from left-to-right. In other words, the values of all the elements of the left subtree of a node have a lesser value than that of the node. Similarly, the values of all the elements of the right subtree of a node have a greater value than that of the node. The search tree property suggests that when elements are read left-to-right in a search tree, we will get the elements in sorted order. For numbers, we can compare and sort numbers by their numeric values. For more complex data like words or sentences, we can compare and sort them in dictionary order. Binary search trees use these intrinsic properties of data to organize elements in a searchable hierarchy (Figure 3.6).

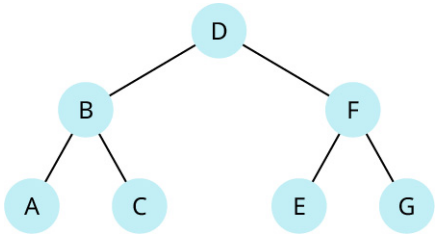


Figure 3.6 A binary search tree organizes elements least to greatest from left to right. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The tree illustrated satisfies the binary tree property based on the natural alphabetical order between letters since the elements in the tree are organized least to greatest from left to right. In other words, for a given list of letters (A, B, C, D, E, F, G), start at the middle of the list of letters with D (the root node) then pick B as the left

sub-node of D which is at the middle of the list of letters (A, B, C) that is on the right of D and pick F as the right sub-node of D, which is at the middle of the list letters (E, F, G) that is on the left of D. Finally, organize the remaining letters under sub-nodes B and F to ensure that they are least-to-greatest from the left to right.

The search tree property is responsible for efficiency improvements over linear data structures. By storing elements in a sorted order in the search tree rather than in an indexed order in a list, binary search trees can more efficiently find a given element. Consider how we might look up words in a dictionary. A binary search tree dictionary storing all the terms and their associated definitions can enable efficient search by starting at the middle of the dictionary (the root node) before determining whether to go left or right based on whether we expect our word to appear earlier or later in the dictionary order. If we repeat this process, we can repeatedly rule out half of the remaining elements each time. Searching for a term in a list-based dictionary that is not sorted, on the other hand, would require us to start from the beginning of the list and consider every word until the end of the list since there is no underlying ordering structure to the elements.

Balanced Binary Search Trees

Binary search trees are not as effective as we have described. The dictionary example represents a best-case scenario for binary search trees. We can only rule out half of the remaining elements each time if the binary search tree is **perfectly balanced**, which means that for every node in the binary search tree, its left and right subtrees contain the same number of elements. This is a strong requirement, since the order in which elements are added to a binary search tree determines the shape of the tree. In other words, binary search trees can easily become unbalanced. It is possible for a binary search tree to look exactly like a linked list, in which each node contains either zero children or one child, which is no more efficient than a linear data structure.

An **AVL tree** (named after its inventors, Adelson-Velsky and Landis) is a balanced binary search tree data structure often used to implement sets or maps with one additional tree property: the **AVL tree property**, which requires the left and right subtrees to be balanced at every node of the tree. AVL trees are just one among many "self-balancing" binary search trees. A **balanced binary search tree** introduces additional properties that ensure that the tree reorganizes elements to maintain balance (Figure 3.7).

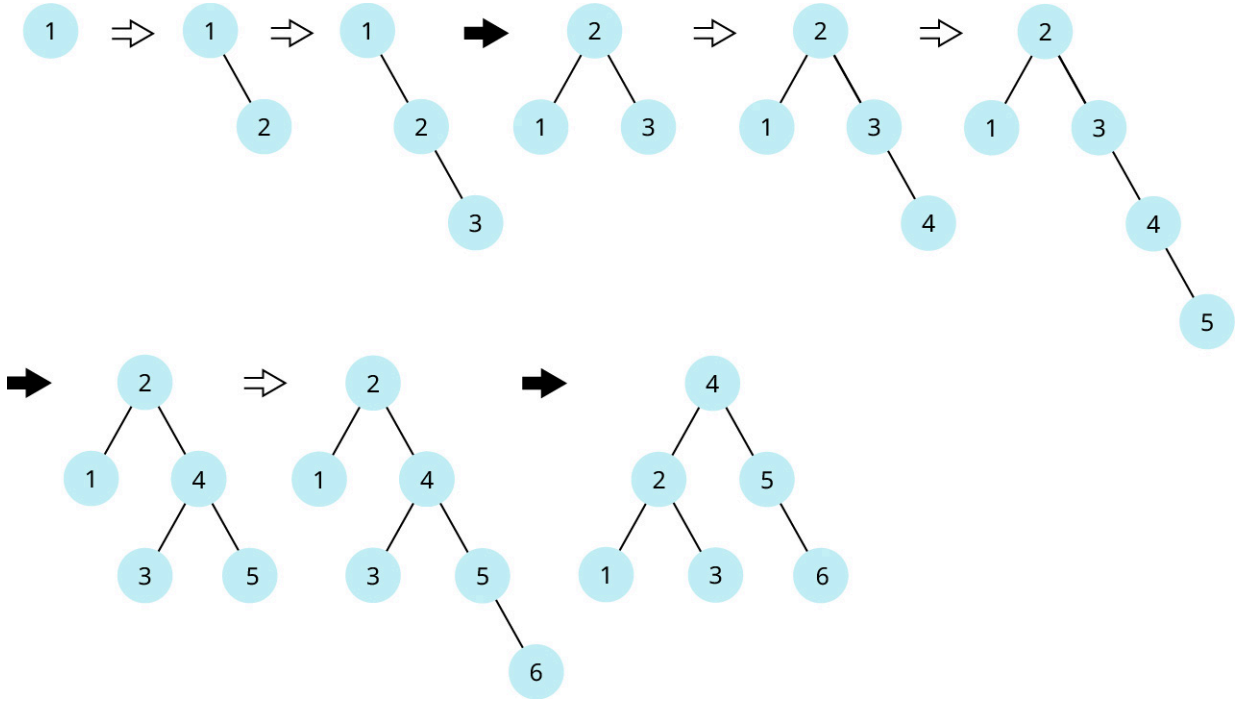
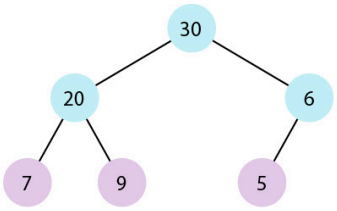


Figure 3.7 An AVL tree rotates nodes in a binary search tree to maintain balance. This sequence of steps illustrates the insertion of numbers 1, 2, 3, 4, 5, 6 into an initially empty AVL tree. (The steps in which rotation occurs are represented by the solid black arrows.) (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Balanced binary search trees such as AVL trees represent just one approach for ensuring that the tree never enters a worst-case situation. There are many other balanced binary search tree data structures in addition to AVL trees. Balanced binary search trees can also be used to implement the priority queue abstract data type if the elements are ordered according to their priority value. But balanced search trees are not the only way to implement priority queues.

Binary Heaps

Priority queues focus on retrieving and removing the highest-priority elements first, adding an element to a priority queue also involves specifying an associated priority value that is used to determine which elements are served next. For example, patients in an emergency room might be served according to the severity of their health concerns rather than according to arrival time. A **binary heap** is a type of binary tree data structure that is also the most common implementation for the priority queue abstract data type (Figure 3.8). A binary heap is not a search tree, but rather a hybrid data structure between a binary tree and an array list. Data is stored as an array list in memory, but the binary heap helps visualize data in the same way that a binary tree does, which makes it easier to understand how data are stored and manipulated. Binary heaps organize elements according to the **heap property**, which requires that the priority value of each node in the heap is greater than or equal to the priority values of its children. The heap property suggests that the highest-priority element will always be the root node where it is efficient to access.



Index	0	1	2	3	4	5
Data	30	20	6	7	9	5

Figure 3.8 A binary heap is the most common implementation of the priority queue abstract data type. The priority value of each node in the binary heap is greater than or equal to the priority values of the children. Note that the value stored in the root node of the right subtree can be smaller than the value stored in any node in the left subtree, while not violating the heap property. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

CONCEPTS IN PRACTICE

Tracking Earthquakes

Earthquakes, hurricanes, tsunamis, and other natural disasters occur regularly, and often demand an international response. How do we track natural disasters and identify the most affected areas in order to coordinate relief and support efforts? In the United States, the U.S. Geological Survey (USGS) is responsible for reporting earthquakes using thousands of earthquake sensors. However, that still leaves many places without earthquake sensors. Outside the United States, sensor technology may be less robust or inaccessible.

Social network data can be used to enhance this information and more quickly alert governments about natural disasters in real-time. By monitoring public social network platforms for occurrences of short posts such as “earthquake?,” we can quickly localize earthquakes based on the user’s geolocation data. However, aggregating and understanding this data—often thousands of data points arriving in minutes—requires efficient data structures and algorithms. We can use a binary heap that implements the priority queue abstract data type for an earthquake-tracking program. For each “earthquake?” post received for a given geolocation, we can increase the priority of the earthquake locations, which helps identify the likely-

earthquake location that is closest to the user's real location. At any time, we can efficiently retrieve the highest-priority element from the priority queue. By choosing to use a binary heap rather than a linear data structure for implementing the priority queue, we can ensure that the earthquake-tracking program is able to keep up with the thousands of posts made every minute during an earthquake.

Graph Data Structures

Both binary search trees and binary heap data structures represent more efficient ways to implement sets, maps, and priority queues by organizing data according to their intrinsic properties. In both cases, the properties of data enable efficient addition, retrieval, and removal of elements.

Graphs are a different kind of abstract data type. Rather than focusing on addition, retrieval, and removal, graphs focus on explicitly modeling the relationships between elements. Graphs afford access not only to elements, but also to relationships between elements.

- A **vertex** represents an element in a graph or a special type of it, such as a tree.
- An **edge** is the relationship between vertices or nodes. Optionally, edges can have associated weights. In a graph abstract data type, the relationships between two vertices connected by an edge are considered **adjacent**.

LINK TO LEARNING

Visualgo is a website that provides animations to help users learn more about algorithms and data structures. They have exercises to help understand several concepts presented in this chapter. You can access animations on various data structures and algorithms (https://openstax.org/r/76Visualgo) such as a linked list, a binary search tree, and graph structures.

In computer networks such as the Internet, graphs can represent individual network routers as nodes with data packets flowing between directly connected routers along edges. Even though not every router is directly connected to every other router, the router at the current node can analyze an incoming data packet to determine which edge it should travel through next. By repeating this process, a data packet can travel from a router on the Internet to another router on the Internet even though the two routers are not directly adjacent to each other.

Graphs are unique in that they can directly represent a wide variety of real-world problems, such as the following:

- A social network, where each vertex is a person, and each edge is a friendship.
- The Web, where each vertex is a webpage, and each edge is a link.
- A campus map, where each vertex is a building, and each edge is a footpath.
- A course prerequisite diagram, where each vertex is a course, and each edge is a prerequisite.

Unlike the list, set, map, and priority queue abstract data types, which have relatively standardized functionality focusing on the addition, retrieval, and removal of elements, the graph abstract data type is much less standardized. Typically, graph algorithm designers will create their own graph data type to represent a problem. The corresponding graph problem can then be represented using or adapting a standard graph algorithm. Unlike programming with other abstract data types, much of the hard work of solving a problem with a graph occurs when programmers decide what the vertices and edges represent, and which graph algorithm would be appropriate to solve the problem. They also must consider the consequences of how they represent the problem.

GLOBAL ISSUES IN TECHNOLOGY

Contact Tracing

Epidemiology is the study of how infectious diseases spread across the world. Within epidemiology, contact tracing attempts to identify confirmed cases of disease and limit its spread by tracing contacted people and isolating them from further spreading the disease.

Graph data structures can help epidemiologists manage the data and people involved in contact tracing. Imagine a graph where each vertex in a tracing graph represents a person, and each edge between two people represents a possible contact. When a person receives a positive test result for contracting the disease, healthcare professionals can identify all the people that they've been in contact with by tracing through the graph.

In addition to improving public health through contact tracing, our imaginary graph can also represent a history of the spread of the disease for epidemiologists to understand how the disease moves through communities. For example, if each vertex includes identity characteristic data such as age, race, or gender, epidemiologists can study which groups of people are most affected by the disease. This can then inform the distribution of vaccines to assist the most impacted groups first.

Complex Data

Now that we have seen several data structure implementations for abstract data, let us consider how these data structures are used in practice. Recall that we compared calculators whose algorithms operated on numbers with the idea of a computer whose algorithms operated on complex data. Data structures can be used to represent complex data by modeling hierarchy and relationships.

We might represent an online retail store as a map associating each item with details such as an image, a brief description, price, and the number of items in stock. This map makes some online storefront features easier to implement than others. Given an item, this map makes it easy to retrieve the details associated with that item. On the other hand, it is not so easy to sort items by price, sort items by popularity, or search for an item by keywords in its description. All these features could be implemented with additional data structures. We can combine multiple data structures together to implement these features. In computer science, we use database systems (see Chapter 8 Data Management) that work behind the scenes in many applications to manage these data structures and facilitate long-term storage and access to large amounts of data.

Just as calculators have algorithms for calculating numbers, computers have algorithms for computing complex data. Data structures represent these complex data, and algorithms act on these data structures.

3.2 Algorithm Design and Discovery

Learning Objectives

By the end of this section, you will be able to:

- Understand the approach to solving algorithmic problems
- Explain how algorithm design patterns are used to solve new problems
- Describe how algorithms are analyzed

Our introduction to data structures focused primarily on representing complex data. But computer scientists are also interested in designing algorithms for solving a wider variety of problems beyond storing and retrieving data. For example, they may want to plan a route between a start location and an end location on a map. Although every real-world problem is unique, computer scientists can use a general set of principles to design solutions without needing to develop new algorithms from scratch. Just like how many data structures

can represent the same abstract data type, many different solutions exist to solve the same problem.

Algorithmic Problem Solving

An algorithm is a sequence of precise instructions that takes any input and computes the corresponding output, while **algorithmic problem-solving** refers to a particular set of approaches and methods for designing algorithms that draws on computing's historical connections to the study of mathematical problem solving. Early computer scientists were influenced by mathematical formalism and mathematical problem solving. George Pólya's 1945 book, *How to Solve It*, outlines a process for solving problems that begins with a formal understanding of the problem and ends with a solution to the problem. As an algorithm's input size is always finite, finding a solution to an algorithmic problem can always be accomplished by exhaustive search. Therefore, the goal of algorithmic problem-solving, as opposed to mathematical problem solving, is to find an "efficient" solution, either in terms of execution time (e.g., number of computer instructions) or space used (e.g., computer memory size). Consequently, the study of algorithmic problem-solving emphasizes the formal **problem** or task, with specific input data and output data corresponding to each input. There are many other ways to solve problems with computers, but this mathematical approach remains the dominant approach in the field. Here are a few well-known problems in computer science that we will explore later in this chapter.

A **data structure problem** is a computational problem involving the storage and retrieval of elements for implementing abstract data types such as lists, sets, maps, and priority queues. These include:

- **searching**, or the problem of retrieving a target element from a collection of elements
- **sorting**, or the problem of rearranging elements into a logical order
- **hashing**, or the problem of assigning a meaningful integer index for each object

A **graph problem** is a computational problem involving graphs that represent relationships between data. These include:

- **traversal**, or the problem of exploring all the vertices in a graph
- **minimum spanning tree** is the problem of finding a lowest-cost way to connect all the vertices to each other
- **shortest path** is the problem of finding the lowest-cost way to get from one vertex to another

A **string problem** is a computational problem involving text or information represented as a sequence of characters. Examples include:

- **matching**, or the problem of searching for a text pattern within a document
- **compression**, or the problem of representing information using less data storage
- **cryptography**, or the problem of masking or obfuscating text to make it unintelligible

Modeling

Computer scientists focus on defining a **problem model**, often simply called a model, which is a simplified, abstract representation of more complex real-world problems. They apply the algorithmic problem-solving process mentioned previously to design algorithms when defining models. Algorithms model phenomena in the same way that data structures implement abstract data types such as lists, sets, maps, priority queues, and graphs. But unlike abstract data types, models are not necessarily purely abstract or mathematical concepts. Models are often linked to humans and social phenomena. A medical system might want to decide which drugs to administer to which patients, so the algorithm designer might decide to *model* patients as a complex data type consisting of age, sex, weight, or other physical characteristics. Because models represent *abstractions*, or simplifications of real phenomena, a model must emphasize some details over others. In the case of the medical system, the algorithm designer emphasized physical characteristics of people that were deemed important and chose to ignore other characteristics, such as political views, which were deemed less important for the model.

If an algorithm is a solution to a problem, then the model is the frame through which the algorithm designer

defines the rules and potential outcomes. Without models, algorithm designers would struggle with the infinite complexity and richness of the world. Imagine, for example, designing a medical system that models patients at the level of individual atoms. This model offers a detailed representation of each patient in the most physical or literal sense. But this model is impractical because we do not know how particular configurations and collections of atoms contribute to a person's overall health. Compared to this atomic-scale model, our former model consisting of age, sex, weight, and other physical characteristics is more practical for designing algorithms, but necessarily involves erasing our individual humanity to draw certain conclusions.

In order to design algorithms, we need to be able to focus on relevant information rather than detailed representations of the real world. Further, computer science requires a philosophical mind to aid in problem solving. According to Brian Cantwell Smith, philosopher and cognitive and computer scientist, "Though this is not the place for metaphysics, it would not be too much to say that every act of conceptualization, analysis, or categorization, does a certain amount of violence to its subject matter, in order to get at the underlying regularities that group things together."[1] Without performing this "violence," there would be too many details to wade through to create a useful algorithm.

The relationship between algorithms, the software they empower, and the social outcomes they produce is currently the center of contested social and political debate. For example, all media platforms (e.g., Netflix, Hulu, and others) use some level of targeted advertising based on user preferences in order to recommend specific movies or shows to their users. Users may not want their information to be used in this way, but there must be some degree of compromise to make these platforms attractive and useful to people.

On the one hand, the technical definition of an algorithm is that it represents complex processes as a sequence of precise instructions operating on data. This definition does not overtly suggest how algorithms encode social outcomes. On the other hand, computer programs are human-designed and socially engineered. Algorithm designers simplify complex real-world problems by removing details so that they can be modeled as computational problems. Because software encodes and automates human ideas with computers, software engineers wield immense power through their algorithms.

To further complicate the matter, software engineering is often a restrictive and formal discipline. Problem modeling is constrained by the **model of computation**, or the rules of the underlying computer that is ultimately responsible for executing the algorithm. Historically, computer science grew from its foundations in mathematics and formal logics, so algorithms were specialized to solve specific problems with a modest model of the underlying phenomena. This approach to algorithm design solves certain types of problems so long as they can be reasonably reduced to models that operate on a modest number of variables—however many variables the algorithm designer can keep in mind. In the case of the medical system, the algorithm designer identified certain characteristics as particularly useful for computing a result.

But there are many other problems that defy this approach, particularly tasks that involve subtle and often unconscious use of human sensory and cognitive faculties. An example of this is facial recognition. If asked to describe how we recognize a particular person's face, an algorithm designer would be challenged to identify specific variables or combinations of variables that correspond to only a single person. The formal logic required to define an algorithm is strict and absolute, whereas our understanding human faces is defined by many subtle factors that are difficult for anyone to express using formal logic.

INDUSTRY SPOTLIGHT

Machine Learning Algorithms

A machine learning algorithm addresses these kinds of problems by using an alternative model of

1 B. C. Smith, "The limits of correctness." *ACM SIGCAS Comput. Soc.*, vol. 14, 15, no. 1, 2, 3, 4, pp. 18-26, Jan. 1985. https://doi.org/10.1145/379486.379512.

computation, one that focuses on generalized algorithms designed to solve problems with a massive model of the underlying phenomena. Instead of attempting to identify a few key variables for facial recognition, for instance, machine learning algorithms can take as input a digital image represented as a rectangular grid of colored pixels. While each pixel in the image offers very little information about the person in mind, the facial features unique to each human arise from the arrangements and patterns of pixels that result from seeing many images of the same person.

Think about the way your Apple iPhone or Google Pixel phone may look at you when you try to access it and have facial recognition enabled. The algorithm is not going to try to match your face to a saved picture of you because it would not work all the time if you do not look exactly like you did in the picture. Rather, it uses machine learning to extract patterns out of a person's face and match them, making it possible to recognize people all the time even if they are wearing glasses but was not wearing them when they set up facial recognition on their phone. This method does seem to mimic the way humans recognize people, even if they have not seen them for decades.

Machine learning algorithms offer a more robust approach to modeling these kinds of problems that are not easily expressed in formal logic. But in this chapter, we focus on the earlier, classical perspective on algorithmic problem-solving with the end goal of designing specialized algorithms to solve problems with modest models of the underlying phenomena.

Search Algorithms

In computer science, searching is the problem of retrieving a target element from a collection that contains many elements. There are many ways to understand search algorithms; depending on the exact context of the problem and the input data, the expected output might differ. For example, suppose we want to find the target term in a dictionary that contains thousands or millions of terms and their associated definitions. If we represent this dictionary as a list, the search algorithm would return the index of the term in the dictionary. If we represent this dictionary as a set, the search algorithm would return whether the target is in the dictionary. If we represent this dictionary as a map, the search algorithm would return the definition associated with the term. The dictionary data structure has implications on the output of the search algorithm. Algorithmic problem-solving tends to be iterative because we might sometime later realize that our data structures need to change. Changing the data structures, in turn, often also requires changing the algorithm design.

Despite these differences in output, the underlying canonical searching algorithm can still follow the same general procedure. The two most well-known canonical searching algorithms are known as sequential search and binary search, which are conducted on linear data structures, such as array lists.

- Sequential search (Figure 3.9). Open the dictionary to the first term. If that term happens to be the target, then great—we have found the target. If not, then repeat the process by reading the next term in the dictionary until we have checked all the terms in the dictionary.
- Binary search (Figure 3.10). Open the dictionary to a term in the middle of the dictionary. If that term happens to be the target, then great—we have found the target. If not, then determine whether the target comes before or after the term we just checked. If it comes before, then repeat the process except on the first half of the dictionary. Otherwise, repeat the process on the second half of the dictionary. Each time, we can ignore half of the remaining terms based on the place where we would *expect* to find the target in the dictionary.

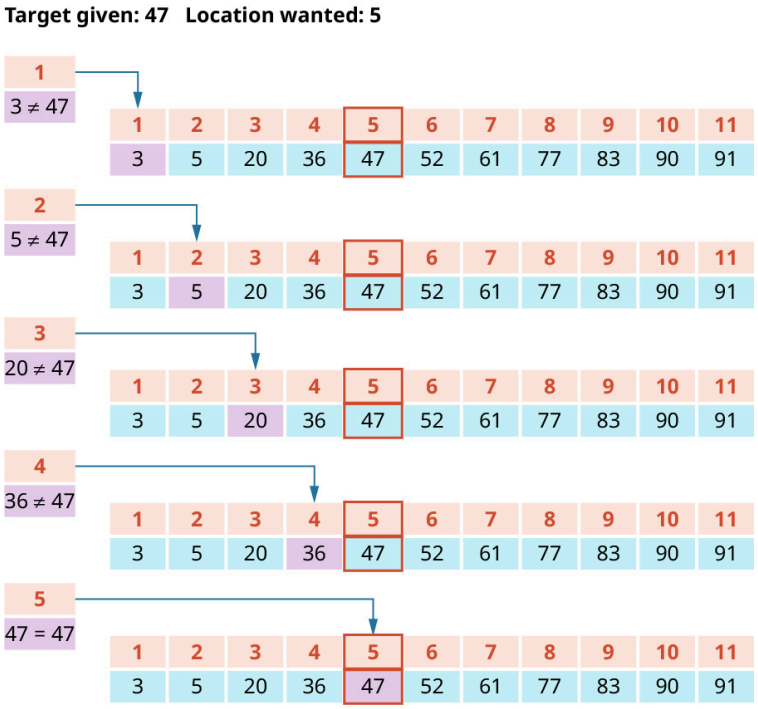


Figure 3.9 A sequential search can find the number 47 in an array by checking each number in order. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

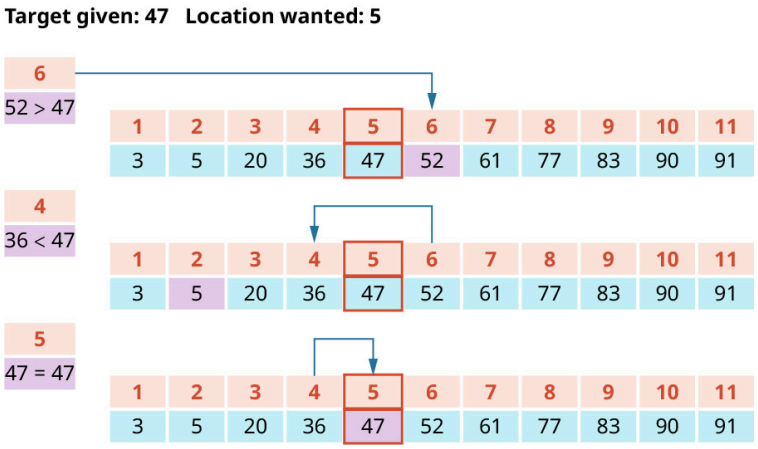


Figure 3.10 A binary search can find the number 47 in an array by determining whether the desired number comes before or after a chosen number. It eliminates half of existing data points and then searches in the remaining half, repeating the pattern, until the number is found. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Algorithm Design Patterns

This case study of canonical search algorithms demonstrates some ideas about algorithmic problem-solving, such as how algorithm design involves iterative improvement (from sequential search to binary search). But this case study does not demonstrate how algorithms are designed in practice. Algorithm designers are occasionally inspired by real-world analogies and metaphors, such as relying on sorted order to divide a dictionary into two equal halves. More often, they depend on knowledge of an existing **algorithm design pattern**, or a solution to a well-known computing problem, such as sorting and searching. Rather than develop wholly new ideas each time they face a new problem, algorithm designers instead apply one or more algorithm design patterns to solve new problems. By focusing on algorithm design patterns, programmers can solve a wide variety of problems without having to invent a new algorithm every time.

For example, suppose we want to design an autocomplete feature, which helps users as they type text into a program by offering word completions for a given prefix query. Algorithm designers begin by modeling the

problem in terms of more familiar data types.

- The input is a prefix query, such as a string of letters that might represent the start of a word (e.g., “Sea”).
- The output is a list of matching terms (completion suggestions) for the prefix query.

In addition to the input and output data, we assume that there is a list of potential terms that the algorithm will use to select the matching terms.

There are a few different ways we could go about solving this problem. One approach is to apply the sequential search design pattern to the list of possible words (Figure 3.11). For each term in the list, we add it to the result if it matches the prefix query. Another approach is to first *sort* the list of potential terms and then apply two binary searches: the first binary search to find the first matching term and the second binary search to find the last matching term. The output list is all the terms between the first match and the last match.

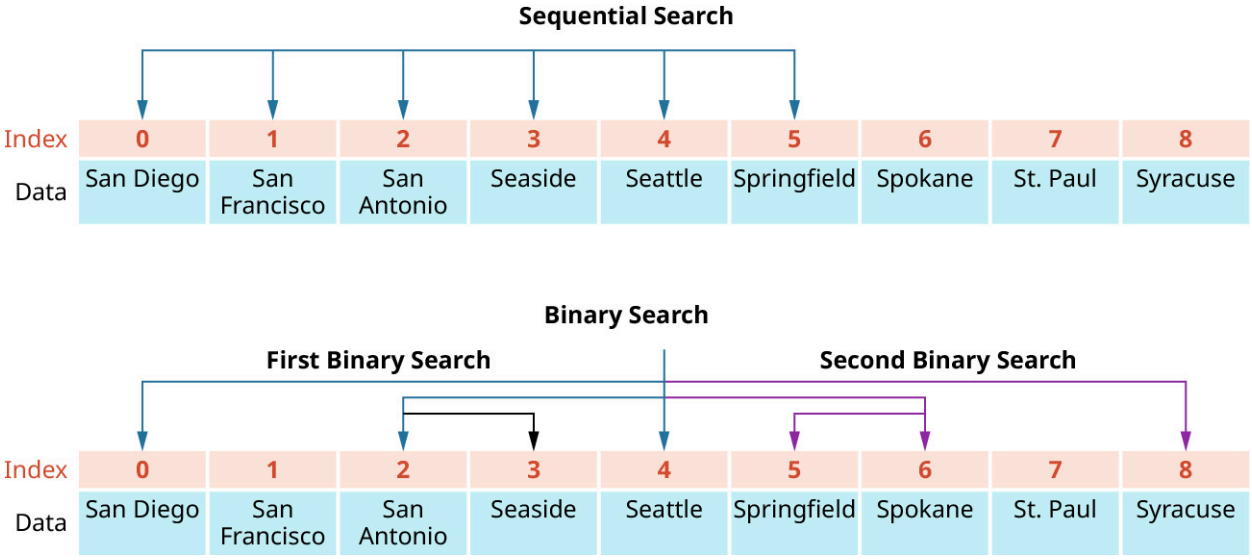


Figure 3.11 Sequential search needs to check every term to see if it matches the prefix “Sea,” whereas two binary searches can be used to find the start and end points of the matching terms in the list. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

TECHNOLOGY IN EVERYDAY LIFE

Online Autocomplete Algorithms

In online mapping, autocomplete might take the prefix *Sea* and automatically suggest the city *Seattle*. We know that search algorithms solve this problem by maintaining a sorted collection of place suggestions. But many online mapping applications allow users to change maps as the places change in the real world. How can we design the autocomplete feature to support real-world user changes? To apply the binary search algorithm, all place names must be stored in a sorted array-based list. So, every change will also need to maintain the sorted order.

If we instead add all the place names to a binary search tree, what are the steps for the autocomplete algorithm? How does this choice affect additions, changes, and removals?

Algorithm Analysis

Rather than rely on a direct analogy to our human experiences, these two algorithms for the autocomplete feature compose one or more algorithm design patterns to solve the problem. How do we know which approach might be more appropriate to use? One type of analysis is known as **algorithm analysis**, which studies the outputs produced by an algorithm as well as how the algorithm produces those outputs. Those

outputs are then evaluated for **correctness**, which considers whether the outputs produced by an algorithm match the expected or desired results across the range of possible inputs. An algorithm is considered correct only if its computed outputs are consistent with all the expected outputs; otherwise, the algorithm is considered incorrect.

Although this definition might sound simple, verifying an algorithm for correctness is often quite difficult in practice, because algorithms are designed to generalize and automate complex processes. The most direct way to verify correctness is to check that the algorithm computes the correct output for *every* possible input. This is not only computationally difficult, but even potentially impossible to achieve, since some algorithms can accept an infinite range of inputs.

Verifying the correctness of an algorithm is difficult not only due to generality, but also due to ambiguity. Earlier, we saw how canonical searching algorithms may have different outputs according to the input collection type. What happens if the target term contains special characters or misspellings? Should the algorithm attempt to find the closest match? Some ambiguities can be resolved by being explicit about the expected output, but there are also cases where ambiguity simply cannot be resolved in a satisfactory way. If we decide to find the closest match to the target term, how does the algorithm handle cultural differences in interpretation? If humans do not agree on the expected output, but the algorithm *must* compute some output, what output does it then compute? Or, if we do not want our algorithms to compute a certain output, how does it recognize those situations?

Correctness primarily considers consistency between the algorithm and the model, rather than the algorithm and the real world. Our autocomplete model from earlier returned all word completions that matched the incomplete string of letters. But in practice, this output would likely be unusable: a user typing "a" would see a list of all words starting with the letter "a." Since our model did not specify how to order the results, the user might get frustrated by the irrelevancy of many of the word completions. Suppose we remedy this issue by defining a relevancy metric: every time a user completes typing a word, increase that word's relevancy for future autocompletion requests. But, as Safiya Noble showed in *Algorithms of Oppression*, determining relevance in this universalizing way can have undesirable social impacts. Perhaps due to relevancy metrics determined by popular vote, at one point, Google search autosuggestions included ideas like:

- Women cannot: drive, be bishops, be trusted, speak in church
- Women should not: have rights, vote, work, box
- Women should: stay at home, be in the kitchen
- Women need to: be put in their places, know their place, be controlled, be disciplined[2]

Noble's critique extends further to consider the intersection of social identities such as race and gender as they relate to the outputs of algorithms that support our daily life.

GLOBAL ISSUES IN TECHNOLOGY

Searching for Identity

In *Algorithms of Oppression*, Safiya Noble describes how search engines can amplify sexist, racist, and misogynistic ideas. While searching for "Black girls," "Latina girls," and "Asian girls" circa 2013, Safiya was startled by how many of the top search results and advertisements that appeared on the first page of Google Search led to pornographic results when her input query did not at all suggest anything pornographic. In contrast, searching for "White girls" did not include pornographic results. As algorithms become more commonplace in our daily lives, they also become a more potent force for determining certain social futures. Algorithms are immensely powerful in their ability to affect not only how we act, but also what we see, what we hear, what we believe about the world, and even what we believe about

2 S.U. Noble, *Algorithms of Oppression: How Search Engines Reinforce Racism*. NYU Press, 2018.

ourselves.

As the amount of input data increases, computers often need more time or storage to execute algorithms. This condition is known as **complexity**, which is based on the degree computational resources that an algorithm consumes during its execution in relation to the size of the input. More computational time also often means consuming more energy. Given the exponential explosion in demand for data and computation, designing efficient algorithms is not only of practical value but also existential value as computing contributes directly to global warming and resultant climate crises.

THINK IT THROUGH

Content Moderation

Online social media platforms facilitate social relationships between users by allowing users to create and share content with each other. This user-generated content requires moderation, or methods for managing content shared between the platform users. Some researchers argue that content moderation defines a social network platform; in other words, content moderation policies determine exactly what content can be shared on the platform, which in turn defines the value of information. As social media platforms become increasingly prevalent, the information on these platforms plays an important role in influencing their users.

One approach for content moderation is to recruit human moderators to review toxic content, or content that is profane, abusive, or otherwise likely to make someone disengage. An algorithm could be developed to determine the toxicity of a piece of content, and the most toxic content could be added to a priority queue for priority moderation.

What are the consequences of this approach? How does the definition of toxicity prioritize (or de-prioritize) certain content? Who does it benefit? Consider the positionality of the users that interact with the platform:

- marginalized users of the platform, who may be further marginalized by this definition of toxicity.
- content moderators, who are each reviewing several hundred pieces of the most toxic content for hours every day.
- legal teams, who want to mitigate government regulations and legislation that are not aligned with corporate interests.
- social media hackers, or users who want to leverage the way certain content is prioritized in order to deliberately shape public opinion.

3.3 Formal Properties of Algorithms

Learning Objectives

By the end of this section, you will be able to:

- Understand time and space complexity
- Compare and contrast asymptotic analysis with experimental analysis
- Explain the Big O notation for orders of growth

Beyond analyzing an algorithm by examining its outputs, computer scientists are also interested in examining its efficiency by performing an algorithmic **runtime analysis**, a study of how much time it takes to run an algorithm.

If you have access to a runnable program, perhaps the most practical way to perform a runtime analysis is to

time exactly how long it takes to run the program with a stopwatch. This approach, known as **experimental analysis**, evaluates an algorithm's runtime by recording how long it takes to run a program implementation of it. Experimental analysis is particularly effective for identifying performance bugs or code that consumes unusually large amounts of computation time or system resources, even though it produces the correct output. In e-commerce, for example, performance bugs that result in slow website responsiveness can lead to millions of dollars in lost revenue. In the worst-case scenario, performance bugs can even bring down entire websites and networks when systems are overloaded and cannot handle incoming requests. As the Internet becomes more heavily used for information and services, performance bugs can have direct impacts on health and safety if the computer infrastructure cannot keep up with demand.

While experimental analysis is useful for improving the efficiency of a program, it is hard to use if we do not already have a working program. Programming large systems can be expensive and time-consuming, so many organizations want to compare multiple algorithm designs and approaches to identify the most suitable design *before* implementing the system. Even with sample programs to represent each algorithm design, we can get different results depending on the processing power, amount of memory available, and other features of the computer that is running the program.

Designing more efficient algorithms is not just about solving problems more quickly, but about building a more sustainable future. In this section, we will take a closer look at how to formally describe the efficiency of an algorithm without directly executing a working program.

CONCEPTS IN PRACTICE

Performance Profiling

Modern computer systems are complicated. Algorithms are just one component in a much larger ecosystem that involves communication between many other subsystems, other computers in a data center, and other systems on the Internet. Algorithmic runtime analysis focuses on the properties of the algorithm rather than all the different ways the algorithm interacts with the rest of the world. But once an algorithm is implemented as a computer program, these interactions with the computing ecosystem play an important role in determining program performance.

A profiler is a tool that measures the performance (runtime and memory usage) of a program. Profilers are commonly used to diagnose real-world performance issues by producing graphs of how computational resources are used in a program. A common graph is a flame graph (Figure 3.12) that visualizes resource utilization by each part of a program to help identify the most resource-intensive parts of a program. Saving even a few percentage points of resources can lead to significantly reduced time, money, and energy expenditure. As the global demand for computation continues to increase, performance engineers who know how to leverage profilers to analyze systems and implement resource-saving changes will be key to a green energy future.

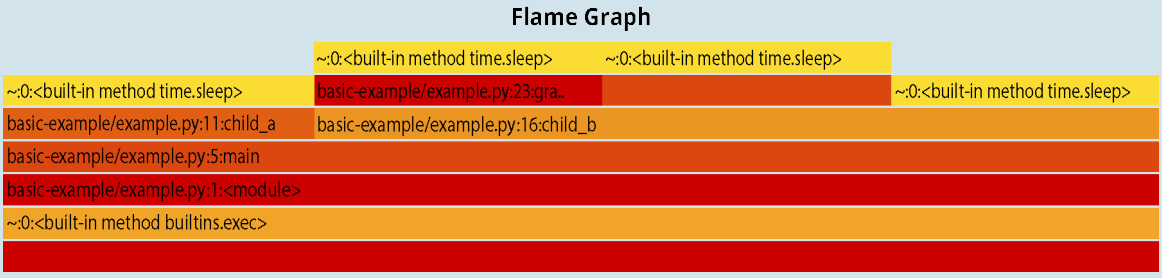


Figure 3.12 A flame graph shows which parts of a program require the most resources. The x-axis shows relative duration and the width indicates the percentage of total duration spent in a function. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Time and Space Complexity

One way to measure the efficiency of an algorithm is through **time complexity**, a formal measure of how much time an algorithm requires during execution as it relates to the size of the problem. In addition to time complexity, computer scientists are also interested in **space complexity**, the formal measure of how much memory an algorithm requires during its execution as it relates to the size of the problem.

Both time and space complexity are formal measures of the efficiency of an algorithm as it relates to the size of the problem, particularly when we are working with large amounts of complex data. For example, gravity, the universal phenomenon by which things attract and move toward each other, can be modeled as forces that act on every pair of objects in the universe. Simulating a subset of the universe that contains only 100 astronomical bodies will take a lot less time than a much larger universe with billions, trillions, or even more bodies, all of which gravitate toward each other. The size of the problem plays a large role in determining how much time an algorithm requires to execute. We will often express the size of the problem as a positive integer number corresponding to the size of the dataset such as the number of astronomical bodies in our simulation.

The goal of time and space complexity analysis is to produce a simple and easy-to-compare characterization of the efficiency of an algorithm as it relates to the size of the problem. Consider the following description of an algorithm that searches for a target word in a list. Start from the very beginning of the list and check if the first word is the target word. If it is, we have found the word. If it is not, then continue to the next word in the list and repeat the process.

The first task is to identify a metric for representing the size of the problem. Typically, time complexity analysis assumes **asymptotic analysis**, focusing on evaluating the time that an algorithm takes to produce a result as the size of the input increases. There are two inputs to this algorithm: (1) the list of words, and (2) the target word. The average length of an English word is about five characters, so the size of the problem is primarily determined by the number of words in the list rather than the length of any word. (This assumption might not be right if our dataset was instead a DNA sequence consisting of millions of nucleotides—the time it takes to compare a pair of long DNA sequences might be more important than the number of DNA sequences being compared.) Identifying the size of the problem is an important first task because it determines the other factors we can consider in the following tasks.

The next task is to model the number of steps needed to execute the algorithm while considering its potential behavior on all possible inputs. A **step** represents a basic operation in the computer, such as looking up a single value, adding two values, or comparing two values. How does the runtime change as the size of the problem increases? We can see that the “repeat” part of our description is affected by the number of words in the list; more words can potentially lead to more repetitions.

In this case we are choosing a **cost model**, which is a characterization of runtime in terms of more abstract operations, such as the number of repetitions. Rather than count single steps, we instead count repetitions. Each repetition can involve several lookups and comparisons. By choosing each repetition as the cost model, we declare that the few steps needed to look up and compare elements can be effectively treated as a single operation to simplify our analysis.

However, this analysis is not quite complete. We might find the target word early in the list even if the list is very large. Although we defined the size of the problem as the number of words in the list, the size of the problem does not account for the exact words and word ordering in the list. Computer scientists say that this algorithm has a best-case situation when the word can be found at the beginning of the list, and a worst-case situation when the word can only be found at the end of the list (or, perhaps, not even in the list at all). One way to account for the variation in runtime is via **case analysis**, which is based on factors other than the size of the problem.

Finally, we can formalize our description using either precise English or a special mathematical notation called **Big O notation**, which is the most common type of asymptotic notation in computer science used to measure

worst-case complexity. In precise English, we might say that the time complexity for this sequential search algorithm has two cases (Figure 3.13):

- In the best-case situation (when the target word is at the start of the list), sequential search takes just one repetition to find the word in the list.
- In the worst-case situation (when the target word is either at the end of the list or not in the list at all), sequential search takes *N* repetitions where *N* is the number of words in the list.

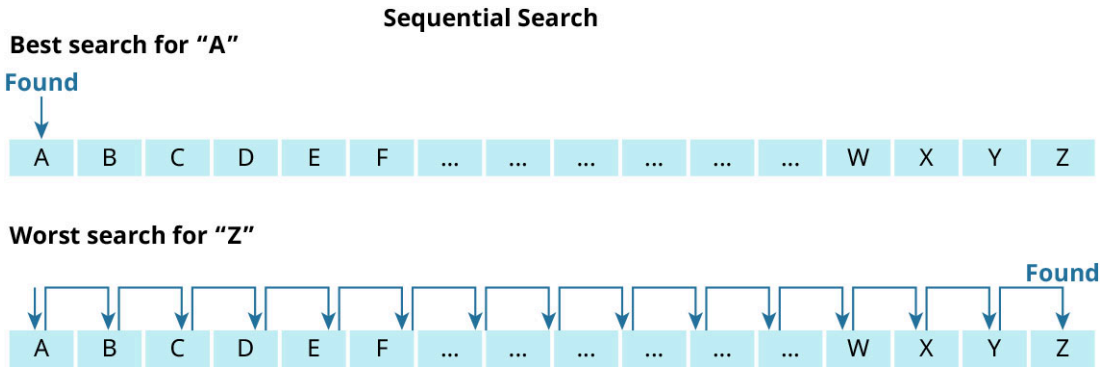


Figure 3.13 The best case for sequential search in a sorted list is to find the word at the top of the list, whereas the worst case is to find the word at the bottom of the list (or not in the list at all). (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

While this description captures all the ideas necessary to communicate the time complexity, computer scientists will typically enhance this description with mathematics to convey a geometric idea of the runtime. The **order of growth** is a geometric prediction of an algorithm's time or space complexity as a function of the size of the problem (Figure 3.14).

- In the best case, the sequential search has a **constant** order of growth that does not take more resources as the size of the problem increases.
- In the worst case, the sequential search has a **linear** order of growth where the resources required to run the algorithm increase at about the same rate as the size of the problem increases. This is with respect to *N*, the number of words in the list.

Constant and linear are two examples of orders of growth. An algorithm with a constant order of growth takes the same amount of time to execute even as the size of the problem grows larger and larger—no matter how large the dictionary is, it is possible to find the target word at the very beginning. In contrast, an algorithm with a linear time complexity will take more time to execute as the size of the problem grows larger, and we can predict that an increase in the size of the problem corresponds to roughly the same increase in the runtime.

This prediction is a useful outcome of time complexity analysis. It allows us to estimate the runtime of the sequential search algorithm on a problem of any size, before writing the program or obtaining a dictionary of words that large. Moreover, it helps us decide if we want to use this algorithm or explore other algorithm designs and approaches. We might compare this sequential search algorithm to the binary search algorithm and adjust our algorithm design accordingly.

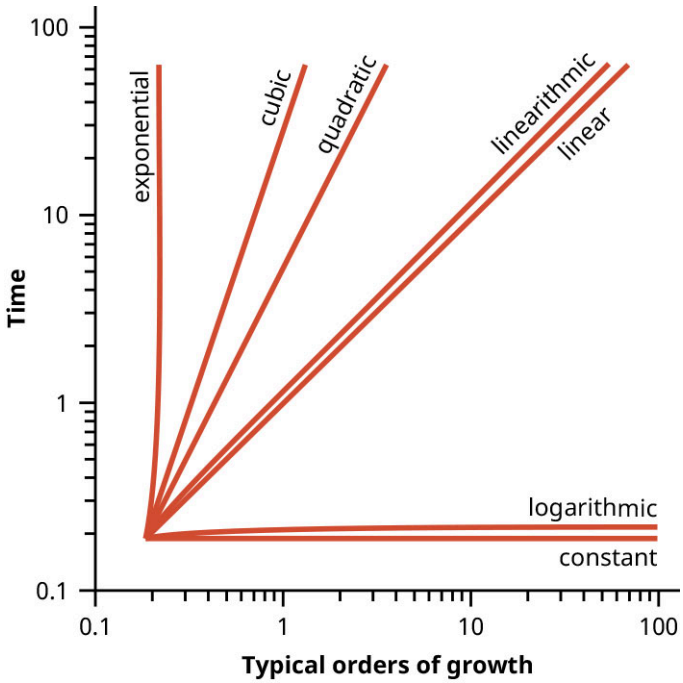


Figure 3.14 The order of growth of an algorithm is useful to estimate its runtime efficiency as the input size increases (e.g., constant, logarithmic, and other orders of growth) to help determine which algorithmic approach to take. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Big O Notation for Orders of Growth

In the 1970s, computer scientists applied **asymptotic notation**, a mathematical notation that formally defines the order of growth. We can use Big O notation to describe the time complexity of the sequential search algorithm. In general, we say a function f(N) is in the class of O(g(N)), denoted by f(N) = O(g(N)) or f(N) in O(g(N)), if when N tends to infinity, the ratio f(N)/g(N) is upper bounded by some constant. The function g(N) is usually some simple function that defines the order of growth such as g(N) = 1 (constant function), g(N) = N (linear function), g(N) = log N (logarithmic function), or other functions as follows:

- In the best case, the order of growth of sequential search is in O(1).
- In the worst case, the order of growth of sequential search is in O(N) with respect to N, the number of words in the list.

The constant order of growth is described in Big O notation as "O(1)" while the linear order of growth is described in Big O notation as "O(N) with respect to N, the size of the problem." Big O notation formalizes the concept of a prediction. Given the size of the problem, N, calculate how long it takes to run the algorithm on a problem of that size. For large lists, in order to double the worst-case runtime of sequential search, we would need to double the size of the list.

O(1) and O(N) are not the only orders of growth.

- O(1), or constant.
- O(log N), or logarithmic.
- O(N), or linear.
- O(N log N), or linearithmic.
- O(N^2), or quadratic.
- O(N^3), or cubic.
- O(2^N), O(3^N), . . . , or exponential.
- O(N!), or factorial.

The O(log N), or logarithmic, order of growth appears quite often in algorithm analysis. The **logarithm** of a

large number tells how many times it needs to be divided by a small number until it reaches 1. The **binary logarithm**, or $\log_2$, tells how many times a large number needs to be divided by 2 until it reaches 1. In the worst case, the time complexity of sequential search is in O(N) with respect to N, the number of words in the list, since each repetition of sequential search rules out one remaining element. How about binary search? In the worst case, the time complexity of binary search is in O(log N) with respect to N, the number of words in the list, since each repetition of binary search rules out half the remaining elements.

Another way to understand orders of growth is to consider how a change in the size of the problem results in a change to the resource usage. When we double the size of the input problem, algorithms in each order of growth respond differently (Figure 3.15).

- O(1) algorithms will not require any more resources.
- O(log N) algorithms will require 1 additional resource unit.
- O(N) algorithms will require 2 times the number of resources.
- O(N log N) algorithms will require a little more than 2 times the number of resources.
- O(N^2) algorithms will require 4 times the number of resources.
- O(N^3) algorithms will require 8 times the number of resources.
- O(2^N), O(3^N), . . . algorithms will require the squared or cubed number of resources.
- O(N!) algorithms will require even more.

This growth compounds, so quadrupling the size of the problem for an O(N^2) algorithm will require 16 times the number of resources. Algorithm design and discovery is often motivated by these massive differences between orders of growth. Note that this explanation of how each order of growth responds differently oversimplifies the problem. Rigorously speaking, a function $f(N)$ expressed in the big-O notation as being is in the class of O($g(N)$) can be much more complex than the simple function $g(N)$. For example, $f(N) = 4\log N + 100 \log(\log N)$ is in O(log N), but when N doubles, $f(2N)$ is definitely not just one unit more than the original function $f(N)$. A similar argument applies for all other functions other than the constant O(1) function.

Order of growth	Algorithm	Execution time for N = 10	Execution time for N = 1,000,000
Constant	O(1)	1 µsec	1 µsec
Logarithmic	O(log N)	3 µsec	18 µsec
Linear	O(N)	10 µsec	1 sec
Log-linear	O(N log N)	33 µsec	19.8 sec
Quadratic	O(N^2)	100 µsec	11.6 days
Cubic	O(N^3)	1 msec	31.7 years
Exponential	O(2^N)	10 msec	Billion of years
Factorial	O(N!)	3.6 sec	Practically infinite

Figure 3.15 This chart shows the time it would take for an algorithm with each of the given orders of growth to finish running on a problem of the given size, N. When an algorithm takes longer than 1025 years to compute, that means it takes longer than the current age of the universe. (data source: Geeks for Geeks, "Big O Notation Tutorial—A Guide to Big O Analysis," last updated March 29, 2024; attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

As the size of the problem increases, algorithmic complexity becomes a larger and larger factor. For problems dealing with just 1,000 elements, the time it would take to run an exponential-time algorithm on that problem exceeds the current age of the universe. In practice, across applications working with large amounts of data, O(N log N) is often considered the limit for real-world algorithms. Even then, O(N log N) algorithms cannot be run too frequently. For algorithms that need to run frequently on large amounts of data, algorithm designers target O(N), O(log N), or O(1).

TECHNOLOGY IN EVERYDAY LIFE

Arranging Invisible Icons in Quadratic Time

Have you ever been annoyed by computer slowness? For some users, opening the start menu can take 20 seconds because of an $O(N^2)$ algorithm, where N is the number of *desktop files*. The Microsoft Windows computer operating system allows users to organize files directly on top of their desktop wallpaper. A quadratic-time algorithm arranges these *desktop icons* in a grid layout so that they fill column-by-column starting from the left side of the screen. The algorithm is executed whenever the user opens the start menu or launches the file explorer.

Most users only keep a couple dozen desktop icons, so the $O(N^2)$ algorithm takes microseconds—practically unnoticeable. But for users with hundreds of desktop icons, the impact of each additional icon adds up. With 1,000 desktop files, launching the start menu takes 20 *seconds*. With 10,000 desktop icons, the runtime grows to 30 *minutes*!

To avoid the clutter of thousands of desktop icons, users can hide desktop icons. But this does not prevent the quadratic-time algorithm from running. Users with too many desktop icons beware: your computer slowness may be due to arranging invisible icons in quadratic time.[3]

3.4 Algorithmic Paradigms

Learning Objectives

By the end of this section, you will be able to:

- Apply the divide and conquer technique
- Explain the brute-force method
- Interpret and apply the greedy method
- Understand how to apply reductions to solve problems

Algorithm design patterns are solutions to well-known computing problems. In 3.5 Sample Algorithms by Problem, we will survey algorithm design patterns by problem. As it turns out, many of these algorithm design patterns share similarities in their approaches to solving problems. Here, we will introduce the **algorithmic paradigm**, which is the common concepts and ideas behind algorithm design patterns.

Divide and Conquer Algorithms

A **divide and conquer algorithm** is an algorithmic paradigm that breaks down a problem into smaller subproblems (divide), recursively solves each subproblem (conquer), and then combines the result of each subproblem to form the overall solution. The algorithm idea of recursion is fundamental to divide and conquer algorithms because it solves complex problems by dividing input data into smaller instances of the same problem known as subproblems. Such recursion calls terminate when the inputs become so small or so simple that other non-recursive procedures can provide the answers.

A **subproblem** is a smaller instance of a problem that can be solved independently, and each subproblem can be solved independently of other subproblems by reapplying the same recursive algorithm. To design recursive subproblems, algorithm designers often focus on identifying structural self-similarity in the input data. This process repeats until the input data is small enough to solve directly. Once all the subproblems have been solved, the recursive algorithm reassembles each of these independent solutions to compute the result for the original problem.

3 B. Dawson, "Arranging invisible icons in quadratic time," 2021. https://randomascii.wordpress.com/2021/02/16/arranging-invisible-icons-in-quadratic-time/

Earlier, we introduced binary search to find a target within a sorted list as an analogy for finding a term in a dictionary sorted alphabetically. Instead of starting from the beginning of the dictionary and checking each term, as in a sequential search, we could instead start from the middle and look left or right based on where we would expect to find the term in the dictionary. But binary search can also be understood as an example of a divide and conquer algorithm (Figure 3.16).

1. The problem of finding a target within the entire sorted list is broken down (divided) into the subproblem of finding a target within half of the list after comparing the middle element to the target. Half of the list can be ruled out based on this comparison, leaving binary search to find the target within the remaining half.
2. Binary search is repeated on the remaining half of the sorted list (conquer). This process continues recursively until the target is found in the sorted list (or reported as not in the list at all).
3. To solve the original problem of finding a target within the entire sorted list, the result of the subproblem must inform the overall solution. The original call to binary search reports the same result as its subproblem.

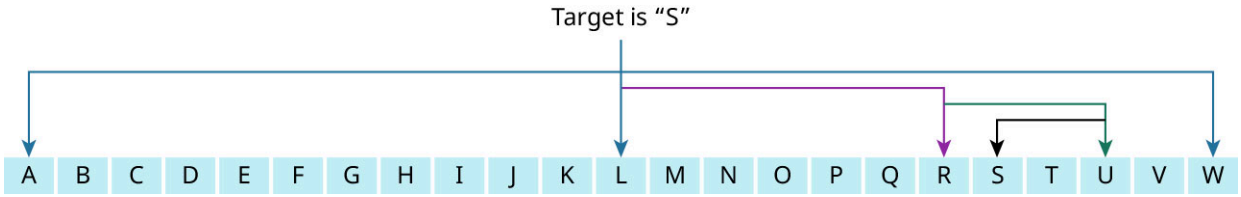


Figure 3.16 Binary search is a divide and conquer algorithm that repeatedly makes one recursive call on half of remaining elements. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Binary search makes a single recursive call on the remaining half of the list as part of the conquer step, but other divide and conquer algorithms make multiple recursive calls to solve their problems. We will also see algorithms that do a lot of work in the final step of combining results more than just reporting the same result as a subproblem.

Given a list of elements in an unknown order, a sorting algorithm should return a new list containing the same elements rearranged into a logical order, such as least to greatest. One canonical divide and conquer algorithm for comparison sorting is called **merge sort**. The problem of comparison sorting is grounded in the comparison operation. The comparison operation is like a traditional weighing scale that tells whether one element is heavier, lighter, or the same weight as another element, but provides no information about the exact weight or value of the element. Though this might seem like a serious restriction, comparison sorting is actually a very rich problem in computer science—it is perhaps the most deeply studied problem in computer science. Choosing comparison as the fundamental operation is also practical for complex data, where it might be hard (or even impossible) to assign an exact numeric ranking (Figure 3.17).

1. The problem of sorting the list is broken down (divided) into two subproblems: the subproblem of sorting the left half and the subproblem of sorting the right half.
2. Merge sort is repeated to sort each half (conquer). This process continues recursively until the sublists are one element long. To sort a one-element list, the algorithm does not need to do anything, since the elements in the list are already arranged in a logical order.
3. To solve the original problem of sorting the entire list, combine adjacent sorted sublists by merging them while maintaining sorted order. Merging each pair of adjacent sorted sublists repeats to form larger and larger sorted sublists until the entire list is fully sorted.

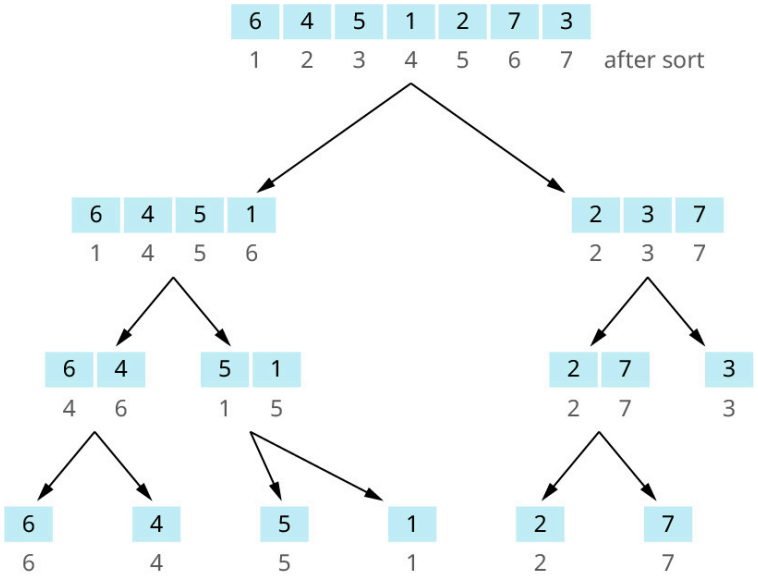


Figure 3.17 Merge sort is a divide and conquer algorithm that repeatedly makes two recursive calls on both halves of the sublist. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Brute-Force Algorithms

Solving a **combinatorial problem** involves identifying the best candidate solution out of a space of many potential solutions. Each solution to a combinatorial problem is represented as a complex data type. Many applications that involve simulating and comparing different options can be posed as combinatorial problems.

- Route planning in online mapping asks, "Out of all the possible routes from point A to point B, which route is the shortest?" (Shortest paths problem)
- Municipal broadband planning asks, "Out of all the possible ways to connect every real-world address to the Internet, which network of connections is the cheapest to build?" (Minimum spanning trees problem)
- The **interval scheduling problem** is a combinatorial problem involving a list of scheduled tasks with the goal of finding the largest non-overlapping set of tasks that can be completed.

Rather than searching for a single element, combinatorial problems focus on finding candidate solutions that might be represented as a list of navigation directions, network connections, or task schedules. And unlike sorting, where the output should be a sorted list, combinatorial problems often attempt to quantify or compare the relative quality of solutions in order to determine the best candidate solution out of a space of many potential solutions. Even if our route plan is not the "best" or shortest route, it can still be a valid solution to the problem.

A **brute-force algorithm** solves combinatorial problems by systematically enumerating all potential solutions in order to identify the best candidate solution. For example, a brute-force algorithm for generating valid credit card numbers might start by considering a credit card number consisting of all zeros, then all zeros except for a one in the last digit place, and so forth, to gradually explore all the possible values for each digit place.

Brute-force algorithms exist for every combinatorial problem, but they are not typically used because of runtime issues. To systematically enumerate all potential solutions, a brute-force algorithm must generate every possible combination of the input data. For example, if a credit card number has sixteen digits and each digit can have any value between zero and nine, then there are 10^{16} potential credit card numbers to enumerate. The **combinatorial explosion** is the exponential number of solutions to a combinatorial problem that makes brute-force algorithms unusable in practice. Despite continual improvements to how quickly computers can execute programs, exponential time brute-force algorithms are impractical except for very small problem input data.

INDUSTRY SPOTLIGHT

Protein Folding

Proteins are one of the fundamental building blocks of biological life. The 3-D shape of a protein defines what it does and how it works. Given the string of a protein's amino acids, a protein-folding problem asks us to compute the 3-D shape of the resulting protein.

Protein folding has been studied since the first images of their structures were created in 1960. In 1972, Christian Anfinsen won the Nobel Prize in Chemistry for his research into the "protein-folding problem," which involved algorithms to predict the structure of proteins. Because of the huge number of possible formations of proteins, computational studies and algorithms are better able to predict the structures. Given that a brute-force algorithm cannot solve this problem in a reasonable amount of time, computational biologists have developed algorithms that generate high-quality approximations or potential solutions that typically are not quite correct, but run in a more reasonable amount of time.

Modern protein-folding algorithms, such as Google's DeepMind AlphaFold machine-learning algorithm,[4] use machine learning to identify protein-folding patterns from millions of input amino acid sequences and corresponding output 3-D conformations. Rather than utilizing a simple rule for selecting the next element to include in the solution, these machine learning algorithms learn highly complicated rulesets from subtle patterns present in the data.

Improving our understanding of protein folding can lead to massive improvements only in medical health contexts such as drug and vaccine development, but also enable us to design biotechnologies such as enzymes for composting plastic waste, and even limit the impact of global warming by sequestering greenhouse gases from the atmosphere.[5] In 2024, the Nobel Prize Committee recognized the impact of this work by granting the Chemistry prize to Demis Hassabis and John M. Jumper for their work on DeepMind and Alphafold[6], as well as David Baker for using a similar tool, Rosetta, to create entirely new proteins.

Greedy Algorithms

A **greedy algorithm** solves combinatorial problems by repeatedly applying a simple rule to select the next element to include in the solution. Unlike brute-force algorithms that solve combinatorial problems by generating all potential solutions, greedy algorithms instead focus on generating just one solution. These algorithms are *greedy* because they select the next element to include based on the immediate benefit.

For example, a greedy interval scheduling algorithm might choose to work on the task that takes the least amount of time to complete; in other words, the cheapest way to mark one task as completed (Figure 3.18). If the tasks are scheduled in advance and we can only work on one task at a time, choosing the task that takes the least amount of time to complete might prevent us from completing multiple other (longer) tasks that just so happen to overlap in time. This greedy algorithm does not compute the right output—it finds a solution but not the optimal solution.

4 W. D. Haven, "DeepMind's protein-folding AI has solved a 50-year-old grand challenge of biology," 2020. https://www.technologyreview.com/2020/11/30/1012712/deepmind-protein-folding-ai-solved-biology-science-drugs-disease/
5 Google DeepMind, "AlphaFold: A solution to a 50-year-old grand challenge in biology," 2020." https://deepmind.com/blog/article/alphafold-a-solution-to-a-50-year-old-grand-challenge-in-biology
6 Google DeepMind, "AlphaFold: Demis Hassabis & John Jumper awarded Nobel Prize in Chemistry," 2024." https://deepmind.google/discover/blog/demis-hassabis-john-jumper-awarded-nobel-prize-in-chemistry/

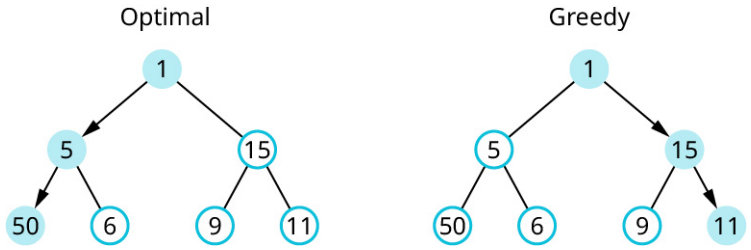


Figure 3.18 Greedy interval scheduling will not work if the simple rule repeatedly selects the shortest interval. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The majority of greedy algorithms do not always compute the best solution. But there are also certain scenarios in which cleverly crafted greedy algorithms guarantee finding optimal solutions. The problems they solve are formulated to ensure that the greedy algorithm never makes a mistake when repeatedly applying the simple rule to select the next element.

Consider the municipal broadband planning problem or, more formally, the minimum spanning tree problem, of finding a lowest-cost way to connect all the vertices in a connected graph to each other. If we want to minimize the sum of the selected edge weights, one idea is to repeatedly select the next edge (connections between vertices) with the lowest weight so long as it extends the reach of the network. Or, in the context of the municipal broadband planning problem, we want to ensure that the next-cheapest connection that we choose to build reaches someone who needs access to the Internet (Figure 3.19).

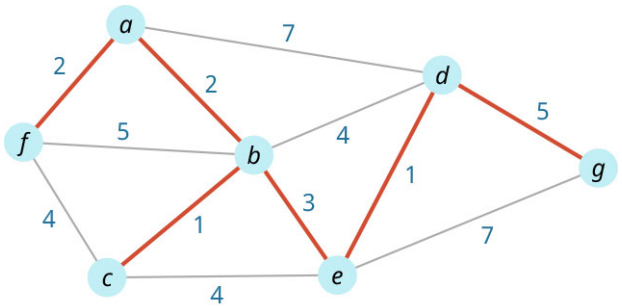


Figure 3.19 Municipal broadband planning can be represented as a minimum spanning trees graph problem where the weight of each edge represents the cost of building a connection between two vertices or places. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

GLOBAL ISSUES IN TECHNOLOGY

Municipal Broadband as a Public Utility

The municipal broadband planning problem is just one component of the larger public policy issue of Internet access. As the Internet and the use of digital platforms becomes the standard mode for communication, many people are viewing municipal broadband as a fundamental public utility and civil right. "Municipal broadband can solve access and affordability problems in areas where private ISPs [Internet service providers] have not upgraded networks to modern speeds, fail to provide service to all residents, and/or charge outrageous rates."[7]

While a minimum spanning tree algorithm can solve the municipal broadband planning problem, the challenges of deploying municipal broadband for everyone is more political rather than algorithmic. But other algorithms can also directly contribute to the way in which society understands the problem. For example, we might use algorithms to better visualize and understand who currently has access to affordable and reliable high-speed Internet. We can design algorithms to ensure equitable distribution,

7 J. Broken, "Victory for municipal broadband as Wash. state lawmakers end restrictions," 2021. Ars Technica, https://arstechnica.com/tech-policy/2021/04/victory-for-municipal-broadband-as-wash-state-lawmakers-end-restrictions

deployment, and integration of new technologies so that marginalized communities can realize the positive economic benefits first. Or we can reconfigure the minimum spanning trees problem model to take specifically account for expanding network access in an equitable fashion.

Computer scientists have designed algorithms that repeatedly apply this simple rule to find an optimal minimum spanning tree:

- **Kruskal's algorithm**, a greedy algorithm which sorts the list of edges in the graph by weight.
- **Prim's algorithm**, a greedy algorithm that maintains a priority queue of vertices in the graph ordered by connecting edge weight.

For most problems, greedy algorithms will not produce the best solution. Instead, algorithm designers typically turn to another algorithmic paradigm called dynamic programming. Still, greedy algorithms provide a useful baseline starting point for understanding problems and designing baseline algorithms for generating potential solutions.

Reduction Algorithms

Algorithm designers spend much of their time modeling problems by selecting and adapting relevant data structures and algorithms to represent the problem and a solution in a computer program. This process of modeling often involves modifying an algorithm design pattern so that it can be applied to the problem. But there is also a different approach to algorithm design that solves problems by changing the input data and output data to fit a preexisting problem. Rather than solve the problem directly, a **reduction algorithm** solves problems by transforming them into other problems (Figure 3.20).

1. Preprocess: Transform the input data so that it is acceptable to an algorithm meant for the other problem.
2. Apply the algorithm meant for the other problem on the preprocessed data.
3. Postprocess: Transform the output of the algorithm meant for the other problem so that it matches the expected output for the original problem.

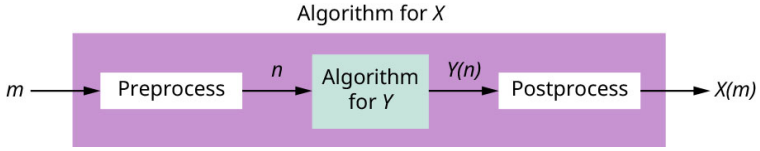


Figure 3.20 A reduction algorithm preprocesses the input data, passes it to another algorithm, and then postprocesses the algorithm's output to solve the original problem. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Consider a slightly different version of the municipal broadband planning problem, where instead of only considering connections (edges), we expand the problem to consider the possibility of installing broadband nodes directly to each address without relying on potentially expensive neighboring connections. That is, all vertices installed with broadband nodes are inter-connected with each other through another broadband network. If we were to run an algorithm for solving the minimum spanning tree problem on this graph directly, then our result would never consider installing broadband nodes directly, since minimum spanning tree algorithms do not consider vertex weights (Figure 3.21).

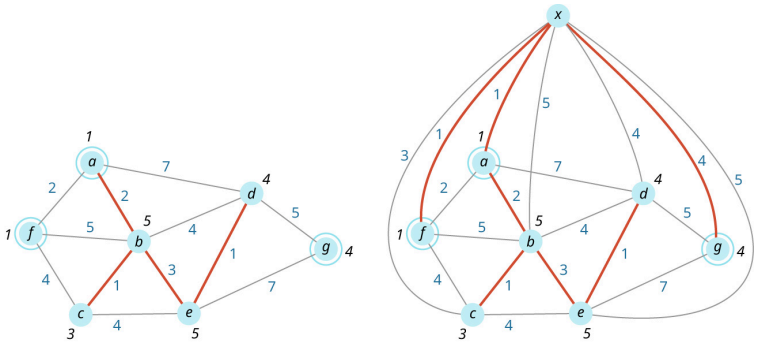


Figure 3.21 The problem of finding a minimum spanning tree in a graph with vertex weights can be reduced to the problem of finding a minimum spanning tree in a graph *without* vertex weights. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

We say that this more complicated municipal broadband planning problem reduces to the minimum spanning tree problem because we can design a reduction algorithm consisting of preprocessing and postprocessing procedures.

- Preprocess: Introduce an extra vertex that does not represent a real location but connects to every address (vertex) in the city. The edge weight of each connection is the cost of installing the broadband node directly at that location.
- Postprocess: After computing a minimum spanning tree for the preprocessed graph, remove the extra vertex that we added during the processing step. Any edges connected to the extra vertex represent a direct broadband node installation, while other edges between real locations are just the same network connections that we saw earlier.

Reduction algorithms enable algorithm designers to solve problems without having to modify existing algorithms or algorithm design patterns. Reduction algorithms allow algorithm designers to rely on optimized canonical algorithms rather than designing a solution by composing algorithm design patterns, which can lead to performance or correctness bugs. Reduction algorithms also enable computer scientists to make claims about the relative difficulty of a problem. If we know that a problem *A* reduces to another problem *B*, *B* is as difficult to solve as *A*.

3.5 Sample Algorithms by Problem

Learning Objectives

By the end of this section, you will be able to:

- Discover algorithms that solve data structure problems
- Understand graph problems and related algorithms

Earlier, we introduced several computing problems, like searching for a target value in a list or implementing an abstract data type (lists, sets, maps, priority queues, graphs). Although every computational problem is unique, these types of problems often share significant similarities with other problems. Computer scientists have identified many canonical problems that represent these common problem templates. Although each of these canonical problems may have many algorithmic solutions, computer scientists have also identified canonical algorithms for solving these problems. In this section, we will introduce canonical problems and survey canonical algorithms for each problem.

Data Structure Problems

Data structure problems are not only useful for implementing data structures, but also as fundamental algorithm design patterns for organizing data to enable efficient solutions to almost every other computing problem.

Searching

Searching is the problem of retrieving a target element from a collection of elements. Searching in a linear data structure, such as an array list, can be done using either sequential search or binary search.

Sequential Search Algorithm

A **sequential search algorithm** is a searching algorithm that sequentially checks the collection element-by-element for the target. The runtime of sequential search is in O(*N*) with respect to *N*, the number of elements in the list (Figure 3.22).

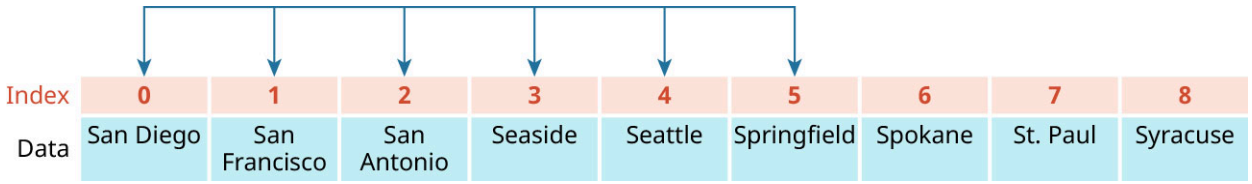


Figure 3.22 Sequential search is a search algorithm that checks the collection element by element to find a target element. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Binary Search Algorithm

A **binary search algorithm** recursively narrows down the possible locations for the target in the sorted list. Initially, the algorithm has no information—the target can be anywhere in the entire range of the sorted list. By comparing the target to the middle element of the range, we can rule-out half of the elements in the range. This process can be repeated until we have found the expected location of the target in the sorted list. The runtime of binary search is in O(log *N*) with respect to *N*, the number of elements in the sorted list, so long as indexing is a constant-time operation (see Figure 3.11).

Although the sequential search algorithm works with any collection type such as lists, sets, dictionaries, and priority queues, the binary search algorithm relies on a sorted list with access to elements by index. Consequently, binary search is efficient on array lists that provide constant-time access to the element at any index and inefficient on linked lists, which do not enable constant-time access to elements by index. Binary search relies on the structure of the sorted list to repeatedly rule-out half of the remaining elements.

Binary search trees represent the concept of binary search in a tree data structure by arranging elements in the tree in sorted order from left to right. Ideally, the root node represents the middle element in the sorted tree and each child roughly divides each subtree in half. But, in the worst case, a binary search tree can look exactly like a linked list where each node contains either zero children or one child. Although such a tree still arranges its elements in sorted order from left to right, comparing the target to each node only reduces the size of the problem by one element rather than ruling out half of the remaining elements—the worst-case binary search tree degrades to sequential search. Balanced binary search trees, such as AVL trees, addressed this worst-case scenario by maintaining the AVL property of balance between left and right subtrees.

Sorting

Sorting is the problem of rearranging elements into a logical order, typically from least-valued (smallest) to greatest-valued (largest). Sorting is a fundamental problem not only because of the tasks that it directly solves, but also because it is a foundation for many other algorithms such as the binary search algorithm or Kruskal's algorithm for the minimum spanning tree problem.

The most common type of sorting algorithm solves the problem of **comparison sorting**, or sorting a list of elements where elements are not assigned numeric values but rather defined in relation to other elements. For simplicity, the data are typically assumed to be stored in an array list for indexed access, and the sorting algorithm can either return a new sorted list or rearrange the elements in the array list so that they appear in sorted order.

Merge Sort Algorithm

A merge sort algorithm recursively divides the data into sublists until sublists are one element long—which we know are sorted—and then merges adjacent sorted sublists to eventually return the sorted list. The merge operation combines two sorted sublists to produce a new, larger sorted sublist containing all the elements in sorted order. The actual rearranging of elements occurs by repeatedly applying the merge operation on pairs of adjacent sorted sublists, starting with the smallest single-element sublists, to larger two-element sublists, and eventually reaching the two halves of the entire list of elements. The runtime of merge sort is in $O(N \log N)$ with respect to N, the number of elements (see Figure 3.17).

Quicksort Algorithm

A **quicksort algorithm** recursively sorts data by applying the binary search tree algorithm design pattern to partition data around pivot elements. Whereas the merge sort algorithm rearranges elements by repeatedly merging sorted sublists after each recursive subproblem, the quicksort algorithm rearranges elements by partitioning data before each recursive subproblem. The partition operation takes a pivot and rearranges the elements into three sections, from left to right: the sublist of all elements less than the pivot, the pivot element, and the sublist of all elements greater than (or equal to) the pivot. Each of the two sublists resulting from the partition operation is a recursive quicksort subproblem; when both of the sublists are sorted, the entire list will be sorted. The runtime of quicksort depends on the choice of each pivot element during the execution of the recursive algorithm, but in practice, for most of the inputs, the runtime is in $O(N \log N)$ with respect to N, the number of elements (Figure 3.23).

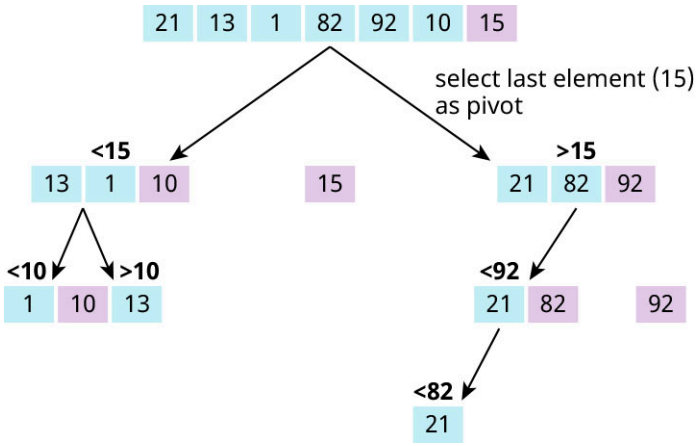


Figure 3.23 Quicksort is a divide and conquer sorting algorithm that sorts elements by recursively partitioning elements around a pivot. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Heapsort Algorithm

A **heapsort algorithm** adds all elements to a binary heap priority queue data structure using the comparison operation to determine priority value, and returns the sorted list by repeatedly removing from the priority queue element by element. The runtime of heapsort is in $O(N \log N)$ with respect to N, the number of elements. The logarithmic time factor is due to the time it takes to add or remove each element from the binary heap (Figure 3.24).

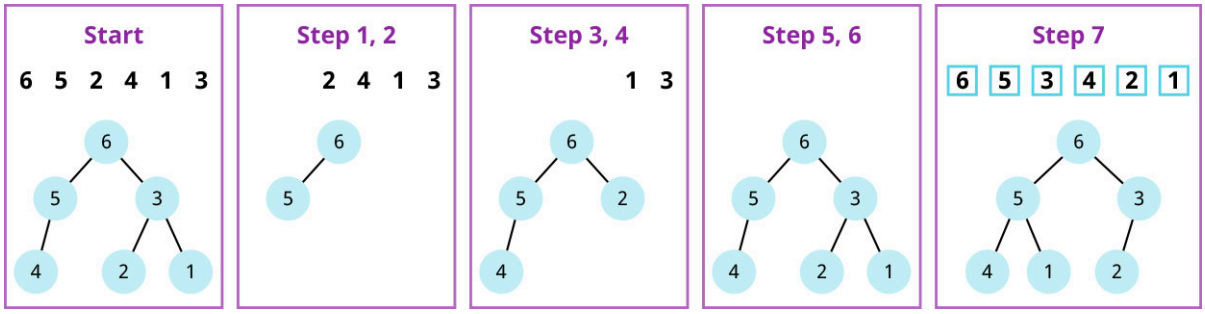


Figure 3.24 Heapsort uses the binary heap data structure to sort elements by adding and then removing all elements from the heap. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Many comparison sorting algorithms share the same O(N log N) runtime bound with respect to N, the number of elements. Computer scientists have shown with a combinatorial proof that, in the worst case, a comparison sorting algorithm cannot do better than O(N log N) comparisons. It is impossible to design a worst-case O(N) runtime comparison sorting algorithm. In fact, a commonly used version of heapsort (which is also asymptotically faster) is to first build a binary heap (i.e., first arrange the input numbers in an array, then repeatedly call a function to turn the original array into a binary heap; one can show that the running time of this part is linear in N, which is why this is faster than constructing the heap by adding numbers one by one), then remove elements one by one from the end of the heap.

But not all sorting problems are comparison sorting problems. In fact, a commonly used version of heapsort (which is also asymptotically faster) is to first build a binary heap (i.e., first arrange the input numbers in an array, then repeatedly call a function to turn the original array into a binary heap. One can show that the running time of this part is linear in N, which is why this is faster than constructing the heap by adding numbers one by one), then remove elements one by one from the end of the heap as explained previously.

Another type of sorting problem that is not restricted to pairwise comparisons is known as **count sorting**, or sorting a list of elements by organizing elements into categories and rearranging the categories into a logical order. For example, the problem of sorting a deck of cards can be seen as a count sorting problem if we put the cards into numeric stacks and then rearrange the stacks into a logical order. By changing the assumptions of the problem, count sorting algorithms can run in O(N) time by first assigning each element to its respective category and then unpacking each category so that elements appear in sorted order.

Hashing

Hashing is the problem of designing efficient algorithms which map each object to an integer so that most (if not all) objects will be assigned distinct integers. Although hashing algorithms are often specific to each data type, there exist some general approaches for designing hashing algorithms. The hash value of a simple data type such as an integer can just be the value of the integer itself. The hash value of a string of text can be some mathematical combination of the numeric value of each character in the string. Likewise, the hash value of a collection such as a list can be some combination of the underlying numeric data within each element in the collection.

Hashing has a variety of applications spanning computer systems, database systems, computer security, and searching algorithms. For example, hashing algorithms are often used designing secure systems for protecting stored passwords even after a security breach occurs. In the context of data structure problems, hashing offers a different approach than binary search. Instead of relying on pairwise comparisons to narrow down the expected location of an element in a sorted list in logarithmic time, hashing search algorithms can instead directly index an element by hash value in constant time. If binary search trees implement sets and maps by applying the concept of binary search, a **hash table** implements sets and maps by applying the concept of hashing (Figure 3.25). Rather than organize elements in sorted order from left to right as in a binary search tree, hash tables store and retrieve elements in an array indexed by hash value.

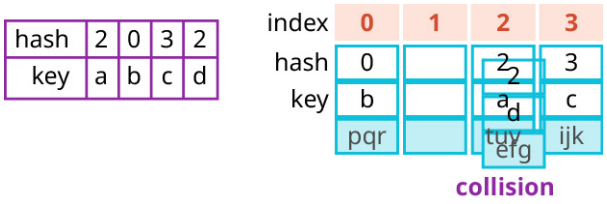


Figure 3.25 Hash tables data structures apply hashing to implement abstract data types such as sets and maps, but must handle collisions between elements that share the same hash value. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Hashing search algorithms are often preferred over binary search algorithms for their runtime benefits, but they come with unique drawbacks. Ideally, different objects would hash to different hash values. But the combinatorial explosion of possibilities for unique strings or collections of complex data means that a **collision**, or a situation in which multiple objects hash to the same integer index value, is inevitable since integers in a computer can typically only represent a certain, fixed range of integer numbers. Combinatorial explosion is not only a problem for the design of efficient algorithms, but also the design of efficient data structures too.

Graph Problems

While data structure problems focus primarily on storage and retrieval of elements in a collection, graph problems include a wide variety of computing problems involving the graph data structure. Unlike other data structures, graphs include not only elements (vertices) but also relationships between elements (edges). Graph problems often ask algorithm designers to explore the graph in order to answer questions about elements and the relationships between elements.

Traversal

Traversal is the problem of exploring all the vertices in a graph. Graph data structures differ from tree data structures in that there is no explicit root node to begin the traversal and edges can connect back to other parts of the graph. In general, there is no guarantee of hierarchy in a graph. Graph traversal algorithms such as depth-first search and breadth-first search begin at an arbitrary start vertex and explore outwards from the start vertex while keeping track of a set of explored vertices.

Depth-First Search

A **depth-first search** is a graph traversal algorithm that recursively explores each neighbor, continuing as far possible along each subproblem depth-first (Figure 3.26). Explored vertices are added to a global set to ensure that the algorithm only explores each vertex once. The runtime of depth-first search is in $O(|V| + |E|)$ with respect to $|V|$, the number of vertices, and $|E|$, the number of edges.

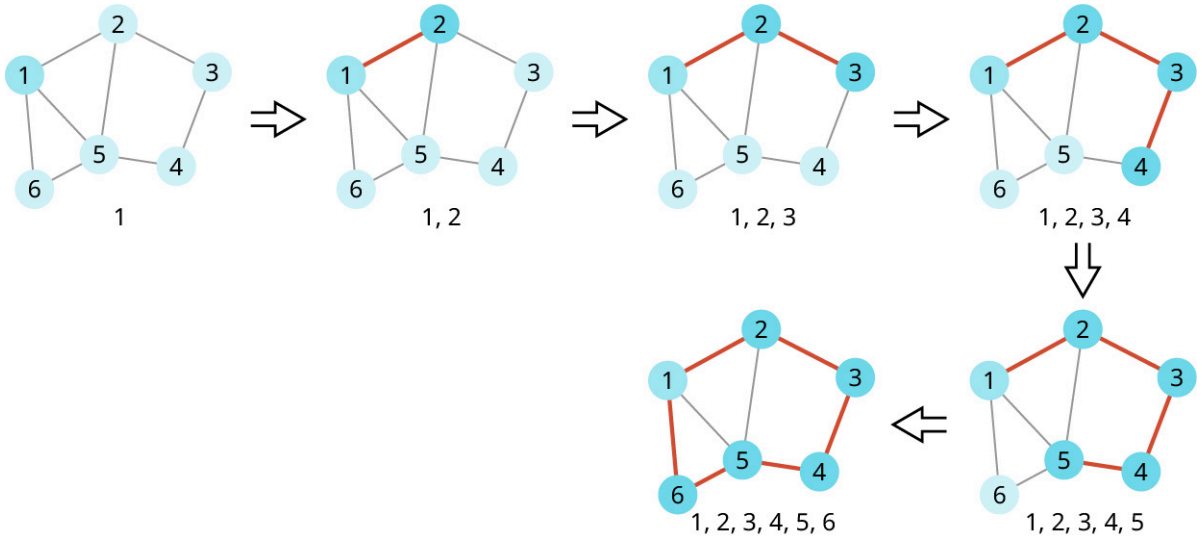


Figure 3.26 Depth-first search is a graph traversal algorithm that continues as far down a path as possible from a start vertex before backtracking. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Breadth-First Search

A **breadth-first search** iteratively explores each neighbor, expanding the search level-by-level breadth-first (Figure 3.27). Explored vertices are also added to a global set to ensure that the algorithm explores each vertex once and in the correct level-order. The runtime of breadth-first search is also in $O(|V| + |E|)$ with respect to $|V|$, the number of vertices, and $|E|$, the number of edges.

Graph traversal algorithms are the foundational algorithm design patterns for most graph processing algorithms. Many algorithms require some amount of exploration, and that exploration typically starts at some vertex and continues processing each reachable vertex at most once. A **reachable vertex** can be reached if a path or sequence of edges from the start vertex exists. As opposed to depth-first search, breadth-first search has the benefit of exploring vertices closer to the start before exploring vertices further from the start, which can be useful for solving problems such as unweighted shortest paths.

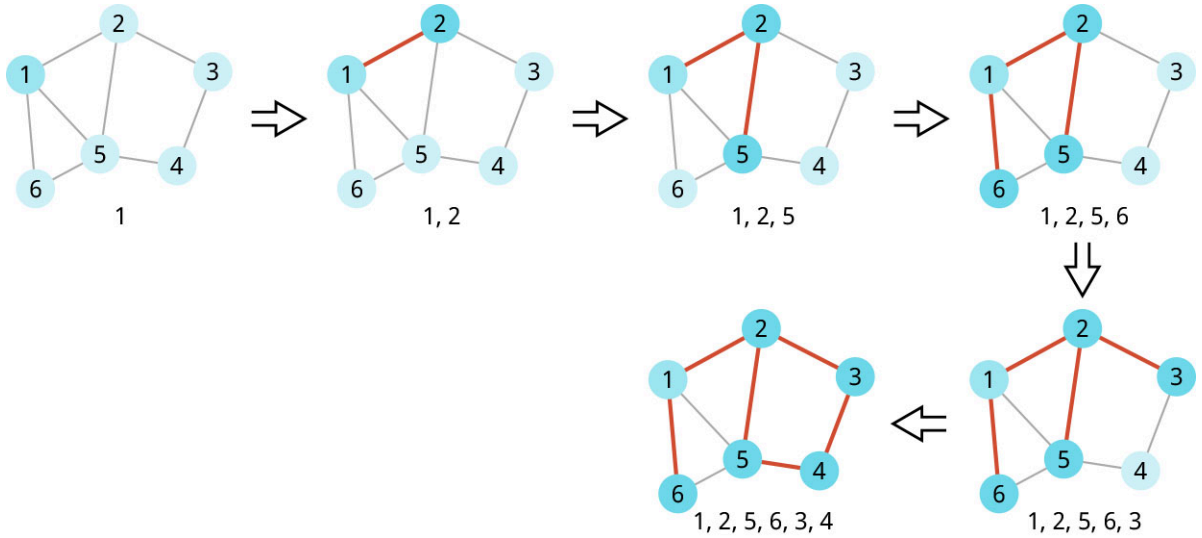


Figure 3.27 Depth-first search is a graph traversal algorithm that continues as far down a path as possible from a start vertex before backtracking. Breadth-first search is a graph traversal algorithm that explores level by level expanding outward from the start vertex. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Minimum Spanning Trees

Minimum spanning trees is the problem of finding a lowest-cost way to connect all the vertices to each other,

where *cost* is the sum of the selected edge weights. The two canonical greedy algorithms for finding a minimum spanning tree in a graph are Kruskal's algorithm and Prim's algorithm. Both algorithms repeatedly apply the rule of selecting the next lowest-weight edge to an unconnected part of the graph. The output of a minimum spanning tree algorithm is a set of $|V| - 1$ edges connecting all the vertices in the graph with the least total sum of edge weights, where $|V|$ is the number of vertices.

Kruskal's Algorithm

Kruskal's algorithm begins by considering each vertex as an independent "island," and the goal of the algorithm is to repeatedly connect islands by selecting the lowest-cost edges. A specialized data structure (called *disjoint sets*) is typically used to keep track of the independent islands. The runtime of Kruskal's algorithm is in $O(|E| \log |E|)$ with respect to $|E|$, the number of edges (Figure 3.28).

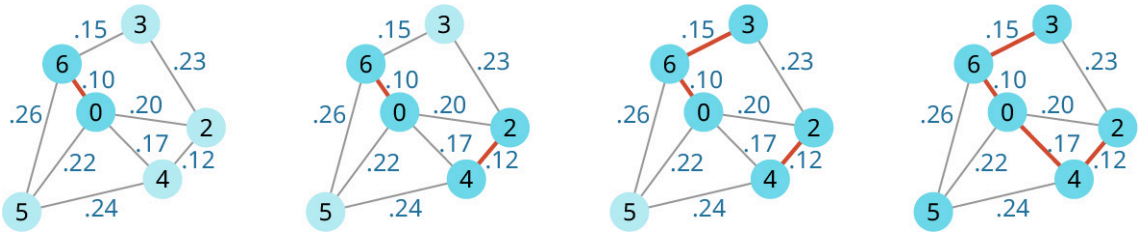


Figure 3.28 Kruskal's algorithm repeatedly selects the next lightest edge that connects two independent "islands." (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Prim's Algorithm

Prim's algorithm grows a minimum spanning tree one edge at a time by selecting the lowest-weight edge to an unexplored vertex. The runtime of Prim's algorithm is in $O(|E| \log |V| + |V| \log |V|)$ with respect to $|V|$, the number of vertices, and $|E|$, the number of edges (Figure 3.29).

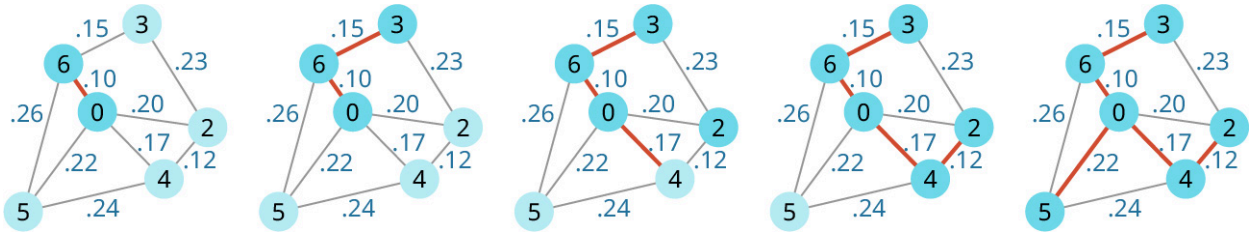


Figure 3.29 Prim's algorithm expands outward from the start vertex by repeatedly selecting the next lightest edge to an unreached vertex. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Shortest Paths

The output of a shortest paths algorithm is a **shortest paths tree**, the lowest-cost way to get from one vertex to every other vertex in a graph (Figure 3.30). The **unweighted shortest path** is the problem of finding the shortest paths in terms of the number of edges. Given a start vertex, breadth-first search can compute the unweighted shortest paths tree from the start vertex to every other reachable vertex in the graph by maintaining a map data structure of the path used to reach each vertex.

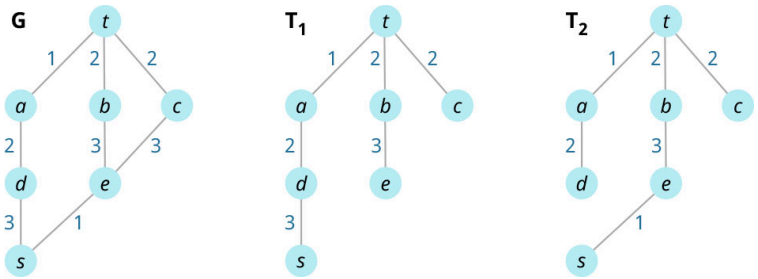


Figure 3.30 Three shortest paths trees of the lowest-cost way to get from the start vertex to every other vertex in the graph are shown. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Weighted Shortest Path

A **weighted shortest path** is the problem of finding the shortest paths in terms of the sum of the edge weights. Earlier, we introduced Prim's algorithm, a minimum spanning tree algorithm that maintains a priority queue of edges in the graph ordered by weight and repeatedly selects the next lowest-cost edge to an unconnected part of the graph.

THINK IT THROUGH

Weighted Shortest Paths for Navigation Directions

One of the most direct applications of the shortest paths problem is to provide recommended routes for navigation directions in real-world mapping. Many applications use distance as the metric for edge weight, so the shortest path between two points represents the real-world route with the smallest distance between the two places.

What does a distance metric not consider when providing a recommended route? What values are centered and emphasized by even using a shortest paths algorithm for recommending routes?

Dijkstra's Algorithm

Dijkstra's algorithm maintains a priority queue of vertices in the graph ordered by distance from the start and repeatedly selects the next shortest path to an unconnected part of the graph. Dijkstra's algorithm is almost identical to Prim's algorithm except processing shortest paths (sequences of edges) rather than individual edges. Dijkstra's algorithm grows a shortest paths tree one shortest path at a time by selecting the next shortest path to an unexplored vertex. The runtime of Dijkstra's algorithm is in $O(|E| \log |V| + |V| \log |V|)$ with respect to $|V|$, the number of vertices, and $|E|$, the number of edges (Figure 3.31).

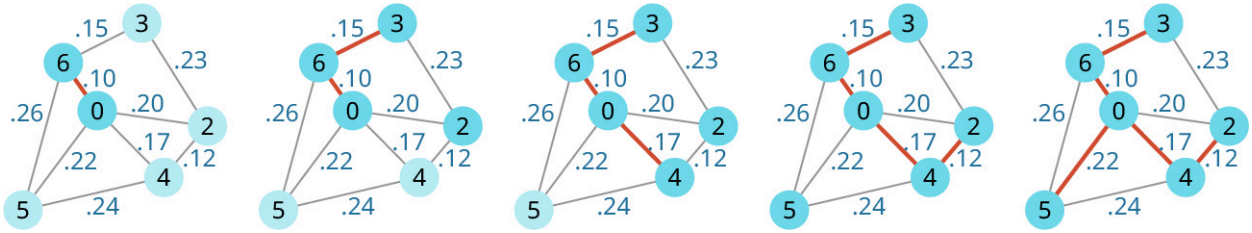


Figure 3.31 Dijkstra's algorithm expands outward from the start vertex by repeatedly selecting the next lowest-cost path to an unreached vertex. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

3.6 Computer Science Theory

Learning Objectives

By the end of this section, you will be able to:

- Understand the models and limits of computing
- Relate Turing machines to algorithms
- Describe complexity classes
- Interpret NP-completeness
- Differentiate between P and NP

Throughout this chapter, we have introduced several techniques and canonical case studies for the design and analysis of algorithms—oftentimes focusing on the ideas and details behind individual algorithms. But the study of algorithms is more than just the study of individual algorithms, algorithm design, or even algorithm analysis.

Models of Computation

Computers include basic algorithms for solving problems like adding, subtracting, or comparing two numbers. Computers, owing to their roots in calculators, are optimized to solve these problems; these basic algorithms are constant-time operations. What programming offers is the ability to define our own algorithms that can be used to solve more complex problems, such as searching, sorting, and hashing. Unfortunately, these programmed algorithms are not as fast as basic operations. We have even seen certain problems deal with a combinatorial explosion in the number of potential solutions. For many of these problems, the best-known algorithms do not do much better than brute-force, which takes exponential time.

In the bigger picture, computer science engages the central question of how humans can encode intelligence. Our discussion of algorithm design grounded the activity in problem modeling, the process of encoding a complex real-world phenomenon or problem in a more abstract or simple form. How is problem modeling constrained by the model of computation, or the rules of the underlying computer that executes an algorithm? Why are certain problems challenging for computers to execute?

Combinatorial explosion poses a problem for computer algorithms because our model of computation assumes computers only have a single thread of execution and only execute one basic operation on each step. If we overturn some part of this assumption, either by creating computers with multiple processors or by creating more sophisticated operations, then it might be possible to deal with combinatorial explosion. Almost all of today's computer hardware, ranging from massive supercomputers to handheld smartphones, rely at least to some degree on expanding the model of computation to compute solutions to problems more efficiently. Even so, much of today's computer hardware still relies on the same fundamental programming assumptions: that there are variables to represent data and arithmetic or logical operations.

Turing Machines

In the 1800s, Charles Babbage imagined a mechanical machine—the Analytical Engine—that could automatically calculate mathematical formulas. Ada Lovelace then extrapolated that the Analytical Engine could solve more general algorithms by using loops to repeat processes and variables to represent data. Lovelace's vision of algorithms represented a synthesis between human intuition and mathematical reasoning. In the mid-1900s, Lovelace's ideas inspired Alan Turing to imagine a more general notion of algorithms and machines that could run those algorithms. A **Turing machine** is an abstract model of computation for executing any computer algorithm. A Turing machine describes computers in terms of three key ideas:

1. a memory bank for storing data.
2. an instruction table, where each instruction can either:
    a. store a value to the memory bank.
    b. retrieve a value from the memory bank.
    c. perform a basic operation on a value.
    d. set which instruction will be executed next by modifying the program counter.
3. a program counter that keeps track of the current instruction in the instruction table.

A Turing machine executes a computer algorithm by following each instruction specified by the program counter. An algorithm can use these basic operations to compute the sum of 1 and 1.

1. Store the value 1 to address A in the memory bank.
2. Store the value 1 to address B in the memory bank.
3. Add the values at addresses A and B and then store the result at address A.

What makes computers useful is not just the fact that they can calculate numbers, but that they can encode logic in the instructions. Instead of just computing the sum of 1 and 1, this program continues adding 1 to a growing sum stored at address A.

1. Store the value 1 to address A in the memory bank.

2. Store the value 1 to address B in the memory bank.
3. Add the values at addresses A and B and then store the result at address A.
4. Set the program counter to execute step 3 next.

The Turing machine abstract model of computation assumes a single thread of execution following each instruction in an algorithm. Although today's computers are much more efficient than the first computers that realized the Turing machine, most computers still rely on the same fundamental assumptions about how to execute algorithms. The O(N)-time sequential search algorithm, though it might execute 1,000 times faster on today's computers, still grows linearly with respect to the size of the input. An O(2^N)-time brute-force algorithm, though it might execute 1,000 times faster on today's computers, still grows exponentially with respect to the size of the input. Even as computers become faster over time, inefficient algorithms still cannot be used to solve any problems larger than a few thousand elements.

Complexity Classes

One subfield of computer science is theoretical computer science, which studies models of computation, their application to algorithms, and the complexity of problems. The complexity of a problem is the complexity (in terms of time or memory resources required) of the most efficient algorithms for solving the problem. Theoretical computer scientists are interested in understanding the difficulty of a problem in terms of time complexity, space complexity, and some other complexity measures.

In this chapter, we have focused on solving problems known to have polynomial time algorithms. Searching, sorting, hashing, traversal, minimum spanning trees, or shortest paths are all examples of problems in the **polynomial (P) time complexity class** because they are all problems that have runtimes with a polynomial expression such as O(1), O(log N), O(N), O(N log N), O(N^2), O(N^3). In general, these problems are considered **tractable** because computers can solve them in a reasonable amount of time. But there are many problems that are considered **intractable** because they do not have efficient, polynomial-time algorithms.

The **nondeterministic polynomial (NP) time complexity class** refers to all problems that can be solved in polynomial time by a nondeterministic algorithm. A **nondeterministic algorithm** is a special kind of Turing machine, which at each step can nondeterministically choose which instruction to execute, and is considered to successfully find a solution if any combination of these nondeterministic choices eventually lead to a correct solution. In other words, in contrast to a deterministic algorithm, such as a greedy algorithm, which must repeatedly apply a simple rule to deterministically select the next element in a solution, a nondeterministic algorithm is able to simultaneously explore all the possible choices. We do not yet have computers that can execute nondeterministic algorithms, but if we did, then we would be able to efficiently solve any combinatorial problem by relying on the special power of nondeterminism.

Technically, all P problems are also NP problems because we already have deterministic algorithms for solving them and therefore do not need to rely on the special power of nondeterminism. For example, Dijkstra's algorithm provides a deterministic polynomial-time solution to the shortest paths problem by building up a shortest paths tree from the start vertex outward. This application of the greedy algorithmic paradigm relies on the structure of the shortest paths tree, since the shortest path to a point further away from the start must build on the shortest path to a point closer to the start.

NP-complete Problems

NP-complete refers to all the hardest NP problems—the combinatorial problems for which we do not have deterministic polynomial-time algorithms. More precisely, a problem PI is said to be NP-complete if PI is in NP and for every problem in NP, there is a reduction that reduces the problem to PI. For example, a **longest path**, or the problem of finding the highest-cost way to get from one vertex to another without repeating vertices, is an NP-complete problem opposite to shortest paths (Figure 3.32). What makes longest paths so much harder to solve than shortest paths? For one, there is no underlying structure to the solution that we can use to repeatedly apply a simple rule as in a greedy algorithm. With the shortest paths problem, we could rely on the

shortest paths tree to inform the solution. But in longest paths, the goal is to *wander* around the graph. The longest path between any two vertices will probably involve traversing as many edges as possible to maximize distance, visiting many vertices along the way. For some graphs, the longest paths might even visit all the vertices in the graph. In this situation, the longest paths do not form a tree and instead involve ordering all the vertices in the graph for each longest path. Identifying the correct longest path then requires listing out all the possible paths in the graph—a combinatorial explosion in the combinations of edges and vertices that can be selected to form a solution.

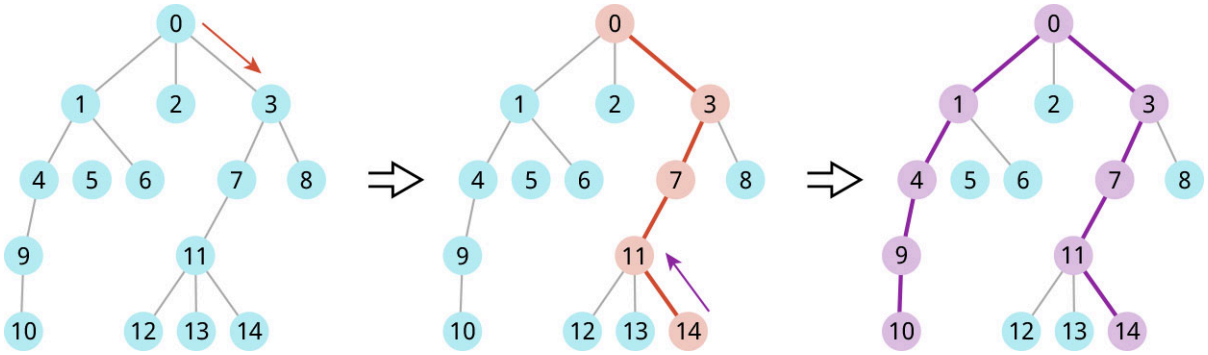


Figure 3.32 The longest path in a graph maximizes the distance, which often (but not always) involves visiting many vertices along the way. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Related to the problem of longest paths is the **traveling salesperson problem (TSP)**, which is the problem of, given the path cost of every pair of vertices, finding the lowest-cost tour, or path from a start vertex visiting every vertex in the graph once including a return to the start vertex. Compared to the TSP, which is finding the lowest-cost tour, the longest paths problem is like finding the highest-cost tour. What makes both these problems difficult is that we do not have a simple rule for selecting the next element to include in the solution. Unless we have the special power of nondeterminism, it is hard to tell from the beginning which edge is the right one to include in the final solution. Applying a simple rule like, "Select the edge with the lowest cost," might not necessarily lead to the overall lowest-cost tour. Even though this simple rule worked for the problem of minimum spanning trees, the additional restriction of a tour rather than a tree makes the TSP a much harder problem to solve efficiently. Although we have efficient algorithms for shortest paths, we do not have efficient algorithms for the shortest tour (TSP).

INDUSTRY SPOTLIGHT

Delivery Logistics

Companies such as Amazon, FedEx, UPS, and others that rely on logistics to deliver goods to various locations seek to optimize the sequence of stops to save costs. The only way to achieve this would be to rely on an optimal algorithm for the traveling salesperson problem. The TSP aims to find the minimum distance tour that visits all the vertices such that no vertex is visited more than once. But this is not a perfect match for real-world delivery logistics. Fuel or battery efficiency, for example, is not just about distance traveled, but also the speed of travel, the time spent idling, and even the way that the route is organized. In the United States, where vehicles drive on the right side of the road, safety can be improved by reducing the number of left-hand turns across the divider. Drivers might also want to take breaks during a trip. Modeling all these factors requires carefully formulating the problem and considering the limits of TSP.

How do we know if a problem is NP-complete? Earlier, we introduced reduction as an algorithm paradigm that is not only useful for solving problems, but also for relating the difficulty of problems. A reduction from a difficult problem A to another problem B proves that B is as difficult as A. It turns out that all NP-complete problems can be reduced to all the others, so an algorithm for solving *any* NP-complete problem solves *every*

NP-complete problem.

LINK TO LEARNING

The longest paths problem and the traveling salesperson problem are just two examples of NP-complete problems. Visit A Graph of NP-Complete Reductions (https://openstax.org/r/76NPCompReuct) to visualize many NP-complete problems and their relationships to each other. For example, the longest paths problem (LPT) and the traveling salesperson problem (TS) both reduce to Hamiltonian paths (HP). In turn, Hamiltonian paths (HP) reduce to vertex cover (VC) which reduces to 3-satisfiability (3-SAT).

P versus NP

Longest paths and TSP are just two among thousands of NP-complete problems for which we do not have efficient algorithms. The question of P versus NP asks whether it is possible to design a deterministic polynomial-time algorithm for solving any—and therefore all—of NP-complete problems. There are two possible answers to the question of P versus NP:

- If P = NP, then there is a deterministic polynomial-time algorithm for solving all NP-complete problems.
- If P ≠ NP, then there are NP-complete problems that cannot be solved with a deterministic polynomial-time algorithm.

Most theoretical computer scientists believe that P ≠ NP; in other words, it is *impossible* to design a deterministic polynomial-time algorithm for longest paths, TSP, or any other NP-complete problem. However, they do not have any definite proof that P = NP or P ≠ NP. An efficient algorithm for any one NP-complete problem would not only directly solve routing and logistics problems but would also enable massive advancements in drug discovery through scientific simulation, for instance. It would also break essentially all modern Internet security and password systems—among thousands of other problems.

Chapter Review

Key Terms

abstract data type (ADT) consists of all data structures that share common functionality
abstraction process of simplifying a concept in order to represent it in a computer
adjacent in a graph abstract data type, the relationship between two vertices connected by an edge
algorithm analysis study of the results produced by the outputs as well as how the algorithm produces those outputs
algorithm design pattern solution to well-known computing problems
algorithmic paradigm common concept and ideas behind algorithm design patterns
algorithmic problem solving refers to a particular set of approaches and methods for designing algorithms that draws on computing's historical connections to the study of mathematical problem solving
array list data structure that stores elements next to each other in memory
asymptotic analysis evaluates the time that an algorithm takes to produce a result as the size of the input increases
asymptotic notation mathematical notation that formally defines the order of growth
AVL tree balanced binary search tree data structure often used to implement sets or maps that organizes elements according to the AVL tree property
AVL tree property requires the left and right subtrees to be balanced at every node in the tree
balanced binary search tree introduces additional properties that ensure that the tree will never enter a worst-case situation by reorganizing elements to maintain balance
Big O notation most common type of asymptotic notation in computer science used to measure worst case complexity
binary heap binary tree data structure used to implement priority queues that organizes elements according to the heap property
binary logarithm tells how many times a large number needs to be divided by 2 until it reaches 1
binary search algorithm recursively narrows down the possible locations for the target in the sorted list
binary search tree tree data structure often used to implement sets and maps that organizes elements according to the binary tree property and the search tree property
binary tree property requires that each node can have either zero, one, or two children
breadth-first search iteratively explores each neighbor, expanding the search level-by-level breadth-first
brute-force algorithm solves combinatorial problems by systematically enumerating all potential solutions in order to identify the best candidate solution
canonical algorithm well-known algorithm
case analysis way to account for variation in runtime based on factors other than the size of the problem
child node descendant of another node
collision situation where multiple objects hash to the same integer index value
combinatorial explosion exponential number of solutions to a combinatorial problem that makes brute-force algorithms unusable in practice
combinatorial problem involves identifying the best candidate solution out of a space of many potential solutions
comparison sorting sorting of a list of elements where elements are not assigned numeric values but rather defined in relation to other elements
complexity condition based on the degree of computational resources that an algorithm consumes during its execution in relation to the size of the input
compression problem of representing information using less data storage
constant type of order of growth that does not take more resources as the size of the problem increases
correctness whether the outputs produced by an algorithm match the expected or desired results across the range of possible inputs

cost model characterization of runtime in terms of more abstract operations such as the number of repetitions

count sorting sorting a list of elements by organizing elements into categories and rearranging the categories into a logical order

cryptography problem of masking or obfuscating text to make it unintelligible

data structure complex data type with specific representation and specific functionality

data structure problem computational problem involving the storage and retrieval of elements for implementing abstract data types such as lists, sets, maps, and priority queues

data type determines how computers process data by defining the possible values for data and the possible functionality or operations on that data

depth-first search graph traversal algorithm that recursively explores each neighbor, continuing as far possible along each subproblem depth-first

Dijkstra's algorithm maintains a priority queue of vertices in the graph ordered by distance from the start and repeatedly selects the next shortest path to an unconnected part of the graph

divide and conquer algorithm algorithmic paradigm that breaks down a problem into smaller subproblems (divide), recursively solves each subproblem (conquer), and then combines the result of each subproblem in order to inform the overall solution

edge relationship between vertices or nodes

element individual data point

experimental analysis evaluates an algorithm's runtime by recording how long it takes to run a program implementation of it

functionality operations such as adding, retrieving, and removing elements

graph represents binary relations among collection of entities, specifically vertices and edges

graph problem computational problem involving graphs that represent relationships between data

greedy algorithm solves combinatorial problems by repeatedly applying a simple rule to select the next element to include in the solution

hash table implements sets and maps by applying the concept of hashing

hashing problem of assigning a meaningful integer index for each object

heap property requires that the priority value of each node in the heap is greater than or equal to the priority values of its children

heapsort algorithm adds all elements to a binary heap priority queue data structure using the comparison operation to determine priority value and returns the sorted list by repeatedly removing from the priority queue element-by-element

index position or address for an element in a list

interval scheduling problem combinatorial problem involving a list of scheduled tasks with the goal of finding the largest non-overlapping set of tasks that can be completed

intractable problems that do not have efficient, polynomial-time algorithms

Kruskal's algorithm greedy algorithm that sorts the list of edges in the graph by weight

leaf node node at the bottom of a tree that has no children

linear type of order of growth where the resources required to run the algorithm increases at about the same rate as the size of the problem increases

linear data structure category of data structures where elements are ordered in a line

linked list data structure that does not necessarily store elements next to each other and instead works by maintaining, for each element, a link to the next element in the list

list ordered sequence of elements and allows adding, retrieving, and removing elements from any position in the list

logarithm tells how many times a large number needs to be divided by a small number until it reaches 1

longest path problem of finding the highest-cost way to get from one vertex to another without repeating vertices

map represents unordered associations between key-value pairs of elements, where each key can only

appear once in the map

matching problem of searching for a text pattern within a document

merge sort canonical divide and conquer algorithm for comparison sorting

minimum spanning tree problem of finding a lowest-cost way to connect all the vertices to each other

model of computation rules of the underlying computer that is ultimately responsible for executing the algorithm

node represents an element in a tree or graph

nondeterministic algorithm special kind of Turing machine, which at each step can non-deterministically choose which instruction to execute and is considered to successfully find a solution if any combination of these nondeterministic choices leads to a correct solution

nondeterministic polynomial (NP) time complexity class all problems that can be solved in polynomial time by a nondeterministic algorithm

NP-complete all the hardest NP problems—the combinatorial problems for which we *do not* have deterministic polynomial-time algorithms

order of growth geometric prediction of an algorithm's time or space complexity as a function of the size of the problem

perfectly balanced for every node in the binary search tree, its left and right subtrees contain the same number of elements

polynomial (P) time complexity class all problems that have runtimes described with a polynomial expression such as O(1), O(log N), O(N), O(N log N), O(N^2), O(N^3)

Prim's algorithm greedy algorithm that maintains a priority queue of vertices in the graph ordered by connecting edge weight

priority queue represents a collection of elements where each element has an associated priority value

problem task with specific input data and output data corresponding to each input

problem model simplified, abstract representation of a more complex real-world problem

program realization or implementation of an algorithm written in a formal programming language

quicksort algorithm recursively sorts data by applying the binary search tree algorithm design pattern to partition data around pivot elements

reachable vertex vertex that can be reached if a path or sequence of edges from the start vertex exists

reduction algorithm solves problems by transforming them into other problems

representation particular way of organizing a collection of elements

root node node at the top of the tree

runtime analysis study of how much time it takes to run an algorithm

search tree property requires that elements in the tree are organized least-to-greatest from left-to-right

searching problem of retrieving a target element from a collection of elements

sequential search algorithm searching algorithm that sequentially checks the collection element-by-element for the target

set represents an unordered collection of unique elements and allows adding, retrieving, and removing elements from the set

shortest path problem of finding a lowest-cost way to get from one vertex to another

shortest paths tree output of the shortest paths problem, the lowest-cost way to get from one vertex to every other reachable vertex in a graph

sorting problem of rearranging elements into a logical order

space complexity formal measure of how much memory an algorithm requires during its execution as it relates to the size of the problem

step basic operation in the computer, such as looking up a single value, adding two values, or comparing two values

string problem computational problem involving text or information represented as a sequence of characters

subproblem smaller instance of a problem that can be solved independently

time complexity formal measure of how much time an algorithm requires during execution as it relates to the size of the problem

tractable problems that computers can solve in a reasonable amount of time

traveling salesperson problem (TSP) problem of, given the path cost of every pair of vertices, finding the lowest-cost tour, or path from a start vertex visiting every vertex in the graph once including a return to the start vertex

traversal problem of exploring all the vertices in a graph

tree hierarchical data structure

Turing machine abstract model of computation for executing any computer algorithm

unweighted shortest path problem of finding the shortest paths in terms of the number of edges

vertex represents an element in a graph or special type of it such as a tree

weighted shortest path problem of finding the shortest paths in terms of the sum of the edge weights

Summary

3.1 Introduction to Data Structures and Algorithms

- Data structures represent complex data types for solving real-world problems. Data structures combine specific data representations with specific functionality.
- Abstract data types categorize data structures according to their functionality and ignore differences in data representation. Abstract data types include lists, sets, maps, priority queues, and graphs.
- To select an appropriate data structure, first select an abstract data type according to the problem requirements. Then, select an appropriate data structure implementation for the abstract data type.
- Linear data structures organize elements in a line, ideal for implementing the list abstract data type. Linear data structures include array lists and linked lists.
- Linear data structures can implement any abstract data type. The study of data structures in general focuses on opportunities to improve efficiency (in terms of execution time or memory usage) over linear data structures.
- Tree data structures organize elements in a hierarchy of levels defined by parent-child relationships. Trees are defined with a root node at the top of the tree, parent-child relationships between each level, and leaf nodes at the bottom of the tree.
- Binary search trees require that elements in the tree are organized least-to-greatest from left-to-right. Binary search trees are often used to implement the set and map abstract data types.
- Balanced binary search trees and binary heaps represent two approaches for avoiding the worst-case situation with binary search trees. Binary heaps are often used to implement the priority queue abstract data type.
- Graph data structures focus on explicitly modeling the relationships between elements. Graphs afford access not only to elements, but also to the relationships between elements.

3.2 Algorithm Design and Discovery

- Just like how many data structures can represent the same abstract data type, many algorithms exist to solve the same problem. In algorithmic problem-solving, computer scientists solve formal problems with specific input data and output data that correspond to each input.
- Modeling is the process of representing a complex phenomenon such as a real-world problem as a formal problem. Modeling is about abstraction: the simplification or erasure of details so that the problem can be solved by a computer.
- Historically, the model of computation emphasized specialized algorithms operating on a modest model of the underlying phenomenon. Modeling is a violent but also necessary act in order to simplify the problem so that it can be solved by a computer.
- Searching is the problem of retrieving a target element from a collection of many elements. Sequential search and binary search are two algorithms for solving the search problem.
- To solve real-world problems, computer scientists compose, modify, and apply algorithm design patterns,

such as search algorithms.

- Algorithm analysis is the study of the outputs produced by an algorithm as well as how the algorithm produces those outputs.
- Correctness considers whether the outputs produced by an algorithm match the expected or desired results across the range of possible inputs. Correctness is defined as a match between the algorithm and the model of the problem, not between the algorithm and the real-world.
- Correctness is complicated by the complexity of social relationships, power, and inequity in the real-world. Since algorithms automate processes and operate in existing power structures, they are likely to reproduce and amplify social injustice.
- In addition to correctness, computer scientists are also interested in complexity, or measuring the computational resources that an algorithm consumes during its execution in relation to the size of the input.

3.3 Formal Properties of Algorithms

- Runtime analysis is a study of how much time it takes to run an algorithm. Experimental analysis is a runtime analysis technique that involves evaluating an algorithm's runtime by recording how long it takes to run a program implementation of it.
- Time complexity is the formal measure of how much time an algorithm requires during execution as it relates to the size of the problem. The goal of time complexity analysis is to produce a simple and easy-to-compare characterization of the runtime of an algorithm as it relates to the size of the problem.
- Space complexity is the formal measure of how much memory an algorithm requires during execution as it relates to the size of the problem.
- Steps in time complexity analysis are to identify a metric for representing the size of the problem; to model the number of steps needed to execute the algorithm; and to formalize the model using either precise English or asymptotic notation to define the order of growth. Big O notation is the most common type of asymptotic notation in computer science.
- Differences in orders of growth are massive: as the input size grows, the difference between orders of growth becomes more and more vast. For problems dealing with just 1,000 elements, the time it would take to run an exponential-time algorithm on that problem exceeds the current age of the universe—whereas that same-size problem running on the same computer would take just 1 second on a quadratic-time algorithm.
- In practice, across applications working with large amounts of data, $O(N^2)$ is often considered the limit for real-world algorithms. For algorithms that need to run frequently on large amounts of data, algorithm designers target $O(N)$, $O(\log N)$, or $O(1)$.

3.4 Algorithmic Paradigms

- Algorithmic paradigms are the common concepts and ideas behind algorithm design patterns, such as divide and conquer algorithms, brute-force algorithms, greedy algorithms, and reduction algorithms.
- Divide and conquer algorithms break down a problem into smaller subproblems (divide), recursively solve each subproblem (conquer), and then combine the result of each subproblem to inform the overall solution. Recursion is an algorithm idea fundamental to divide and conquer algorithms that solves complex problems by dividing input data into smaller, independent instances of the same problem known as subproblems.
- Binary search is an example of divide and conquer algorithm with a single recursive subproblem. Merge sort is an example of a divide and conquer algorithm with two recursive subproblems.
- Brute-force algorithms solve combinatorial problems by systematically enumerating all potential solutions in order to identify the best candidate solution. Combinatorial problems identify the best candidate solution out of a space of many potential solutions.
- Brute-force algorithms exist for every combinatorial problem, but they are not typically used in practice because of long run time issues. To enumerate all potential solutions, a brute-force algorithm must generate every possible combination of the input data.

- Greedy algorithms solve combinatorial problems by repeatedly applying a simple rule to select the next element to include in the solution. Unlike brute-force algorithms that solve combinatorial problems by generating all potential solutions, greedy algorithms instead focus on generating just one solution.
- Greedy algorithms are not always guaranteed to compute the best solution depending on the assumptions and goals of the problem. A greedy algorithm for the interval scheduling problem will not compute the correct result if we choose to complete the shortest tasks.
- Kruskal's algorithm and Prim's algorithm are two examples of greedy algorithms for the minimum spanning trees problem. These algorithms are a rare example of a greedy algorithm that is guaranteed to compute the correct result.
- Reduction algorithms solve problems by transforming them into other problems. In other words, reduction algorithms delegate most of the work of solving the problem to another algorithm meant for a different problem.
- Reduction algorithms allow algorithm designers to rely on optimized canonical algorithms rather than designing a solution by composing algorithm design patterns, which can lead to performance or correctness bugs. Reduction algorithms also enable computer scientists to make claims about the relative difficulty of a problem.

3.5 Sample Algorithms by Problem

- Data structure problems focus on the storage and retrieval of elements for implementing abstract data types such as lists, sets, maps, and priority queues. Data structure problems include sorting, searching, and hashing.
- Searching is the problem of retrieving a target element from a collection of elements. Searching in a linear data structure such as an array list can be done using either sequential search or binary search.
- Sorting is the problem of rearranging elements into a logical order, typically from least-valued (smallest) to greatest-valued (largest). Sorting is a fundamental problem not only because of the tasks that it directly solves, but also because it is a foundation for many other algorithms such as the binary search algorithm or Kruskal's algorithm for the minimum spanning tree problem.
- Merge sort and quicksort are two examples of divide and conquer algorithms for sorting. Heapsort is a sorting algorithm that relies on adding to a heap and then repeatedly removing each element in sorted order.
- Hashing is the problem of assigning a meaningful integer index (hash value) for each object. Hash tables are a data structure for implementing sets and maps by applying the concept of hashing.
- Graph problems include a wide variety of problems involving the graph data type. Graph problems include traversal, minimum spanning trees, and shortest paths.
- Traversal is the problem of exploring all the vertices in a graph. Depth-first search and breadth-first search are both graph traversal algorithms that expand outward from a start vertex, ultimately visiting every reachable vertex.
- Minimum spanning trees is the problem of finding a lowest-cost way to connect all the vertices to each other, where cost is the sum of the selected edge weights. The two canonical greedy algorithms for finding a minimum spanning tree in a graph are Kruskal's algorithm and Prim's algorithm.
- Shortest paths is the problem of finding a lowest-cost way to get from one vertex to another. The output of a shortest paths algorithm is a shortest paths tree from the start vertex to every other vertex in the graph.
- Breadth-first search computes the unweighted shortest paths tree, the shortest paths in terms of the number of edges. Dijkstra's algorithm computes the weighted shortest paths tree, the shortest paths in terms of the sum of the edge weights.

3.6 Computer Science Theory

- Problem modeling is constrained by the model of computation, or the rules of the underlying computer that is ultimately responsible for executing the algorithm. Combinatorial explosion poses a problem for computer algorithms because our model of computation assumes computers only have a single thread of execution and only execute one basic operation on each step.

- A Turing machine is an abstract model of computation for executing any computer algorithm. A Turing machine describes computers in terms of three key ideas: a memory bank, an instruction table, and a program counter.
- Although today's computers are much more efficient than the first computers that realized the Turing machine, most computers still rely on the same fundamental assumptions about how to execute algorithms. Even as computers become faster over time inefficient algorithms still cannot be used to solve any problems larger than a few thousand elements.
- The complexity of a problem is the complexity (i.e., the time or memory resources required) of the most efficient algorithms for solving the problem. In this chapter, we have focused on solving problems known to have polynomial time algorithms that can be described with a polynomial expression such as O(1), O(log N), O(N), O(N log N), O(N2), O(N3).
- Nondeterministic polynomial (NP) time complexity class refers to all problems that can be solved in polynomial time by a nondeterministic algorithm. A nondeterministic algorithm is a kind of algorithm that can rely on the special power of exploring infinitely many possible "alternate universes" in order to complete a computation.
- Technically, all P problems are also NP problems because we already have deterministic algorithms for solving them and therefore do not need to rely on the special power of nondeterminism. NP-complete refers to all the hardest NP problems—the combinatorial problems for which we do not have deterministic polynomial-time algorithms.
- Longest paths and the traveling salesperson problem (TSP) are two well-known examples of NP-complete problems. What makes both these problems difficult is that we do not have a simple rule for selecting the next element to include in the solution.
- All NP-complete problems can be reduced to all the others, so an algorithm for solving any NP-complete problem solves every NP-complete problem. The question of P versus NP asks whether it is possible to design a deterministic polynomial-time algorithm for solving any—and therefore all—of these NP-complete problems.
- Most theoretical computer scientists believe that it is impossible to design an efficient algorithm for longest paths, TSP, or any other NP-complete problems. An efficient algorithm for any one NP-complete problems would not only directly solve routing and logistics problems but would also enable massive advancements in drug discovery through scientific simulation, for instance. It would also break essentially all modern Internet security and password systems—among thousands of other problems.

Review Questions

1. Why did we introduce tree data structures as an alternative to linear data structures?
   a. Some complex data can only be represented with a tree data structure.
   b. Some simple data can only be represented with a tree data structure.
   c. Linear data structures cannot implement sets and maps.
   d. Tree data structures are typically more effective at implementing sets and maps.

2. How does the graph abstract data type differ from other abstract data types?
   a. It can model the relationships between elements.
   b. It is more efficient than other abstract data types.
   c. It can solve problems that other abstract data types cannot solve.
   d. It does not specify a particular data structure implementation.

3. What abstract data type do binary heaps most commonly implement?
   a. lists
   b. sets
   c. maps
   d. priority queues

4. What is one way to describe the relationship between algorithms, problems, and modeling?
- **a.** Algorithms are the foundation for problem models.
- **b.** Algorithms solve a model of a problem.
- **c.** Each algorithm can only be used to solve a single problem.
- **d.** Each problem can only have a single model.

5. At what point do computer scientists apply algorithm design patterns?
- **a.** when learning canonical algorithms
- **b.** when modeling a problem
- **c.** when solving new problems
- **d.** when analyzing an algorithm

6. How does the model of computation relate to the problem model?
- **a.** The model of computation is synonymous with problem model.
- **b.** Problem models constrain the model of computation.
- **c.** The model of computation constrains the problem modeling process.
- **d.** The model of computation describes a single algorithm for each problem model.

7. Why is case analysis important?
- **a.** Case analysis provides an alternative to asymptotic analysis.
- **b.** Case analysis focuses on small inputs.
- **c.** Case analysis simplifies the step-counting by introducing a cost model.
- **d.** Case analysis considers factors other than the size of the problem.

8. What is true about the order of growth of binary search with respect to the size of the sorted list?
- **a.** In the best case, the order of growth of binary search is constant.
- **b.** In the best case, the order of growth of binary search is logarithmic.
- **c.** In the worst case, the order of growth of binary search is constant.
- **d.** In the worst case, the order of growth of binary search is linear.

9. How does time complexity relate to space complexity?
- **a.** Time complexity measures efficiency according to the size of the problem, while space complexity does not.
- **b.** Both time and space complexity measure the efficiency of algorithms as they relate to the nature of the problem.
- **c.** Time complexity focuses on asymptotic analysis while space complexity focuses on experimental analysis.
- **d.** Both time and space complexity can apply methods from asymptotic analysis and experimental analysis.

10. What are the three steps in divide and conquer algorithms?

11. Why do many greedy algorithms fail to compute the best solution to a combinatorial problem?

12. What are the three steps in reduction algorithms?

13. What is quicksort's algorithm design pattern and algorithmic paradigm?
- **a.** Quicksort is an application of the binary search tree algorithm design pattern and an example of the divide and conquer algorithmic paradigm.
- **b.** Quicksort is an application of the binary search tree algorithm design pattern and an example of the brute-force algorithmic paradigm.
- **c.** Quicksort is an application of the binary search tree algorithm design pattern and an example of the greedy algorithmic paradigm.
- **d.** Quicksort is an application of the binary search tree algorithm design pattern and an example of the

randomized incremental construction algorithmic paradigm.

14. What graph problems can breadth-first search solve?
- **a.** exponential node tree
- **b.** minimum spanning trees
- **c.** unweighted shortest paths
- **d.** weighted shortest paths

15. What is a primary drawback of hashing?
- **a.** There can be collisions as the same element can hash to multiple values.
- **b.** There can be collisions between multiple elements that hash to the same value.
- **c.** Hashing is slower than binary search for search problems.
- **d.** Hashing is faster than binary search for search problems.

16. What is the relationship between Turing machines and models of computation?

17. What is one of the three key ideas of the Turing machine?
- **a.** infinite memory by virtualization
- **b.** a memory bank for storing data
- **c.** using divide and conquer to always reduce an algorithm to O(n) runtime
- **d.** using divide and conquer to always reduce an algorithm to O(1) runtime

18. What is P versus NP?
- **a.** P refers to the polynomial time complexity class, whereas NP refers to the nondeterministic polynomial time complexity class.
- **b.** P refers to any Big O notation past O(N^3), whereas NP refers to any Big O notation less than O(N^3).
- **c.** NP is a constant runtime, whereas P is polynomial runtime.
- **d.** P refers to constant runtime, whereas NP is linear runtime.

Conceptual Questions

1. Explain the difference between algorithms and programs.

2. Explain the difference between data structures and abstract data types.

3. Explain the relationship between data representation, data structures, and algorithms.

4. What is the relationship between search algorithms and the searching problem?

5. What is the relationship between search algorithms and the autocomplete feature?

6. Why is algorithmic correctness difficult to determine?

7. What are some limitations of experimental analysis?

8. What are some benefits of experimental analysis over asymptotic analysis?

9. Why is a 1-element list the best-case situation for sequential search?

10. If phone numbers are ten digits long and can contain digits from zero through nine, what is the total number of potential phone numbers?

11. Why might we prefer a sub-optimal greedy algorithm over a correct brute-force algorithm?

12. What's problematic about the statement, "municipal broadband planning reduces to Kruskal's algorithm"?

13. Describe the relationship between the pivot element and the left and right sublists after the first partition in quicksort.

14. The runtime of Kruskal's algorithm is in O($|E|$ log $|E|$) with respect to $|E|$, the number of edges. What

primarily contributes to this linearithmic order of growth?

15. Both Prim's algorithm and Dijkstra's algorithm are greedy algorithms that organize vertices in a priority queue data structure. What is the difference between the ordering of vertices in the priority queue for Prim's algorithm and Dijkstra's algorithm?

16. What is the relationship between models of computation and algorithms?

17. What is the common difficulty preventing us from designing an efficient algorithm for solving NP-complete problems?

18. What are the consequences of P = NP?

Practice Exercises

1. Binary search trees organize elements in ascending sorted order within the tree. However, binary search trees can become unbalanced. In the worst-case, a binary search tree can look exactly like a linked list. Give an order for inserting the following numbers into a binary search tree such that the resulting tree appears like a linked list: 7, 3, 8, 1, 2, 5, 6, 4.

2. Consider these two different approaches for implementing the priority queue abstract data type using a linked list data structure: (1) organize the elements by decreasing priority value, and (2) organize the elements arbitrarily. Describe algorithms for inserting an element as well as retrieving and removing the highest-priority element from these two data structures.

3. In our definition of a priority queue, we emphasized retrieval and removal of the highest-priority elements—a maximum priority queue. What if we wanted to instead prioritize retrieval and removal of the lowest-priority elements—a minimum priority queue? Describe a simple change that we could make to make any maximum priority queue function as a minimum priority queue.

4. Formally describe the problem model for a drug administration medical system in terms of input data and output data represented as lists, sets, maps, priority queues, and graphs.

5. Formally describe the problem model for a music recommendation system in terms of input data and output data represented as lists, sets, maps, priority queues, and graphs.

6. There can sometimes be thousands, if not millions, of results that match a Web search query. To make this information more helpful to humans, we might want to order the results according to a relevance score such that more-relevant results appear before less-relevant results. Describe how we can solve this problem of retrieving the *N*-largest elements using the following algorithm design patterns: (1) a sorting algorithm, and (2) a priority queue abstract data type.

7. What is the best-case and worst-case Big O order of growth of sequential search with respect to *N*, the size of the list?

8. What is the best-case and worst-case Big O order of growth of binary search with respect to *N*, the size of the sorted list?

9. What is the worst-case Big O order of growth of sequential search followed by binary search with respect to *N*, the size of the sorted list?

10. What two sublists are combined in the final step of merge sort on the list [9, 8, 2, 5, 4, 1, 3, 6]?

11. If a connected graph has unique edge weights, will Kruskal's algorithm find the same minimum spanning tree as Prim's algorithm? How about a connected graph with duplicate edge weights?

12. What is a reduction algorithm for the problem of finding the median element in a list?

13. The heapsort algorithm applies a binary heap priority queue ordered by the comparison operation to sort

elements. What can we say about the first element removed from the binary heap if it implements a minimum priority queue? What about the last element removed? Is the binary heap data structure sorted?

14. Hashing algorithms can provide a constant-time solution to the search problem under certain conditions. What are the conditions necessary to ensure the runtime of a hashing search algorithm is constant?

15. Why is it the case that depth-first search cannot be directly applied to compute an unweighted shortest paths tree?

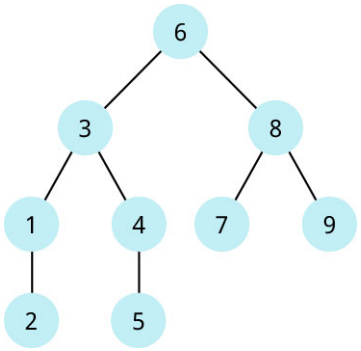
Problem Set A

1. Linked lists and binary search trees are two examples of linked data structures, in which each node in the data structure is connected to other nodes. Given a linked list of the numbers one through seven, organized in ascending sorted order, draw two different examples of binary search trees representing the same numbers.



(attribution: Copyright Rice University, OpenStax, under CC BY 4.0 license)

2. Given the following binary search tree, draw the corresponding ascending-sorted linked list containing the same numbers.



(attribution: Copyright Rice University, OpenStax, under CC BY 4.0 license)

3. We defined autocomplete as a problem that takes as an input a string of letters that might represent the start of a word, and outputs a list of words that match the input. Describe two other problem models for autocomplete and define the ramifications of the models.

4. Compare and contrast the autocomplete problem models. What are the trade-offs of each problem model?

5. Consider the problem of arranging desktop icons in a grid layout so that they fill column-by-column, starting from the left side of the screen. Suppose we are given a list of desktop icons in alphabetical order and that placing an icon in a grid position is a constant-time operation. What is the Big O order of growth of an algorithm that takes each icon in the given order and places each icon in the next open grid position with respect to *N*, the number of desktop icons?

6. Suppose we're given a list of desktop icons in alphabetical order, but that placing an icon in a grid position is a linear-time operation with respect to the number of icons. (Perhaps a sequential search is needed to check that the icon has not already been placed on the desktop.) What is the Big O order of growth of this icon arrangement algorithm with respect to *N*, the number of desktop icons?

7. Why does the greedy interval scheduling, which selects the cheapest, least time-consuming task, fail to maximize the number of completed tasks?

8. What is a simple rule for greedy interval scheduling that will maximize the number of completed tasks?

9. The runtime of most binary heap priority queue operations is in O(log N) with respect to N, the size of the priority queue. The logarithmic factor is due to the height of the binary heap data structure. But finding an element in a binary heap typically requires sequential search. How can we apply the hashing algorithm design pattern to find an element in a heap in constant time?

10. The runtime of breadth-first search is in O($|V| + |E|$) because each reachable vertex and edge is processed one-by-one during the level-order graph traversal. Why is the runtime of Prim's algorithm in O($|E| \log |V| + |V| \log |V|$)? Explain in terms of the time it takes to process each vertex or edge in the graph.

Problem Set B

1. Each unique key in a map is paired with one (possibly not-unique) value. Sometimes, we want to associate more than one value with a given key. Describe how we can use a list or set abstract data type to associate more than one value with a unique key.

2. Each vertex in a graph can be labeled with a unique identifier, such as a unique number. Describe the relationship between adjacent vertices. How could we use abstract data types such as lists, sets, and maps to represent these relationships?

3. Describe how we might implement the graph abstract data type using other abstract data types such as lists, sets, and/or maps. Explain for graphs whose edges have associated weights as well as graphs whose edges do not have associated weights.

4. Describe two or three different problem models for a medical system designed to help doctors recommend preventive care for patients.

5. Compare and contrast the medical system problem models. What are the trade-offs of each problem model?

6. How does the choice of problem model affect potential algorithms? Describe an algorithm for each problem model.

7. Consider an autocomplete implementation that relies on a sequential search to find all matching terms. What is the worst-case Big O order of growth for computing a single autocomplete query with respect to N, the number of potential terms?

8. Consider an autocomplete implementation that sorts the list of potential terms and then performs binary search to find the matching terms. If the order of growth of the sorting algorithm is in O($N \log N$), what is the worst-case Big O order of growth for computing a single autocomplete query with respect to N, the number of potential terms?

9. Why might algorithm designers prefer to use binary search instead of sequential search for autocomplete?

10. In a 1-D space where each point is defined with x coordinates, a common problem is to find the closest pair of points in the space: the pair of points that has the least distance among all potential pairs of points. What is a brute-force algorithm for solving this 1-D closest pair problem? What is the Big O notation order of growth of this algorithm?

11. In a 2-D space, each point is defined with (x, y) coordinates. What is a brute-force algorithm for solving the 2-D closest pair problem?

12. What are the recursive subproblems in a divide and conquer algorithm for solving for the closest pair problem?

13. Digital images are represented in computers as a 2-D grid of colored pixels. In image editing, the flood fill problem takes a given starting pixel and replaces all the pixels in a contiguous region that share the same color with a different color. How can we represent the colored pixels in a digital image as a graph with

vertices and edges for the flood fill problem?

14. How should we modify a graph traversal algorithm to solve the flood fill problem using your graph representation?

15. What is a reduction algorithm for reducing from the flood fill problem to the graph traversal problem? In this case, the graph traversal algorithm cannot be modified. Instead, define a preprocessing step to create a graph representation that encodes the flood fill same-color rule.

Thought Provokers

1. Maps can be defined in terms of sets: every map is a set whose elements represent key-value pairs, where the key must be unique, but the value might not be unique. Consider other relationships between abstract data types. Can sets and maps be defined in terms of graphs? Can lists be defined in terms of maps? Can priority queues be defined in terms of maps? Why might it be useful to define abstract data types in terms of other data types?

2. Graph theory refers to the mathematical study of graphs. How might a graph theorist describe linked lists and tree data structures? How does this differ from our use of abstract data types?

3. Sorting and searching are two examples of data structure problems related to the storage and retrieval of elements. Where do sorting and searching appear in linear data structures, tree data structures, and/or graph data structures?

4. What are some benefits and drawbacks of simpler problem models, as they compare to more complicated problem models?

5. The formal definition of Big O notation does not exactly match our working definition for orders of growth. Do some additional research to explain why binary search is also in O(N).

6. Since binary search is in O(N), it is also true that binary search is in O(N^2). Explain why computer scientists might find O(N^2) to be a less useful description of the runtime of binary search compared to O(log N).

7. We can show that the worst-case order of growth for any comparison sorting algorithm must be at least linearithmic using an argument from combinatorial explosion in the number of unique permutations of elements in a list. What are the number of unique permutations of a list with N elements? How many comparison questions need to be asked to identify a particular permutation from among all the permutations? How do these questions relate to comparison sorting?

8. Breadth-first search is a fundamental algorithm design pattern for graph problems. How is breadth-first search applied as a foundation for designing greedy algorithms such as Prim's algorithm and Dijkstra's algorithm? How does Kruskal's algorithm fit into these algorithm design patterns and paradigms? If Prim's algorithm is analogous to sorting in Kruskal's algorithm, why is there no analogue to the Dijkstra's algorithm in sorting as well?

9. Suppose we want to find the longest path from a starting vertex to an ending vertex in a graph. How might a nondeterministic algorithm solve this problem in polynomial time?

10. Suppose we want to find the longest path from a starting vertex to an ending vertex in a graph (solving the *function problem*) without using a nondeterministic algorithm. Let's say that P = NP and we have a deterministic polynomial-time algorithm that returns whether there is a path with exactly cost k (solving the *decision problem*). We also know the cost of the actual longest path. How can we repeatedly apply this *decision algorithm* to design a polynomial-time longest paths *function algorithm*?

Labs

1. Simulate patients entering and exiting a hospital emergency room with a priority queue using patients'

time of arrival, basic assessment of severity, and availability of doctors specializing in the appropriate type of care. Decide how to prioritize patients based on a property of each patient, such as their arrival time, numeric severity rating, numeric urgency rating, and availability of care providers. Then, consider how your decision might result in unfair allocation of medical care to patients.

2. Choose a lab from the Ethical Reflections Modules (https://openstax.org/r/76Ethics) for CS1. Follow the instructions to complete the assignment. Once you have finished, answer this additional reflection question connecting back to algorithm design: How did you utilize algorithm design patterns? How did your choice of algorithm designs affect the end outcomes in your program?

3. Experimental analysis: Use a software-based "stopwatch" to compare the time (in microseconds) it takes to run a sequential search versus a binary search for successively larger and larger inputs. Relate experimental analysis to asymptotic analysis. What happens to small arrays? What happens to large-size array inputs? What happens when the target is near the front of the array? What happens when the target is near the end of the array?



4

Linguistic Realization of Algorithms: Low-Level Programming Languages

Figure 4.1 Low-level programming languages support little or no abstraction; they allow programmers to write software in languages that are closer to English and are suitable for system software that powers mobile devices with limited energy and computing resources such as prosthetics. (credit: modification of "Tilly Lockey at the SingularityU The Netherlands Summit 2016" by Sebastiaan ter Burg/Flicker, CC BY 2.0)

Chapter Outline

4.1 Models of Computation
4.2 Building C Programs
4.3 Parallel Programming Models
4.4 Applications of Programming Models

Introduction

The machines we call "computers," including modern desktop computers, laptops, and web servers, are remarkably fast and capacious. However, computer hardware is also embedded in devices that do not fit the traditional definition of computers: home appliances, automobiles, smart thermostats, tools, and televisions. Along with being energy-efficient and affordable, these devices may also need to be lightweight, portable, or even wearable. For these reasons, embedded systems have meager processing speed and memory capacity. This chapter focuses on low-level programming languages, which are used in practice to create software for resource-constrained devices. Efficiency is critical to making these devices useful and economically viable. The efficiency of embedded software is make-or-break; in other words, if we can write efficient code that runs fast and uses little memory, we enable technologies that can help people with their daily lives. Therefore, computer scientists place considerable emphasis on the efficiency and speed of low-level languages, which is why low-level languages are important to society. For efficiency reasons, the syntax of low-level programming languages relies on instructions that are computer-centric and challenging for humans to work with. This has led computer scientists to create "middle-level" languages that emphasize human-readability without compromising efficiency.

Consider our fictional company, TechWorks, which is bringing a line of next-generation prosthetics to the market. While legacy prosthetics are sometimes awkward to use and limited in function, "smart" prosthetics can be revolutionary. These prosthetics are computer-controlled, Internet-connected, and make use of artificial

intelligence, allow those who need prosthetics to enjoy a quality of life that meets or exceeds expectations.

Computer control refers to the ability to manage, organize, or run something on a computer, whereas intelligent control is a class of control techniques that use various artificial intelligence computing approaches. For example, artificial intelligence algorithms can accurately determine the intentions of the wearer and control a prosthetic's motion in an accurate and natural way. Internet connectivity means devices can be conveniently controlled by applications, relay telemetry to healthcare providers, and automatically apply over-the-air updates. TechWorks has fitted a small, inexpensive, energy-efficient system-on-chip computer to their devices, and are using the middle-level language C to implement these features efficiently.

4.1 Models of Computation

Learning Objectives

By the end of this section, you will be able to:

- Define low-level programming languages, including assembly language
- Define middle-level and high-level programming languages, such as C and JavaScript
- Compare and contrast the various programming paradigms

Algorithms are used to solve computational problems and create computational models. A **computational model** is a system that defines what an algorithm does and how to run it. Examples of such computational models include physical devices that can run software, programming languages, or a design specification of such. A programming language is a linguistic application of an algorithm, which uses computational models focused on defining notations for writing code.

Many computational models have been devised for a host of other applications. There are many different roles and perspectives within the worlds of computer science and software development. The end goal of software development is to create working software that can run on a **hardware model**, which itself uses a (hardware) realization of an algorithm that enables specific physical computers to execute software programs. A hardware model is designed for the convenience of a machine, not a human software author, so hardware models are poorly suited to writing code. Computer scientists have created programming languages which are designed specifically for programmers to develop practical applications. These languages are usually classified into high-level (Java, Python) and low-level languages (assembly language). A **high-level programming language** operates at a high level of abstraction, meaning that low-level details such as the management of memory are automated. In contrast, a **low-level programming language** operates at a low level of abstraction. Languages like C and C++ can perform both high-level and low-level tasks.

Most software is designed, written, and discussed in terms of how a program should work. It is basically a series of steps that provide a direction of how the program must be executed. An example of this would be the "Map Reduce model" which is used in distributed systems like the Google search engine to produce search results for large data sets using a complex algorithm. Moving even further away from hardware models, computer scientists have also defined an **abstract model**, which is a technique that derives simpler high-level conceptual models for a computer while exploring the science of what new algorithms can or cannot do.

Modern computers are equipped with a central processor, referred to as a **central processing unit (CPU)**, which is a computer chip capable of executing programs. A CPU's hardware model relies on a specific CPU **instruction set architecture (ISA)** that defines a list of operations that the CPU can execute, such as storing the results of calculations (Figure 4.2). With the advancements of technology, computer engineers have designed computer architectures with increasing sophistication and power. Examples of hardware models include the MOS Technology 6502 architecture used by the Nintendo Entertainment System, the ARM architecture used by mobile phones, and the x86-64 and AMD64 architectures used by modern personal computers. Computer engineers design architectures with hardware specifications, such as execution speed or energy use, in mind. Therefore, hardware models are not suitable for humans to use for communicating

algorithms.

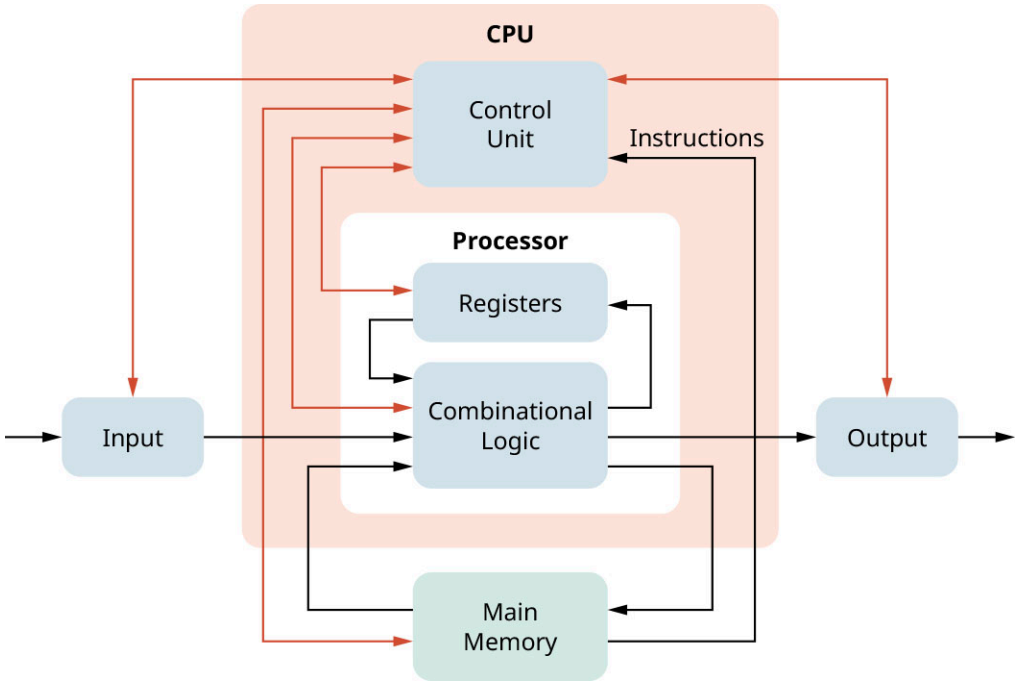


Figure 4.2 A standard CPU model shows how a program logic applies low-level instructions to an input to get an output; the program leverages registers and memory (black arrows) and the CPU orchestrates the overall execution of the program (red arrows). (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

A **programming model** is designed for humans to read and write. A programming model focused on defining a notation for writing code can also be called a programming language. A programming model can be used to implement a software algorithm using a strict set of syntactical rules. A language's syntax can define keywords such as "if." The syntax may include a mathematical **operator**, a fundamental programming operation that combines values, such as "+." The syntax can define punctuation such as ";". Essentially, the syntax gives the precise meaning for what each of these elements directs a computer to do. The text of a program written in a programming language is called **source code**. Software engineers have created practically all software by writing source code in various programming languages. Since a programming model cannot directly execute a program, a **compiler** or **interpreter** must translate source code from a middle-level or high-level language into something a computer can execute.

As mentioned, abstract models are computational models used to think about algorithms in the abstract, rather than being used to create and run software. The goal of an abstract model is to make it easy for people to devise and convey algorithms. Computer scientists use abstract models to create new algorithms, analyze the efficiency of algorithms, and prove facts about what algorithms can and cannot do. An abstract model is not concerned with the details of computer architectures, which makes it easier to focus on these sorts of deep questions. Examples of abstract models include the Random Access Machine, the Turing machine, and the Lambda calculus. The **Random Access Machine** (Figure 4.3) is a CPU that consists of unlimited memory cells that can store any arbitrary value. Just like any other CPU, the PC determines the statement to be executed next. A Random Access Machine can be used to analyze the efficiency of algorithms. A Turing machine (Figure 4.4) is a mathematical model that can implement any algorithm. The **Lambda calculus** is a theoretical computation concept using lambda functions. It was defined by Alonzo Church and inspired the functional programming paradigm, which you will learn more about in Programming Language Paradigms.

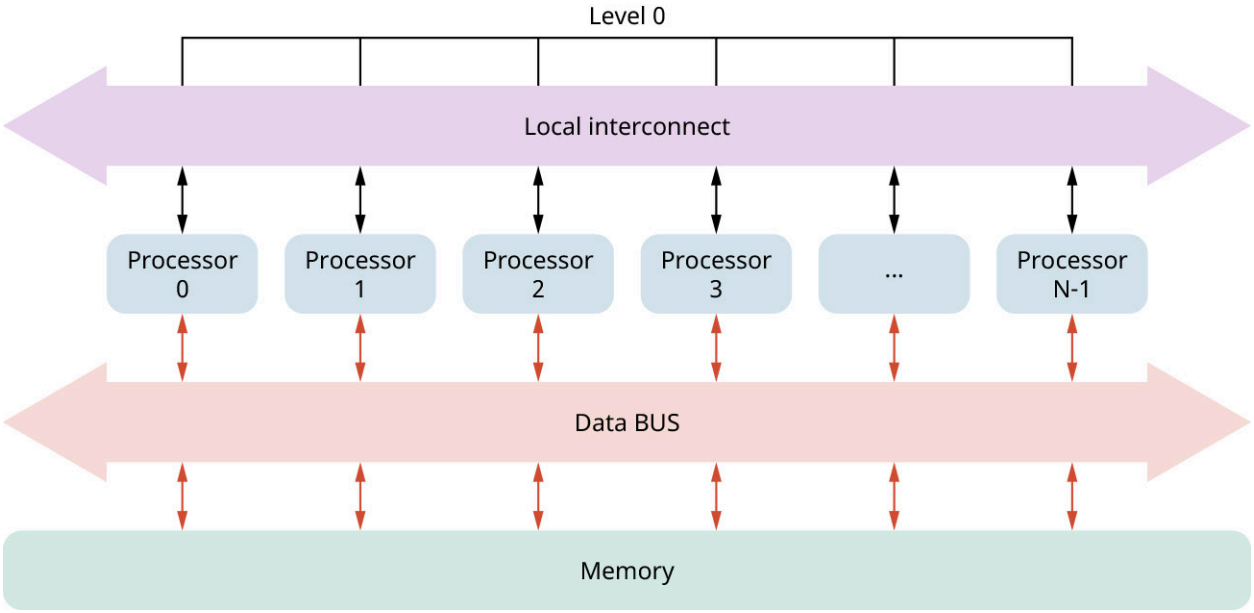


Figure 4.3 A Random Access Machine has unlimited memory cells that can store any arbitrary value. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

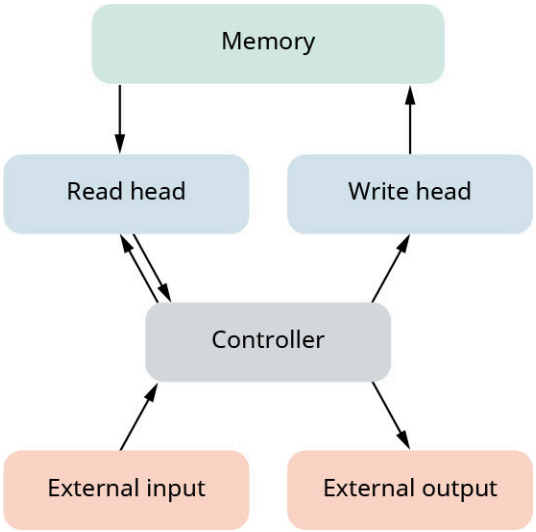


Figure 4.4 A neural Turing machine (NTM) leverages the pattern matching capabilities of neural networks in addition to more traditional computational models. It uses a controller that interacts with external memory resources through attention mechanisms that mimic human attention to improve performance. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

A function defines how to convert an input into an output, and **functional programming** is a paradigm in which algorithms are written as mathematical functions. An example of a functional programming paradigm is a recursion (refer to the following code snippet), where a function is used to call itself. The factorial of an integer can be computed using a recursive function.

```
import java.util.*;
public class Recursion {
  public static int Factorial_Recursion(int Val){
    if(Val==0){
      return 1;
    }
    else
      return Val*Factorial_Recursion(Val-1);
```

```
  }
  public static void main(String[] args){
    Scanner s =new Scanner(System.in);
    System.out.println("Enter an input value: ");
    int Val=s.nextInt();
    System.out.println("The factorial of " + Val + " is: " +
Factorial_Recursion(Val));
  }
}
```

An algorithm described in an abstract model cannot be run directly. First, a software developer must implement the algorithm, which means translating the abstract algorithm into source code in a programming language.

It may be surprising that so many different programming models can exist, and that algorithms can be translated from one model to another. However, this translation is by design; computer science established the **Church-Turing Thesis**, which is a scientific theory stating that an algorithm can be converted from any reasonable computational model to another. The Church-Turing Thesis provides a lens through which computer scientists can invent many computer architectures and programming languages, all of which can run algorithms.

Computer scientists have created terminology to make sense of the similarities and differences among all these programming languages. For example, any programming language can be low level or high level, or it can fall anywhere on the spectrum.

Low-Level Programming

A programming language's **level of abstraction** is the degree to which a computational model, programming language, or piece of software relates to computer hardware. A low-level language has a low level of abstraction, while a high-level language has a high level of abstraction. In a low-level language, the programmer must describe an algorithm in terms that the computer hardware can understand directly. Source code must describe details such as the location of data in memory and the particulars of how the computer calculates arithmetic.

Generally, low-level programs execute faster but are more labor-intensive to create and maintain. In a low-level language, the programmer is forced to think deliberately about how the computer hardware executes, so the finished program usually executes efficiently. However, that deliberate thought takes time and effort. In a high-level language, the programmer is not burdened with thinking about so many details and can finish their work faster while preventing certain types of programming errors from occurring. A compiler automates converting high-level code to low-level code, but that automated process can introduce some inefficiency. In some settings code performance is more important, and in other settings programmer productivity is more important, which is why we have both kinds of languages.

We can think of low-level programming languages in terms of cooking: when you cook a meal from scratch, you control every ingredient and every detail of preparation, so the finished meal has precisely the taste and nutrition that you desire. An alternative is to prepare a meal that uses some prepared ingredients, and when you do that, you lose a lot of control over details, but the process is significantly faster and easier.

There are many examples of low-level programming languages, but the most fundamental language understood by computers is made up of a sequence of digits.

Machine Code

The sequence of binary digits (bits) that can be understood and executed directly by a computer is called **machine code** (Figure 4.5). Machine code is the most low-level language. It is also known as **binary code**. It is

a program in the native format that can be understood by a CPU, in the form of a long series of 0s and 1s. Machine code, or binary code, is the only computational model that a computer can execute; a program written in any other language must be compiled or interpreted into machine code before the program can run. The CPU of a computer is a computer chip capable of executing machine code programs (Figure 4.6). It is impractical for humans to work with machine code directly because a machine code program is not designed to be human-readable. The patterns of 0s and 1s are designed to be convenient for a CPU to decode, not for humans to manipulate; and such programs are long, typically millions or billions of bits long. Another obstacle is that machine code is hardware dependent. As discussed in 5.3 Machine-Level Information Representation, every processor architecture has its own machine language, so machine language written for one architecture (for example, INTEL X86) cannot work on any other architecture (such as ARM). When the very first digital computers were built, and programming languages had yet to be invented, programmers had no choice but to write machine code by hand. However, this is extremely time-consuming and prone to errors, so is almost never done today.

Machine Code

```
11010001100101110110 0
11011001101111000011 0
11000001111000011110 1
01010100001111110010 1
01000000111011101111 1
10000100111000110111
10010011000111000000 0
```

Figure 4.5 Machine code, with its 0s and 1s, is the only computational model a computer can execute. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

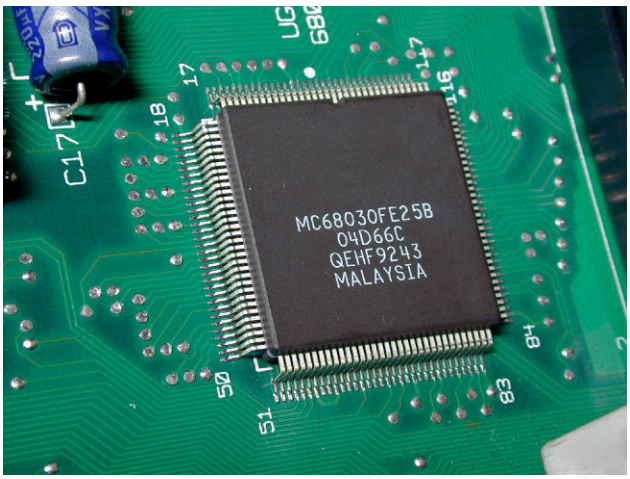


Figure 4.6 A computer chip is a computer's central processing unit (CPU). (credit: modification of "IIci" by raneko/Flickr, CC BY 2.0)

Assembly Language

The low-level language in which every statement corresponds directly to a machine instruction is called **assembly language**. Assembly language is a small step above machine code but is still a very low-level language. Assembly language is a textual representation of machine code. Just like a machine code program, an assembly language program is a series of instructions that a CPU will execute. However, rather than writing the instructions in a binary format of 0s and 1s, each instruction has a textual name such as "ADD" or "MOVE." An **assembler** is a program that translates assembly language source code into machine code. As shown in Figure 4.7, an assembler translates each textual instruction into the corresponding list of 0 and 1 bits.

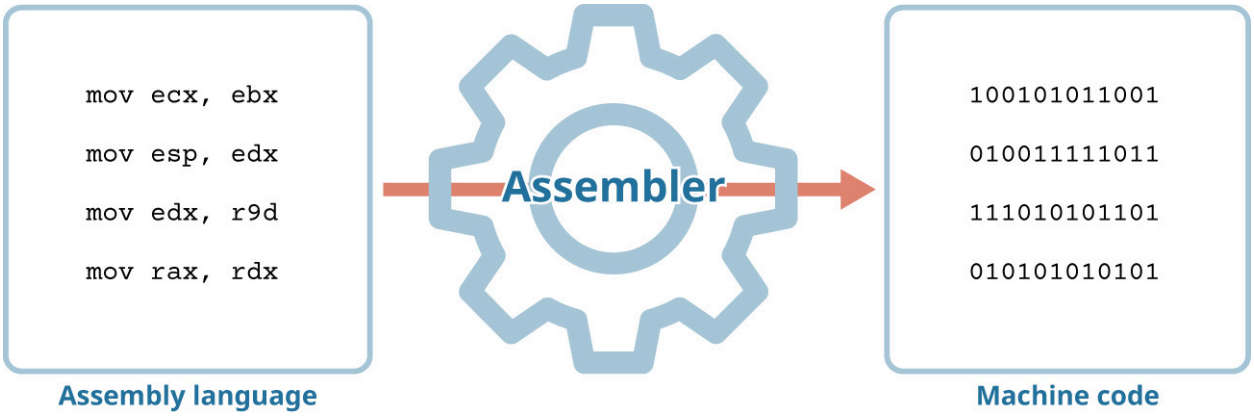


Figure 4.7 An assembler is a program that translates assembly language source code into machine object code. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

While it is practically impossible for a human to write a complete program in machine code, writing programs in assembly language is viable. Because assembly language is extremely low-level, these programs tend to run quickly, but they are labor-intensive to write, and are machine-dependent. This type of programming was sufficient in the 1960s, 1970s, and 1980s when software was written for one-off capital-intensive machines, such as multimillion-dollar mainframes or space vehicles. Programmer labor was comparatively cheap then, and there was no need to move programs to different hardware. But today, we expect applications to be compatible with multiple kinds of platforms, including phone, computer, and gaming systems. Programmer labor is more expensive than computer hardware, so writing entire programs in assembly language is uneconomical. Consequently, programs are often written predominantly in a high-level language, with assembly language used to write short excerpts on an as-needed basis. Writing code in a higher level language makes it easier to write correct code that does not have defects.

LINK TO LEARNING

Assembly language has been used in high-profile, high-budget projects, such as Apollo 11, the NASA spaceflight that first landed humans on the moon. You can examine the assembly code for the embedded computers (https://openstax.org/r/76AssemblyCode) in the space vehicles, which has been released publicly. Notice how it is quite low-level, perhaps difficult to follow, and reflects an immense amount of fastidious work.

Middle-Level and High-Level Programming

As the name implies, a **middle-level programming language** is at a level of abstraction in between low-level and high-level language, and allows for direct hardware access. The **C** programming language is a middle-level language that has been in wide use since the 1970s. The **C++** programming language is a middle-level object-oriented language based on C. In general, the trade-off between low-level and high-level languages is that writing low-level code is laborious and error-prone, but the finished code executes very quickly; high-level code is faster, easier, and safer to write but does not run quite as quickly. Middle-level code is a compromise that executes nearly as fast as low-level code yet has some of the productivity benefits of high-level code. Like low-level languages, middle-level languages allow direct access to computer hardware, making it possible to write hardware-specific programs such as operating systems and device drivers. An **operating system** is the software that provides a platform for applications and manages hardware components. A **device driver** is a piece of code responsible for connecting to a hardware component, such as a video card or keyboard. Figure 4.8 summarizes the trade-offs between low-level, middle-level, and high-level programming languages.

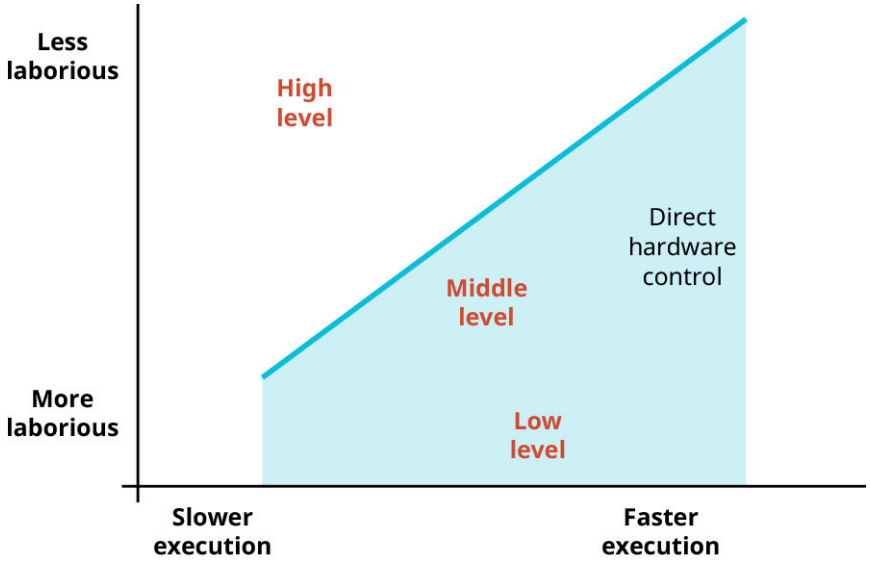


Figure 4.8 As a rule, high-level languages are less laborious to write, and slower to execute, than low-level languages. High-level languages typically do not support direct hardware control. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Middle-level languages are ideally suited to writing **systems software**, programs that provide infrastructure and platforms that other programs rely upon. The core part of an operating system that is responsible for managing and interfacing with hardware components is called the **kernel**. Kernels need direct hardware access, so high-level languages are inadequate. Practically all widespread kernels are written in C and/or C++ (such as Windows, MacOS, Linux, iOS, Android, Xbox, and PlayStation). Compilers for high-level languages, such as Python, Java, and C#, are themselves implemented in middle-level languages such as C.

Figure 4.9 summarizes the various types of programming languages and how middle-level languages overlap with low-level and high-level programming languages.

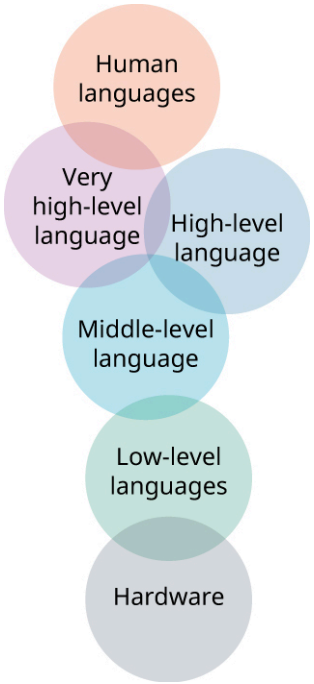


Figure 4.9 Computation models fit onto a spectrum from low to high level. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

A high-level language is farther from a hardware model, and closer to an abstract model. Source code in a

high-level language does not address low-level details, and instead focuses on how an algorithm proceeds, such as "visit every item in a list." A high-level language is like viewing Earth at a high altitude, revealing large features such as the contours of rivers and highways, whereas a low-level language is like viewing Earth at ground level, which allows for focusing on fine details such as the activity of individual people and animals.

Web application frameworks (e.g., React, Node) are written in high-level languages, principally JavaScript. A web framework is a special tool for building and managing web applications. Some common ones used in web clients are HTML5, CSS, and JavaScript. Native Android apps are primarily written in the high-level language Java, and iOS apps are primarily written in the high-level language Swift.

Programming Language Paradigms

So far, we have categorized programming languages according to their level of abstraction into low-level, middle-level, and high-level languages. A different approach is to categorize languages into paradigms. A **programming language paradigm** is a philosophy and approach for organizing code, the ideas in a program, and the layout of its source code. Real-world programs involve many thousands of lines of source code, which is too much for a human to digest without some kind of organizational structure. Computer scientists have developed several different paradigms for creating this structure.

Unlike level of abstraction, paradigms do not fall on a spectrum. Instead, a particular programming language either adheres to the philosophy of a paradigm, or it does not. For example, C is a structured procedural language and not an object-oriented language. Without getting into too many details, C is procedural because it allows programmers to place code in functions that can be called from various places in a program. However, C is not object-oriented because C does not allow, like Java does, the creation of objects that are instances of classes. We will broadly explore these different types of paradigms later in this section, but the chapter on Chapter 7 High-Level Programming Languages elaborates on these and other paradigms in more detail.

The Imperative Programming Paradigm

In **imperative programming**, the programmer writes a series of steps that must be followed in order. Source code spells out a precise series of operations that the computer must execute in order. Since the computer is told to take specific actions and execute these statements, the language is referred to as "imperative." An imperative is an order or command. Low-level languages are imperative languages, and middle-level languages, such as C, are imperative and include another paradigm. While low-level languages can mimic the style of a structured language, these properties are not inherent in the language itself and must be imposed by the programmer as a practice. Assembly code can easily be written in a non-structured way.

Declarative and Functional Programming

Another type of programming, **declarative programming**, is a paradigm in which code dictates a desired outcome without specifying how that outcome is achieved. Declarative languages are an alternative to imperative languages. In a declarative language, the programmer declares the desired outcome, and it is the compiler's job to create a series of imperative steps that obtains that outcome. For example, the Structured Query Language (SQL) used to query database systems makes it possible to specify what data should be retrieved from a database, without specifying how the database system should retrieve that data.

Functional languages are another alternative to imperative languages. Recall that a function is a mathematical object that defines how to convert an input into an output. For example, given $x = 4$, the function $f(x) = x + 3$ converts the input 4 into the output 7. Functions can be defined in most programming languages and correspond to small sections of code that perform a specific task such as a calculation. Functions can be defined in most programming languages.

Functional programming is a programming paradigm in which algorithms are written as mathematical functions (Figure 4.10). In functional programming, practically every part of the program is written as a function. The programmer writes functions that convert inputs to outputs, and it is the compiler's job to create

imperative steps to evaluate the functions.

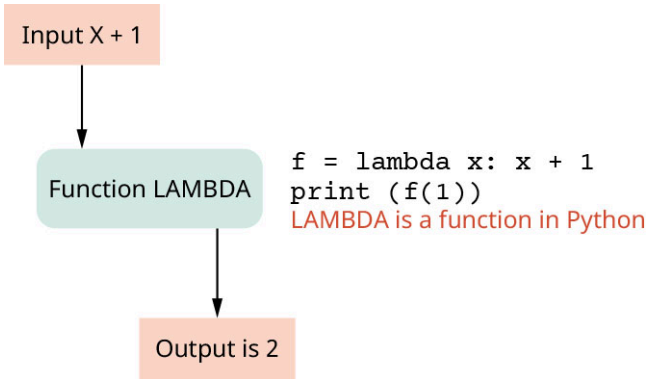


Figure 4.10 This diagram shows a functional programming example using Python. Here the function "LAMBDA" is used to increment the value of *x* by 1 whenever it is called. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Declarative and functional languages are considered high-level languages because the compiler is creating these steps on behalf of the programmer. Functional languages are discussed in more detail in Chapter 7 High-Level Programming Languages.

Structured Programming

In low-level languages and early high-level languages such as **BASIC**, some special statements called conditional statements (using "if/then") and iteration (called "loops") are programmed using an operation called **GOTO**, a non-structured operation that instructs a computer to jump to an entirely different part of the program. In large programs, these jumps from one spot to another interact in complex ways, so the flow of execution is difficult to understand when attempting to read the code. These sorts of programs are criticized for being messy "spaghetti code" (Figure 4.11).

Structured:

```
IF x<=y THEN
   BEGIN
      z := y-x;
      q := SQRT(z);
   END
ELSE
   BEGIN
      z := y-x;
      q := -SQRT(z);
   END;
WRITELN(z,q);
```

Unstructured:

```
IF x<=y THEN GOTO 2;
z := y-x;
q := SQRT(z);
GOTO 1;
2: z:= y-x;
q:=-SQRT(z);
1: writeln(z,q);
```

Figure 4.11 The same program that prints the square root of an integer (using SQRT(*z*)) is shown in a structured format and an unstructured format. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Newer languages were developed to help avoid spaghetti code. In a **structured programming** language, control flow leverages conditionals (e.g., "if-then") and iteration statements (e.g., "while" or "do while") and never uses GOTO statements. For example, C is a structured language that includes the conditional statements "if" and "switch," and the iteration statements "for," "while," and "do." In proper C, all the code sections that involve conditionals and iteration are written with these statements, and GOTO should not be used. Note that the fact that GOTO is provided as a keyword in the C language relates to the fact that C is a low-level language and programmers at that level are given the choice of using unstructured programming if necessary, although it is not recommended.

LINK TO LEARNING

Read this seminal article about how GOTO statements can be considered harmful (https://openstax.org/r/76GOTOStatemnts) written by Edgar Dijkstra.

The benefit of using these statements is that they make the flow of execution clear in the source code. When writing an "if," for instance, it is clear which code is inside the "if" and which is outside. And when mixing an "if" with a "for" loop, it is clear whether the "if" is inside the "while" or vice-versa. These sorts of inside/outside relationships are difficult to perceive in unstructured code. In a conditional statement like "if", the compiler executes a line if the condition has been met or is true. Otherwise, it moves to the next statement:

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.println("Enter an input value: ");
        int Val = s.nextInt();
        int Curr_Val = 10;

        if (Val > Curr_Val) {
            System.out.println("The Value that you entered is greater than the
current.");
        } else {
            System.out.println("The Current value is greater than the value that you
entered.");
        }

    }
}
```

Inside a loop, like "while", the statements are executed only if the condition in the loop is true. Otherwise, the loop execution terminates, and the compiler moves to the statements after the loop:

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.println("Enter an input value: ");
        int Val = s.nextInt();
        int Prod = 1;

        while (Val != 0) {
            Prod = Prod * Val;
            Val--;
        }

        System.out.println("The factorial is " + Prod);
```

```
    }
}
```

There is a substantial upside to making a language structured, and the only significant downside is that it makes the language a bit more high-level. Therefore, among programming languages that are currently in widespread use, all the middle-level and high-level languages are structured.

Procedural Programming

In a procedural language, each part of the program is a **procedure**, which is a function in the context of programming. Known as **procedural programming**, this is a paradigm in which code is organized into procedures (Figure 4.12). It is a sub-type of imperative programming. All procedural languages, then, are imperative, but not all imperative languages are procedural. A programmer designs each procedure to accomplish a specific task and gives it a descriptive name. This allows the programmer to break a large and complicated program into smaller, more manageable pieces, which are easier to write and easier for other programmers to understand. This property of code being divided into small, reusable piece is called **modularity**, and it is considered a virtue.

Procedural Programming

1. Take 2 pieces of bread.
2. Separate the pieces of bread.
3. Get a knife.
4. Get some peanut butter and jelly.
5. Take the knife and put peanut butter on it.
6. Spread the peanut butter on the bread with the knife.
7. Take the knife and put some jelly on it.
8. Spread the jelly on the other slice of the bread.
9. Take the slice of bread with the peanut butter and the slice of bread with the jelly and put the two together.

Procedural Programming in C++

```
cout << “Enter the Length:”,
cin >> Length;
cout << “Enter the Width:”,
cin >> Width;
Area = Length * Width;
Cout << “The area is:
<<Area << endl;
```

Figure 4.12 This procedural programming comparison relates a real-life scenario, such as making a sandwich, to the corresponding C++ version of the same scenario. In both cases, each step has its own procedure. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

For example, in C, a procedure that opens a **socket**, or an Internet connection between two computers, is called `g_socket_connect()`(Figure 4.13). This procedure involves executing a series of imperative commands that use operating system features (such as the transport layer), system calls, and networking hardware (such as network interface cards, or NICs), to set up a socket connection. To close that connection and end communication, C uses the `g_socket_close()` procedure, which executes a series of commands that shut down the connection. These procedures use the imperative approach to accomplish their respective tasks, so each function contains a series of imperative statements.

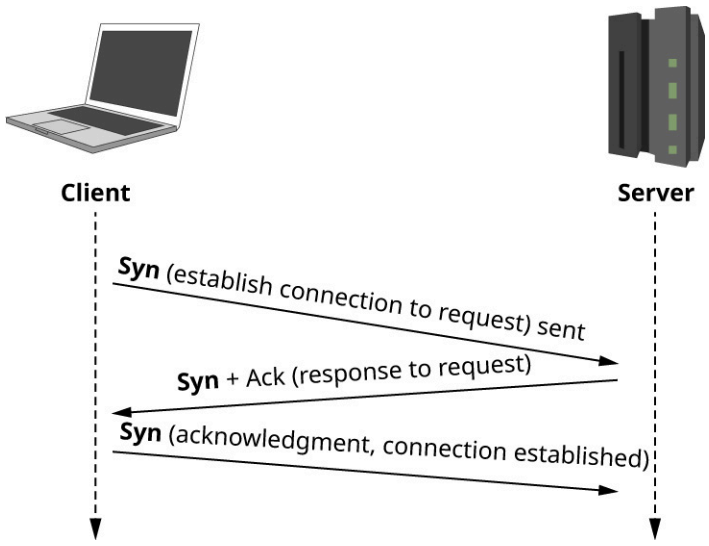


Figure 4.13 This diagram demonstrates various functions used by the socket on both the client and the server side to establish and close a connection. (credit: modification of "Tcp connect" by Sébastien Koechlin/Wikimedia Commons, CC BY 3.0)

A procedural programming language provides syntax for defining procedures but cannot force individual programmers to follow through with breaking their code up into small procedures and giving the procedures descriptive names. So, even when a program is written in a procedural language, the source code may not necessarily be written in a procedural style.

Object-Oriented Programming

Object-oriented code is organized around objects. An **object** has both data, or variables, and procedures that work together to represent a specific human concept. For example, in a desktop or mobile application, every button on the screen is an object. Each button has variables to represent information, such as the location and color of the button, and procedures that perform tasks, such as clicking, hiding, or displaying the button. This programming paradigm is known as **object-oriented programming**. It is a programming paradigm in which code is organized into objects, where each object has both data and procedures. It is a sub-type of procedural programming. All object-oriented languages, then, are procedural (and by extension, imperative), but not all procedural languages are object-oriented. A simple example of an object can be a rectangle used to represent meaningful concepts in real life, such as the rooms in a house or a person or robot and what it can do. Different rooms may have different attributes, representing features that are specific to a kitchen, a living room, or a bedroom. A robot can have a name and age and can receive input commands and respond or print a greeting, as illustrated in Figure 4.14.

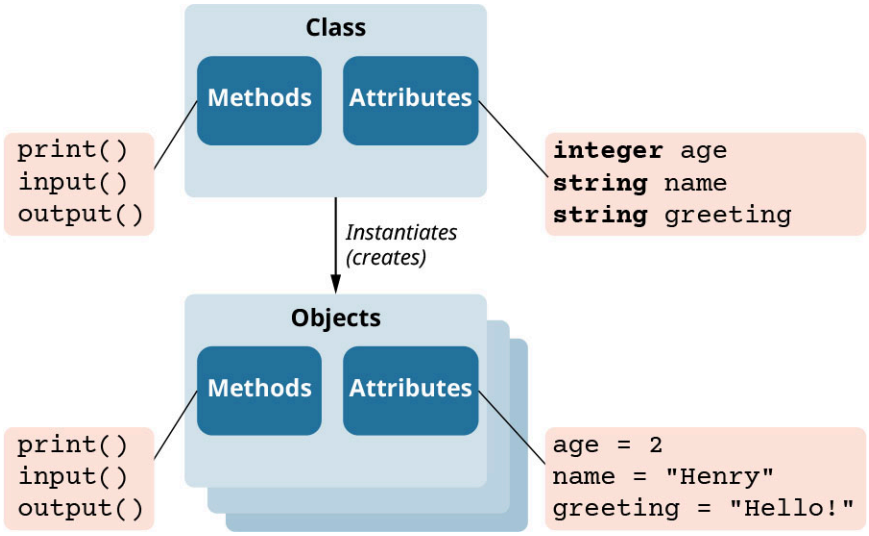


Figure 4.14 The object-oriented use of a robot class instantiates robot objects that have attributes of age, name, and greeting and methods of `input()`, `output()`, and `print()`. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Object-oriented programming was invented to help programmers organize their code, and it has been very successful. The most widely used high-level languages, including C++, C#, Java, JavaScript, and Swift, are all object-oriented. Object-oriented programming is discussed further in Chapter 7 High-Level Programming Languages.

4.2 Building C Programs

Learning Objectives

By the end of this section, you will be able to:

- Write C code using fundamental elements of the language
- Summarize the steps to develop a C program
- Understand the process to compile and run a C program
- Describe how linking is used in a C program
- Understand how to apply version control management

As discussed, a programming language is a kind of computational model that is used to write programs. C is a popular middle-level language that is widely used to create systems software. This section is a crash course in the essentials of C.

Introduction to C

The C programming language was invented in 1972 by Dennis Ritchie of Bell Labs (Figure 4.15) and popularized by the book *The C Programming Language* by Brian Kernighan and Dennis Ritchie. C's peculiar name—a single letter—was a pragmatic choice, since C replaced an earlier language named B. C is a procedural, middle-level language that gives programs low-level access to memory. It is a relatively simple language, which makes learning C, and creating a C compiler, easier than for more complex languages. This combination of features made C an instant hit, and it has maintained great popularity and import to this day. C has influenced other programming languages, too. C++ is a newer language that adds the object-oriented paradigm to C.

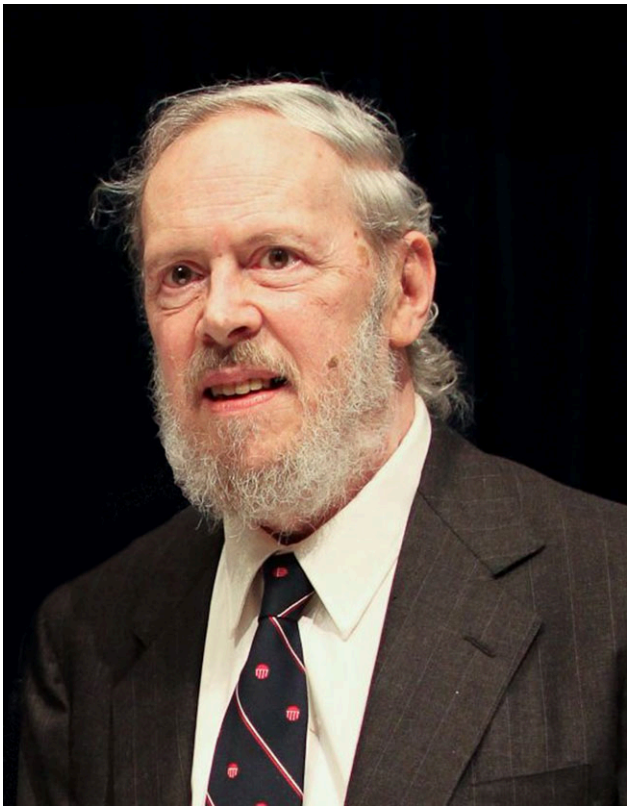


Figure 4.15 Dennis Ritchie of Bell Labs invented the C programming language. (credit: "Dennis Ritchie 2011" by Denise Panyik-Dale/ Wikimedia Commons, CC BY 2.0)

Why is C so popular? Mainly because its designers managed to strike a balance between low-level and middle-level features that allows C code to execute at practically the same speed as assembly language, while allowing programmers to be productive enough to create large, dependable, programs. C is the programming language behind much of the lower-level software that we depend on, including operating systems, language compilers, assemblers, text editors, print servers, network drivers, language interpreters, and command-line utilities. Here are some specific software products that are written in C:

- The Java virtual machine (ANSI C)
- **Linux**, an open-source operating system (C, and some assembly)
- Python (C)
- macOS X kernel (C)
- Windows (C, C++)
- The Oracle database (C, C++)
- Cisco routers (C)

INDUSTRY SPOTLIGHT

Applications of C

C is used in a variety of industries. One example is astrophysics, where scientists write programs that simulate the motion of stellar bodies, and control instruments such as telescopes. Owing to the large size of the universe, these simulations involve performing calculations on very large arrays of numbers. C's ability to execute fast, and control the layout of large arrays in memory, is advantageous for this application. As a relatively simple language, C is approachable to physicists who are not necessarily expert in computer science. Scientific experiments need to be reproducible, which means that code involved in science needs to work even decades in the future. The fact that C has been a stable, popular language for

so long means that it is very likely to endure, which cannot be said of newer niche languages.

One notable feature of C is the way it handle memory. In a program, we have variables and values. For example, in $x = 10$, x is a variable and 10 is the value. Every value in a program is stored in memory. Memory regions are divided into four blocks: stack, heap, static, and code blocks. These regions store various parts of a running program. Running programs create and destroy values extremely rapidly (perhaps millions or billions per second), and memory is finite, so memory locations must be reused, or else would run out quickly. When a value is created, memory is set aside as **allocated memory** to hold that value. Eventually, when the value is no longer needed, that memory becomes **freed memory**, meaning it is given back so that it can be reused. The process of allocating and freeing memory is called **memory management**. A **memory leak** happens when some memory is allocated but never freed. A memory leak is a bug that causes a program to waste resources; severe leaks can waste all the memory on the computer, causing it to become unresponsive or crash. As a middle-level programming language, C requires programmers to handle memory management manually. This type of flexibility must be used with caution as it may result in creating programs that are not reliable and secure. In high-level languages, memory management is automated.

Here are some other notable features of C:

1. Efficient execution: C is lower in expressive power than some other middle-level languages like C++ and yet simple enough that compilers can generate machine code that is comparable in speed to hand-written assembly code. A lot of research and development have focused on creating performance-oriented C compilers.
2. Portability: C can run in multiple computing environments, also known as having the property of portability. Unix was designed to work on various hardware architectures, so the C language is not hardware-dependent. The same C code can be compiled and executed on different hardware architectures and operating systems.
3. Modularity: Modular programming refers to the process of dividing computer programs into separate sub-programs. A module is a separate software component, such as an error handler, that may be used by a variety of applications and functions within a system. C has language support for modularity.
4. Procedural and structured programming support: C adheres to the procedural and structured paradigms.
5. Data types and operators: Every variable in a C program has a data type. Data types dictate how much memory is used to store the variable, and which kinds of operators can be used with the variable.
6. Recursion support: Recursion is the phenomenon of a system being defined in terms of itself. In code, this means a function may call itself again and again. C supports recursion. However, it does not provide a feature called “tail-calling” that makes recursion efficient, so recursion is not used in C as much as in languages that provide tail-calling. A tail call is a function call performed as the final action of a function. If the target function of a tail is the same function, the function is said to be tail recursive, which is a special case of recursion. Tail recursion (also called “tail-end recursion”) is useful and helps with code optimizations.
7. Pointers: A **pointer** is a variable that holds the memory address of another variable and points to that variable (Figure 4.16). Pointers play a crucial role in the C language. They are used to store and manage addresses of dynamically allocated blocks in memory in the underlying computer system. Managing hardware devices involves manipulating certain memory locations, and C’s support for pointers is one of the reasons that it is used to implement kernels and device drivers.

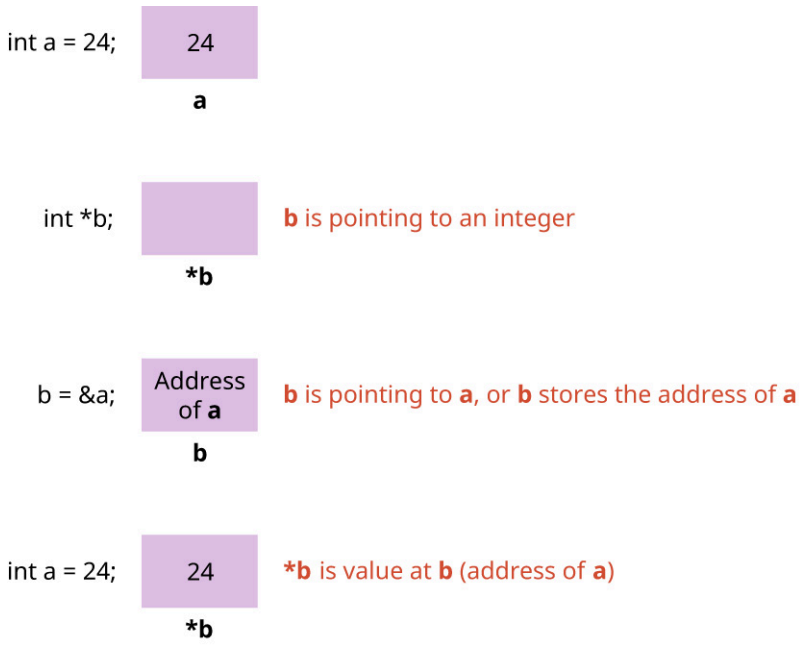


Figure 4.16 A pointer is a variable whose value is another variable's address in memory. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

TECHNOLOGY IN EVERYDAY LIFE

C's Application in the Early Stages of YouTube

C has a variety of integer types, such as `short`, `int`, and `long`. A C programmer needs to decide on the most appropriate type for each piece of information in their program. Smaller types use less memory but can only store a narrower range of values. On a typical computer, the maximum `short` value is about 32 thousand and the maximum `int` value is about 2 billion. A good practice is to think critically about how large a particular value might become and pick the smallest data type that accommodates that range.

The YouTube programmers faced this issue when they implemented the view counter for YouTube videos. They had to think through: what is the maximum number of views that a video is likely to garner? Two billion seemed like a safe choice, so they chose `int`.

This decision turned out to be misguided. In 2014, the viral hit music video "Gangnam Style" by the Korean artist Psy accumulated more than two billion views, and the view counter broke. The `int` variable storing the number of views of "Gangnam Style" overflowed and wrapped around to a negative number. This proved to be an embarrassment for YouTube, who had to quickly change their code to use `long` instead.

What is the most appropriate integer data type (`short`, `int`, or `long`) for the following quantities?

1. The number of people on an airplane
2. The number of people on Earth
3. The number of people in a household
4. The number of dollars in a bank account

High-level languages usually check array indices at runtime, which makes out-of-range bugs easy to identify and fix, but slows down array subscripts slightly. As a middle-level language, C does not check array indices. An array is a storage space where the elements are stored in contiguous memory cells. They are indexed from 0 (the first cell) to n−1 (last cell).

In C, an invalid array subscript will access memory outside of the array variable. If the subscript is only out of range by a little bit, this will access nearby variables, which is a subtle bug that may go unnoticed. A **segmentation fault** ("segfault" for short) occurs if the subscript is very far out of range. When this occurs, it will access a memory address that is off-limits to the program, and your operating system will forcibly shut down the program in response. This kind of runtime error can be notoriously difficult to remedy. Out-of-bounds array subscripts are a common source of segmentation fault errors.

Every value in a program is stored at a specific memory address. A pointer is a value that contains a memory address. Technically, a pointer should contain the location of a valid data value. However, many memory locations do not contain valid data values, so it is possible to have an **invalid pointer** that does not hold a valid location. The pointee is the value that a pointer points at. A pointer is analogous to a street address such as "123 Main Street," because it refers to a specific location. In that analogy, each building is a pointee. Usually, an address is valid and refers to a place you can visit. However, it is possible to have an invalid address that is not a place that can be visited; for example, if the building at that location was demolished.

One of the differences between middle-level and high-level languages is that high-level languages either prohibit invalid pointers entirely, or provide mechanisms to handle them safely. As a middle-level language, C gives programmers the freedom to create null/invalid pointers, which can be helpful when writing code that interfaces with hardware devices. Since all hardware devices do not support the same functionality, the support of individual features by a given device may be indicated as a null/uninitialized pointer, which is fine as long as the program checks for un-initialized pointers to determine if a given functionality is available. However, in general, the freedom of using null/invalid pointers comes with a responsibility to ensure that pointers are always used properly. This has proven to be difficult; invalid pointers are a common source of bugs in C programs.

In C, the programmer is responsible for making sure that character arrays are actually big enough to fit strings, and that strings include the null terminator character. A character array is a string of characters sometimes terminated using a null. An example might be something like: `char *arr= "string\0"`. Overlooking either of these results in bugs. This is a prime example of how middle-level languages such as C expect programmers to deal with more details than do high-level languages.

LINK TO LEARNING

The C standard library has dozens of header files and hundreds of functions. It is impractical to memorize all this information. Programmers do not memorize the prototypes (i.e., name and parameters) of library functions. Instead, they refer to reference documents, and develop the skill of finding information in these documents quickly. These C library reference documents (https://openstax.org/r/76CLibraryDocs) are available in many places.

Developing C Programs

A programmer spends significant time working in their development environment; indeed, a professional developer might spend most of their workday using it. It pays to invest some up-front time and attention toward learning your environment and customizing it to your needs so that your ongoing experience will be frictionless and ergonomic. Chefs, mechanics, and other tradespeople focus much attention on cultivating safe and productive workspaces, and in the same way, experienced programmers attend to their development environment.

Programmers working with compiled languages, including C, generally work using the cycle shown in Figure 4.17.

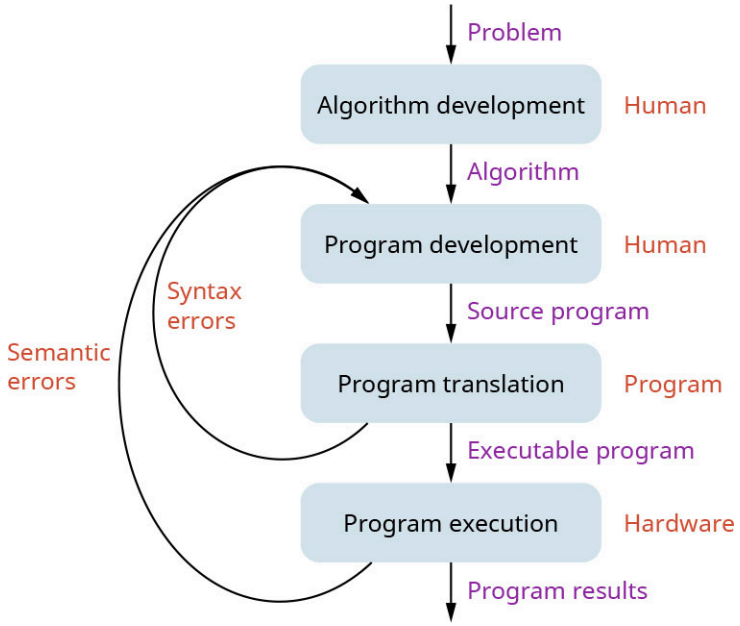


Figure 4.17 A typical work cycle includes multiple compilation steps after a program has been written and is ready for compilation. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Specifically, these steps are:

1. Algorithm Development: The developer designs a high-level understanding of what the code will do and how it will do it. They may document the algorithm with pseudocode, a block diagram, or a sketch. In the case of extremely simple programs, the algorithm may be trivial enough that the programmer can keep it in their head. In some cases, a programmer is implementing an algorithm that someone else created and described in a reference work or research paper.
2. Program Development: The programmer writes code that implements the steps of the algorithm.
3. Program Translation: The programmer runs the compiler on the code. Often, the code has syntax errors, and the compiler provides error messages describing the errors. A syntax error is a violation of the rules for constructing valid statements in the language. For example, the user may have introduced a typo of some sort, like a missing semicolon, or using a keyword as a variable name. In this case, the programmer goes back to Step 2 (Program Development) to resolve the errors one by one.
4. Program Execution: At this step, the code has no syntax errors, so it successfully compiled into a runnable program. The developer runs the program, and tests that it operates properly. An initial draft of code often has a **semantic error**, which is when code compiles and runs, but does not behave as it should. When a programmer finds a semantic error, they go back to Step 2 to debug the code and fix the semantic error. Eventually, after thorough testing, which requires a specific approach not described here, no more semantic errors can be found, and the code is considered finished.

Some C compilers include:

- **GCC**, an open-source C compiler developed by the GNU Project
- **Clang**, an open-source C compiler developed by the LLVM project
- **Visual C++**, a C and C++ compiler developed by Microsoft

Depending on which operating system you are using, there will be many viable alternative C development environments. An operating system is a complex software program that helps the user control the hardware and help with several other applications. Examples include Windows 10 and 11, and Linux versions such as Ubuntu, Fedora, CentOS. You may choose to use an **integrated development environment (IDE)**, which is a program with a graphical user interface that includes a text editor, compiler, and other tools, all in one application. For example, you can install and use Eclipse for C/C++, an open-source multi-language IDE

originally created for Java programming. Eclipse is portable as it is built in Java and can be installed on any operating system.

Compiling and Running C Programs

The compilation process involves several steps:

- compiler: high-level language converts to assembly
- assembler: assembly converts to machine code
- **linker**: a program that performs **linking**, a process of collecting and combining various pieces of object code into a single program file that can be loaded into memory and executed

In practice, compilers such as GCC bundle all these steps into one command. Usually, when you run the GCC command, GCC compiles, assembles, and links a program.

To write, compile, and run a simple C program:

1. Write text of program (i.e., source code) using a text editor, and save it as a text file (e.g., "my_program.c")
2. Run the compiler, assembler, and linker to convert your program from source to an "executable" or "binary." Compilation is necessary for every program to run and perform the desired operation.
   `$ gcc –Wall –g –o my_program my_program.c`
   GCC compiler options:
   - `-Wall` tells the compiler to generate all "warnings." These warnings will often identify mistakes.
   - `-g` tells the compiler to generate debugging information.
   - If you don't supply a –o option to set an output filename, it will create an executable called `a.out`.
   - A `.c` file is called a "module." Often programs are composed of multiple `.c` files and libraries that are linked together during the compilation process.
3. If the compiler gives errors and warnings, edit the source file, fix it, and recompile. It is a good practice to work on just one error/warning at a time, namely the first one. This is because a syntax error can cause false-alarm errors later in the source code, so warnings/errors after the first one could be false alarms. We recommend that, when you get compile errors or warnings, you edit to fix just the first one, and recompile; do not try to fix warnings/errors after the first one.

Consider the following "Hello World" C program[1]:

```
#include <stdio.h>      /* include printf prototype */
/* The simplest C Program */
int main(int argc, char **argv)      /* main program entry point */ {
    printf("Hello World\n");
    return 0;      /* return without error */
}
```

To run a program in the current directory (on Linux) use `./program .` ("." means the current directory). In the world of operating systems, everything is defined in terms of directories and files. Even the desktop is a directory, which is a collection of files. A directory can sometimes be empty too, and some directories have hidden files for security reasons. A subdirectory is a directory within a directory.

```
> ./my_program
Hello World
>
```

1 **argv means that the program is accepting a multidimensional array of input arguments. It is a pointer to the pointer of array of arguments.

Linking Programs

Figure 4.18 illustrates the processing steps of C programs from source code to execution.

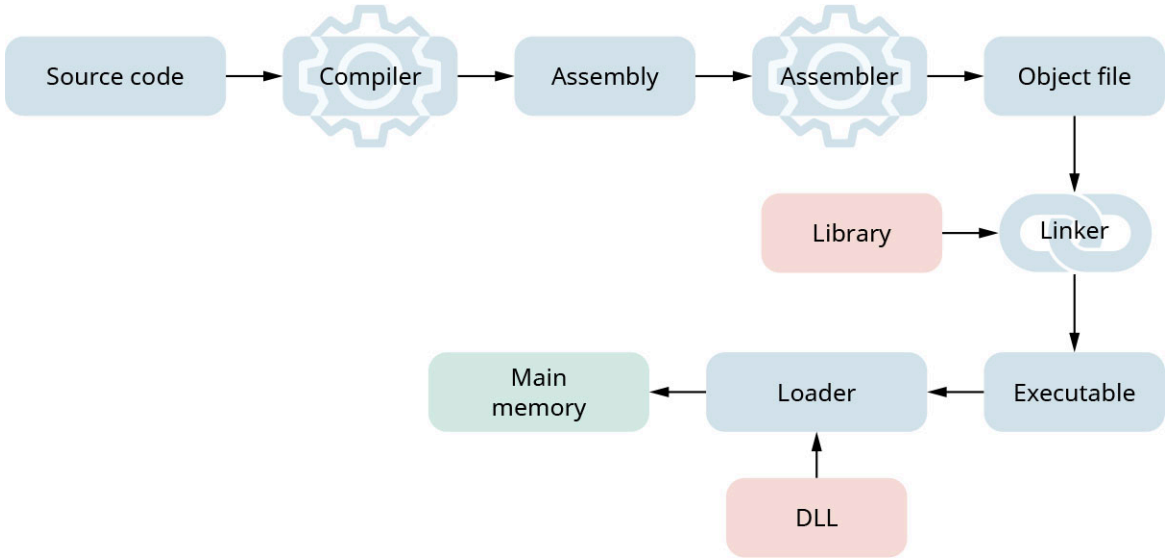


Figure 4.18 The linking process that is used by languages to make them portable requires a number of different steps. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Linking refers to the process of collecting and combining various pieces of object code into a single program file that can be loaded into memory and executed. A linker is a program that performs linking. Understanding linkers will help you build large programs, avoid dangerous programming errors, understand how language scoping rules are implemented, understand other important systems concepts (such as virtual memory and paging), and use shared libraries (a file that is to be shared by an executable file). Virtual memory is an operating system concept where the secondary memory acts as main memory to compensate for memory shortage. Paging is a technique where the secondary memory is used to store and retrieve the data into the main memory. The memory is divided into small regions called pages which enables for the quick access of the data. If a page is found, it is called a “Page hit;” otherwise, it is a “Page miss.”

Programs are translated and linked using a compiler driver, a program that invokes other components that helps in translating the high-level program to a machine code, as in Figure 4.19 and using the following code:

```
linux> gcc -Og -o prog main.c sum.c
linux> ./prog
```

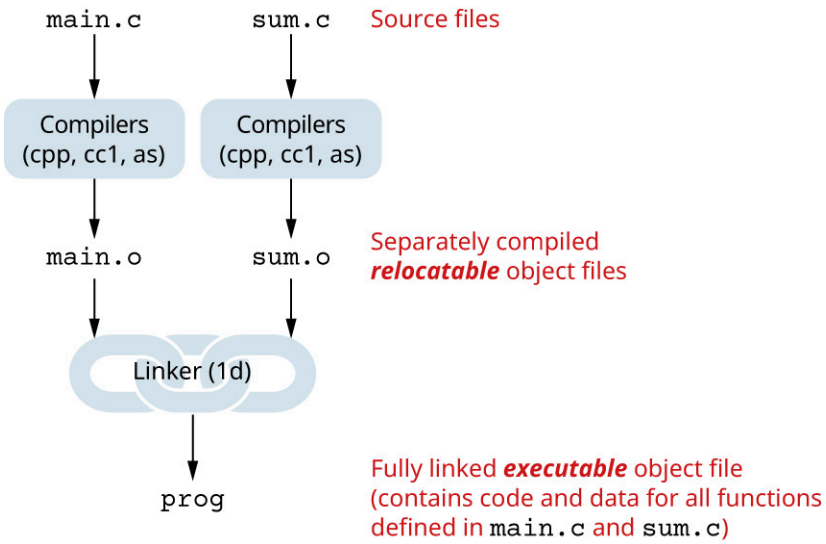


Figure 4.19 Source files and separately compiled relocatable object files can be linked into an executable object file. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Linkers are used to ensure:

- Modularity: Program can be written as a collection of smaller source files, rather than one monolithic mass. Using a linker facilitates building libraries of common functions (e.g., Math library, standard C library). A **library** is a file that contains object code (a code from the object file that is generated after compilation) for functions and global variables (variables that have global scope and can be used anywhere in the program) that are intended to be reused.
- Efficiency: It saves time to run separate compilations and change one source file, compile, and then relink since there is no need to recompile other source files. Also, libraries save memory space because common functions can be aggregated into a single file and yet executable files (the end product after compiling and linking) and running memory images (current memory) contain only code for the functions they actually use.

Linking Steps

Programs define and reference symbol. A **symbol** is an identifier for a function or a global variable. The first linking step performs **symbol resolution**. During the symbol resolution step, the linker associates each symbol reference with exactly one symbol definition (Figure 4.20).

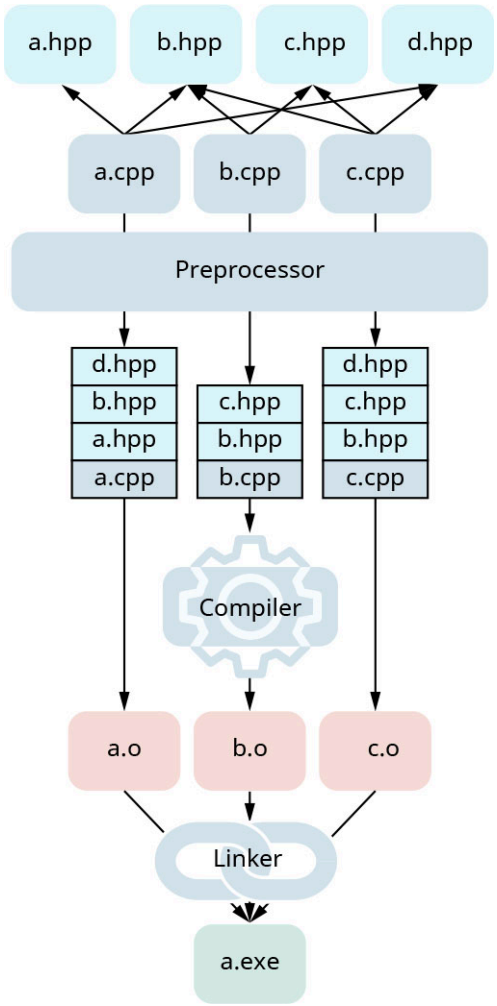


Figure 4.20 In the process of compilation and linking in C++, ".hpp" are the header files, ".cpp" are the actual C++ programs, ".o" are the object files, and ".exe" is the executable. (credit: modification of "C++ compilation process" by "Prog"/Wikimedia Commons, CC0 1.0)

Symbol definitions are stored in an object file (by the assembler) called a **symbol table**. A symbol table is an array of structures in which each entry includes name, size, and location of symbol.

The second linking step performs **code relocation** (Figure 4.21). This step merges separate code and data sections into single sections (one for code and one for data). It relocates symbols from their relative locations in the `.o` files (the object files) to their final absolute memory locations in the executable. It updates all references to these symbols to reflect their new positions.

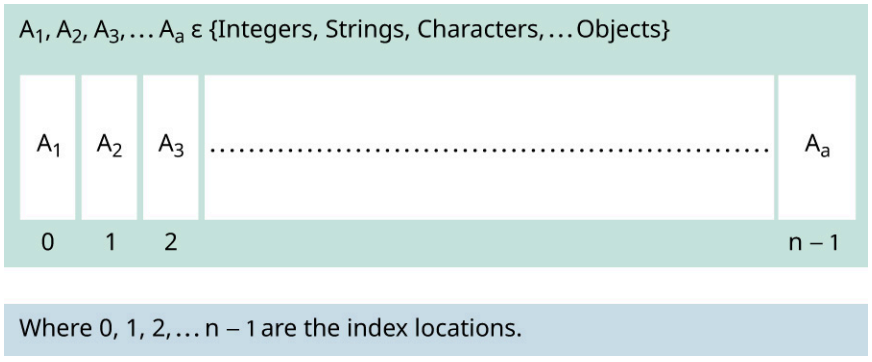


Figure 4.21 This diagram shows the input types in an array and the index locations. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Executable and Linkable Module Format

There are three kinds of object files (modules) that relate to the linking process (Figure 4.22):

- Relocatable Object File (`.o` file): Contains code and data in a form that can be combined with other relocatable object files to form executable object file. Each `.o` file is produced from exactly one source (`.c`) file.
- Executable Object File (`a.out` file): Contains code and data in a form that can be copied directly into memory and then executed.
- Shared Object File (`.so` file): Special type of relocatable object file that can be loaded into memory and linked dynamically, at either load time or runtime. These object files are called Dynamic Link Libraries (DLLs) on Windows.

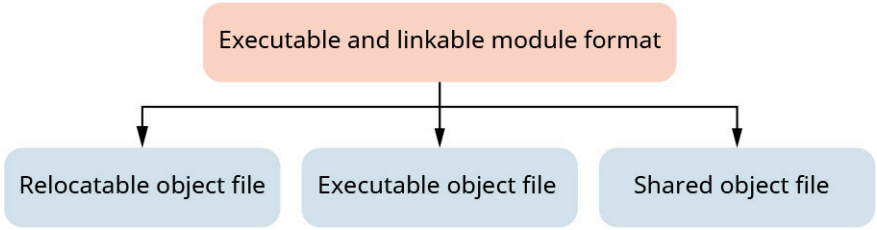


Figure 4.22 The subcategories of Executable and Linkable modules are arranged in a hierarchy. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

All three object files follow the **executable and linkable format (ELF)** which is a standard binary format for object files originally proposed by AT&T System V Unix, and later adopted by BSD Unix variants and Linux. **Unix** is an operating system that has been used widely, primarily in servers and software development since the 1970s, and Linux is Unix-compatible.

Symbol Types and Resolution

A linker classifies symbols in three categories as illustrated in Figure 4.23 and Figure 4.24. A symbol can be the name of a variable or a string. In other cases, it can be the function names or procedure, such as

- Global symbols: Symbols defined by module *m* that can be referenced by other modules (e.g., non-static C functions and non-static global variables)
- External symbols: Global symbols that are referenced by module *m* but defined by some other module.
- Local symbols*:* Symbols that are defined and referenced exclusively by module m (e.g., C functions and global variables defined with the static attribute); local linker symbols are *not* local program variables (linker does not deal with the local variables of a function). Also note that local non-static C variables are stored on the stack while local static C variables are stored in either .bss, or .data.

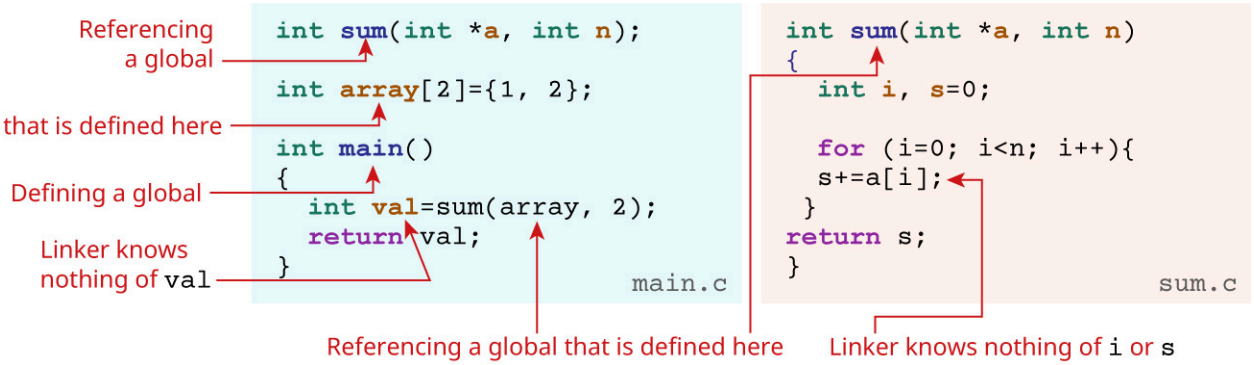


Figure 4.23 The code shown illustrates how the linker identifies local and global symbols. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

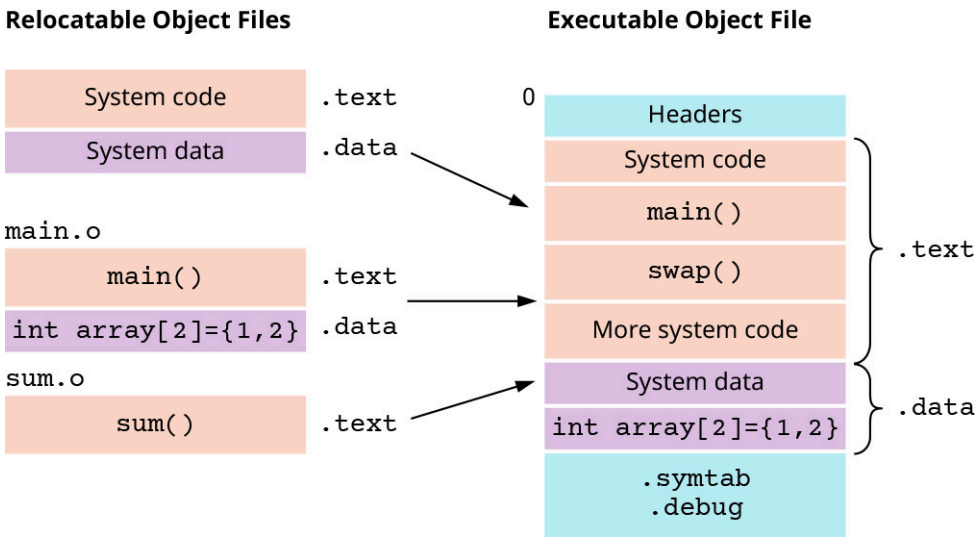


Figure 4.24 This diagram illustrates how various symbols are organized in .text and .data segments within relocatable object files and are mapped into the .text and .data segment by the linker to create executable object files. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Program symbols are either strong (e.g., procedures and initialized globals) or weak (e.g., uninitialized globals). A strong symbol has a unique memory location. Let's take the example of: `int array[2] = {1 ,2};`. This creates an ambiguity during the linking process when there is another file that tries to access the same symbol again due to strong definition. On the other hand, a weak symbol allows multiple definitions of the same symbol without creating an ambiguity. This helps during the linking process when another file creates a strong definition of the same name. In languages like C and C++, the weak symbol is defined using the `__attribute__((weak))` keyword. Global variables should be avoided (i.e., use static whenever you can, initialize the global variable, or use extern if you reference an external global variable).

The linker applies the following rules:

- Rule 1: Multiple strong symbols are not allowed. Each item can be defined only once, otherwise the linker issues an error.
- Rule 2: Given a strong symbol and multiple weak symbols, choose the strong symbol (references to the weak symbol resolve to the strong symbol).
- Rule 3: If there are multiple weak symbols, pick an arbitrary one (can override this with `gcc –fno-common`). "-fno-common" helps in catching accidental common name collisions.

Static Libraries

Functions commonly used by programmers (e.g., math, I/O, memory management, string manipulation) can be packaged into a file called a library. A **static library** (or `.a`, an **archive file**) is a simple kind of library that that copies the contents of object files into a single file called an *archive*. The linker tries to resolve unresolved **external references** by looking for the symbols in one or more archives. An external reference is a symbol that is used in a module, but not defined in that module, so it is expected to be defined in some other module. If an archive member file resolves a reference, the linker links it into the executable. The archiver allows incremental updates; it also recompiles functions that changed and replaces the corresponding `.o` file in the archive (Figure 4.25).

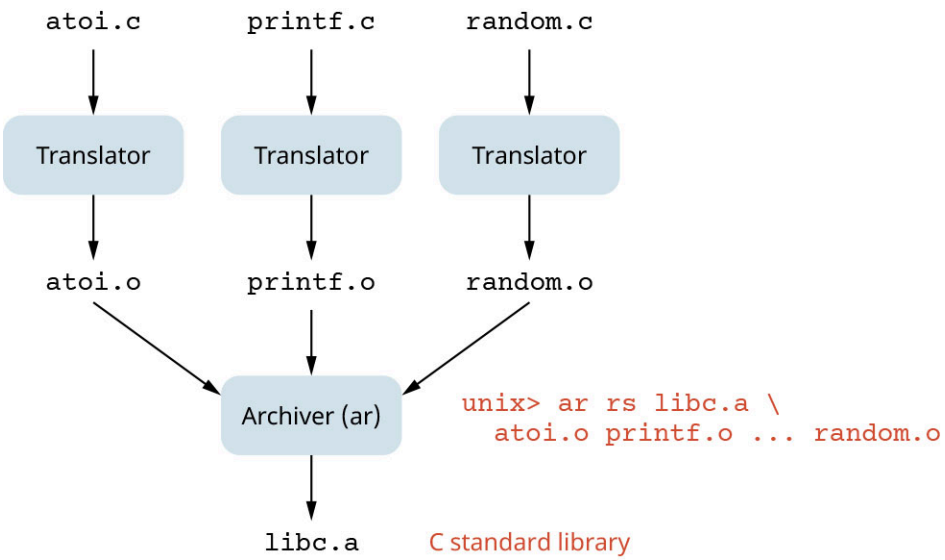


Figure 4.25 The diagram illustrates how the ar archiver utility is used to create a sample version of the `libc.a` static library that only includes the `atoi.o`, `printf.o`, and `random.o` object files. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Commonly used libraries include `libc.a` (the C standard library), which handles: I/O, memory allocation, signal handling, string handling, data and time, random numbers, and integer math. Another common library is `libm.a` (the C math library) that handles floating point math (e.g., `sin`, `cos`, `tan`, `log`, `exp`, `sqrt`).

Figure 4.26 illustrates how to link programs with static libraries.

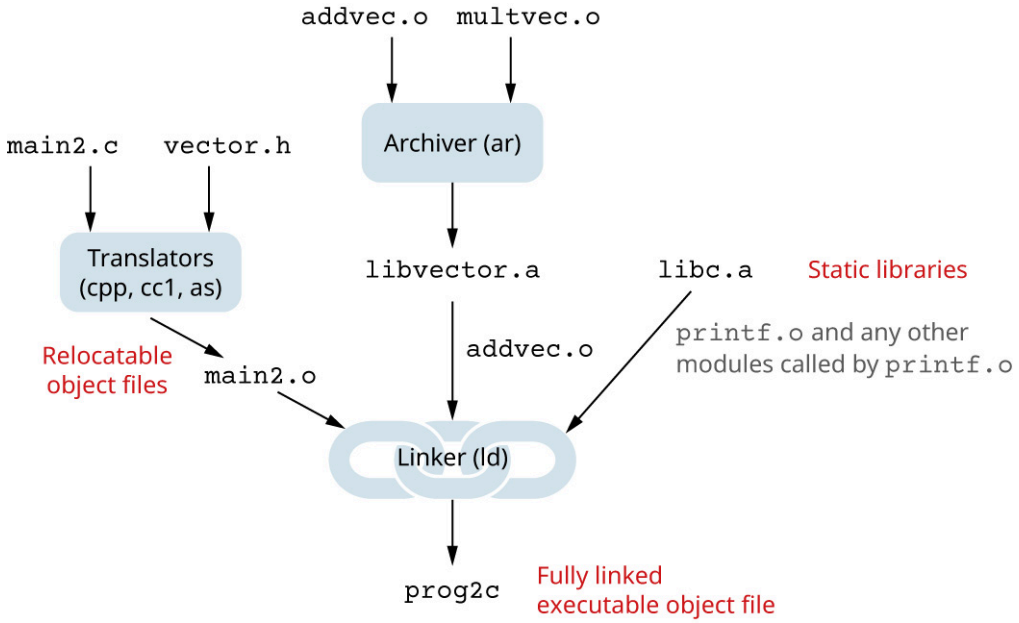


Figure 4.26 This diagram demonstrates the creation of an executable file using various static libraries. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The linker uses the following algorithm to resolve external references:

1. Scan `.o` files and `.a` files in the command line order.
2. During the scan, keep a list of the current unresolved references.
3. As each new `.o` or `.a` file, *obj*, is encountered, try to resolve each unresolved reference in the list against the symbols defined in *obj.*
4. If any entries in the unresolved list at end of scan, then issue error.

Therefore, the command line order matters and libraries should be placed at the end of the command line to avoid linker errors as illustrated in Figure 4.27.

```
unix> gcc -L. libtest.o -lmine
unix> gcc -L. lmine libtest.o
libtest.o: In function `main':
libtest.o(.text+0x4): undefined reference to 'fun'
```

Figure 4.27 Notice the link errors caused due to the incorrect order of the files. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

CONCEPTS IN PRACTICE

APIs in C

The C language makes it possible to create a modular API (Application Programming Interface) as a library with publicly visible function prototypes but secret function definitions. This is accomplished by distributing the `.h` files with function declarations freely, while keeping the `.c` files secret and instead distributing only `.a` or `.so` compiled object code. A `.h` file is used in C, C++, where the libraries can be used in the current program instead of writing the code completely.

An example is `math.h`. This strategy is used in many industries, such as video games. DirectX is an API created by Microsoft for the platforms that are used on Windows PCs and Xbox. Microsoft provides a C library with many function calls for game-related operations such as drawing graphics, playing sounds, and reading inputs from the keyboard, mouse, or joystick. A game programmer writes their game as a C program that calls those functions. This arrangement is a good compromise—the convenience of the DirectX API makes game programmers' work easier, and entices them to create games for Windows and Xbox. But keeping the `.c` files proprietary means that Microsoft does not have to give away the hard work that went into creating DirectX, Windows, or Xbox.

The same arrangement works on other platforms, too. OpenGL is a cross-platform API that works on almost every modern platform, and Sony PlayStation has a similar API. Both of these are distributed as C libraries with public .h files and proprietary implementations.

Loading Executable Object Files

An object file is a file that is a combination of metadata from the source or object code along with a combination of bytecode

Dynamic Load-Time Linking

Static libraries have the following disadvantages: duplication in the stored executables (every function needs `libc`), duplication in the running executables, and minor bug fixes of system libraries require each application to explicitly relink. A modern solution to this problem is to use shared libraries (also called dynamic link libraries, DLLs, or `.so` files). A **shared library** is a library file that can be shared by multiple programs at the same time (Figure 4.28).

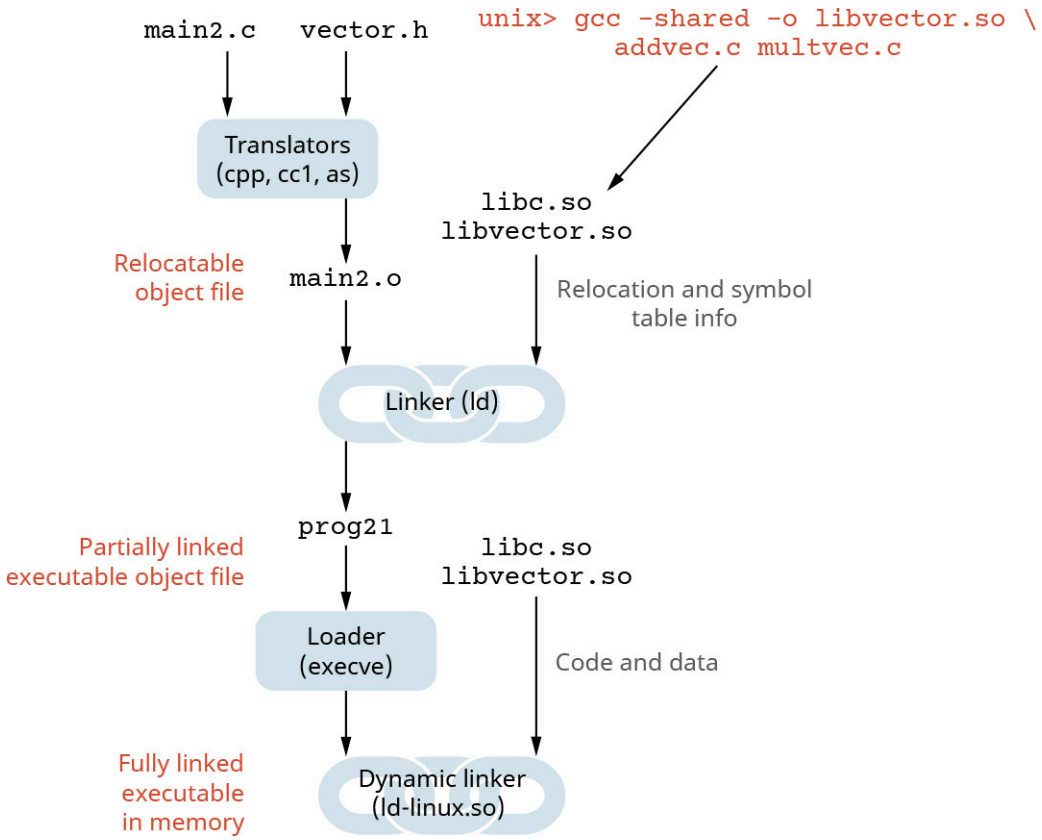


Figure 4.28 A shared library can be used by multiple programs simultaneously. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

When using shared libraries, object files that contain code and data may be loaded and linked into an application dynamically at load time, as illustrated in Figure 4.29. This **load time linking** occurs when dynamic linking happens at the same time that a program executable is first run. This is a common case for Linux, which is handled automatically by the dynamic linker (`ld-linux.so`). The standard C library (`libc.so`) is usually dynamically linked. The ldd tool may be used to identify dependencies/libraries needed at load time. In static linking the routines code becomes a part of the executable. In dynamic linking, the routines can be updated during the code execution. To dynamically link a library at load time on Linux, place it in the `/lib/x86_64-linux-gnu/` directory and compile the source files with the `-l` flag (e.g., `gcc main.c -lcso`).

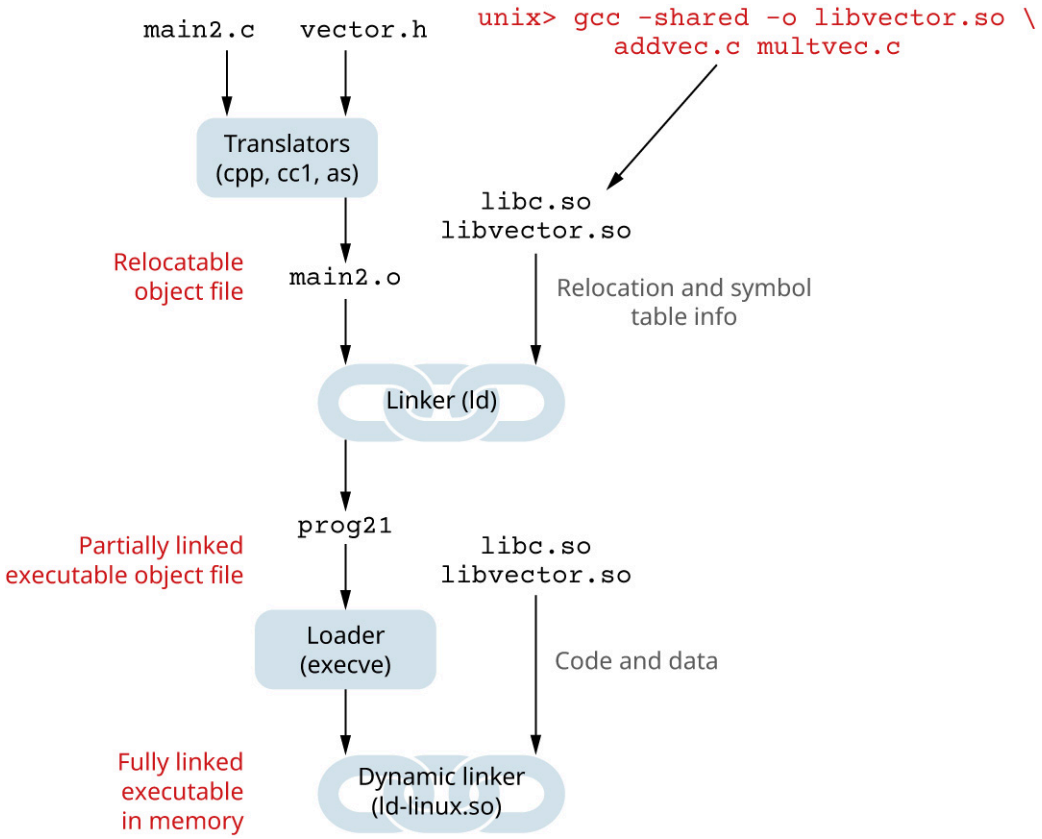


Figure 4.29 The diagram illustrates how object files may be loaded and linked into an application *dynamically*, at *load-time*; in that case, dynamic linking can occur when the program executable is first loaded and run (i.e., load-time linking), which is a common case for Linux that is handled automatically by the dynamic linker (`ld-linux.so`). (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Dynamic Runtime Linking

An alternative to load-time linking is **runtime linking**, which means that linking occurs after a program has already started running. As illustrated in the sample code, the program source code needs to explicitly call functions to link additional libraries. In Linux, this is done by calls to the `dlopen()` interface and compiling the source with the `-l` flag (e.g., `gcc main.c -ldl`). This is a better approach to help distribute software, support high-performance Web servers, or perform runtime library interpositioning.

```
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>
int x[2] = {1, 2};
int y[2] = {3, 4};
int z[2];
int main() {
    void *handle;
    void (*addvec)(int *, int *, int *, int);
    char *error;
    /* Dynamically load the shared library that contains addvec() */
    handle = dlopen("./libvector.so", RTLD_LAZY);
    if (!handle) {
      fprintf(stderr, "%s\n", dlerror());
      exit(1);
    }
```

```
    ...
    /* Get a pointer to the addvec() function we just loaded */
    addvec = dlsym(handle, "addvec");
    if ((error = dlerror()) != NULL) {
      fprintf(stderr, "%s\n", error);
      exit(1);
    }
    /* Now we can call addvec() just like any other function */
    addvec(x, y, z, 2);
    printf("z = [%d %d]\n", z[0], z[1]);
    /* Unload the shared library */
    if (dlclose(handle) < 0) {
      fprintf(stderr, "%s\n", dlerror());
      exit(1);
    }
    return 0;
}
```

Tools to Manipulate Object Files

An object file contains a lot of information such as metadata, machine code, and other information from symbols. To manipulate such files, Unix provides certain tools to use them effectively, such as:

- ar: Creates static libraries, and inserts, deletes, lists and extracts members.
- strings: Lists all the printable strings contained in an object file.
- strip: Deletes symbol information from an object file.
- nm: Lists the symbols defined in the symbol table od an object file.
- size: Lists the names and sizes of the sections in an object file.
- readelf: Displays the complete structure of an object file, including all of the information encoded in the ELF header; subsumes the functionality of “size” and “nm.”
- objdump: Displays all of the information in an object file; useful for disassembling binary instructions in the .text section.
- ldd (linux): Lists the shared libraries that an executable needs at runtime.

Version Control Management

The process and tools used to store and improve multiple versions of project files is called **version control**. Version control also helps support team collaboration, and allows for the ability to revert to an earlier versions. **Git** is a widely-used version control system. Creating and updating project files using Git requires the creation of a Git **repository**, also known as “repo” for short. A repository is a container for files and related information stored in a version control tool. **GitHub** is a website that allows free storage of public git repositories.

LINK TO LEARNING

Learn more by installing Git on your local machine (https://openstax.org/r/76InstallGit) on any platform. You may run “`brew install git`” on MacOS to install Git or “`sudo apt install git`” on Linux.

Useful Git commands are as follows:

- `git config --global user.email "you@example.com"` and `git config --global user.name "Your Name"`
- Clone: to download contents

- Pull: `git pull origin master` to pull latest changes
- Status: `git status` to see staged (shown in green) and un-staged (shown in red) files
- Staging: `git add <filename'` to add files to staged area (wildcards accepted)
- Commit: `git commit –m "<your message here>"` to commit the staged files
- Push: `git push origin master` to push all changes made locally to the origin

LINK TO LEARNING

Explore the Git/GitHub tutorial (https://openstax.org/r/76GitHubTutor) for more details on how to use Git.

4.3 Parallel Programming Models

Learning Objectives

By the end of this section, you will be able to:

- Define parallel computing and related terminology
- Discuss parallel programming approaches

So far, our programs have run on a single at a time and the assumption is that the underlying machine only supported a single GPU core. A CPU **core** is a chip consisting of billions of transistors that function according to an instruction or opcode. It is like a single processor. While there is a lot that we can do with these single-core programs, there is also a need for programs to run in parallel, meaning that they execute code on multiple CPUs, cores, or computers at the same time. In parallel programming, bigger tasks are split into smaller ones, and they are processed in parallel, sharing the same memory. Parallel programming is trending toward being increasingly needed and widespread as time goes on. Many computers now come equipped with a **graphics processing unit (GPU)**, which is a massively parallel processor that supplements a CPU. GPUs were originally designed for rendering real-time graphics in video games and are sometimes called "video cards." A typical GPU has thousands of cores, although each is weaker than a CPU core. Parallel techniques are essential for making use of GPUs.

Parallel Computing Overview

In the 20th century, a computer typically had only one processor. Now, a CPU chip typically holds not just one processor, but multiple processors built into a single computer chip. Each individual processor built into a CPU is a core. A **multicore** processor is a CPU chip that has multiple cores. Multicore CPUs are prevalent; smartphones and budget PCs typically have two to four cores, and high-end PCs have eight or more cores. The trend is for these core counts to increase over time.

By default, a program runs on one core at a time. That means that a four-core computer can run up to four programs at full speed at the same time. That capability is occasionally useful, but more often a user wants a single high-demand program to make full use of their computer. This is the case with productivity software, games, embedded systems, and Web server software. For this to work, the program needs to be coded in a way that explicitly divides work among multiple cores.

Fundamentally, in order to use multiple cores, a program needs to work in "parallel." That means that multiple cores are working together at the same time (Figure 4.30). A real-world example of parallel work is a factory assembly line. If an assembly line has twenty workers, then at any given moment twenty people are working in parallel. This concept of parallel work also applies to software. The parallelism concepts discussed here are:

- **parallel computer**: a multiple-processor system that supports parallel programming.
- **parallel computing**: the practice of making productive use of parallel computers.
- **parallel programming**: a computer programming technique that provides for executing code in parallel

on multiple processors.

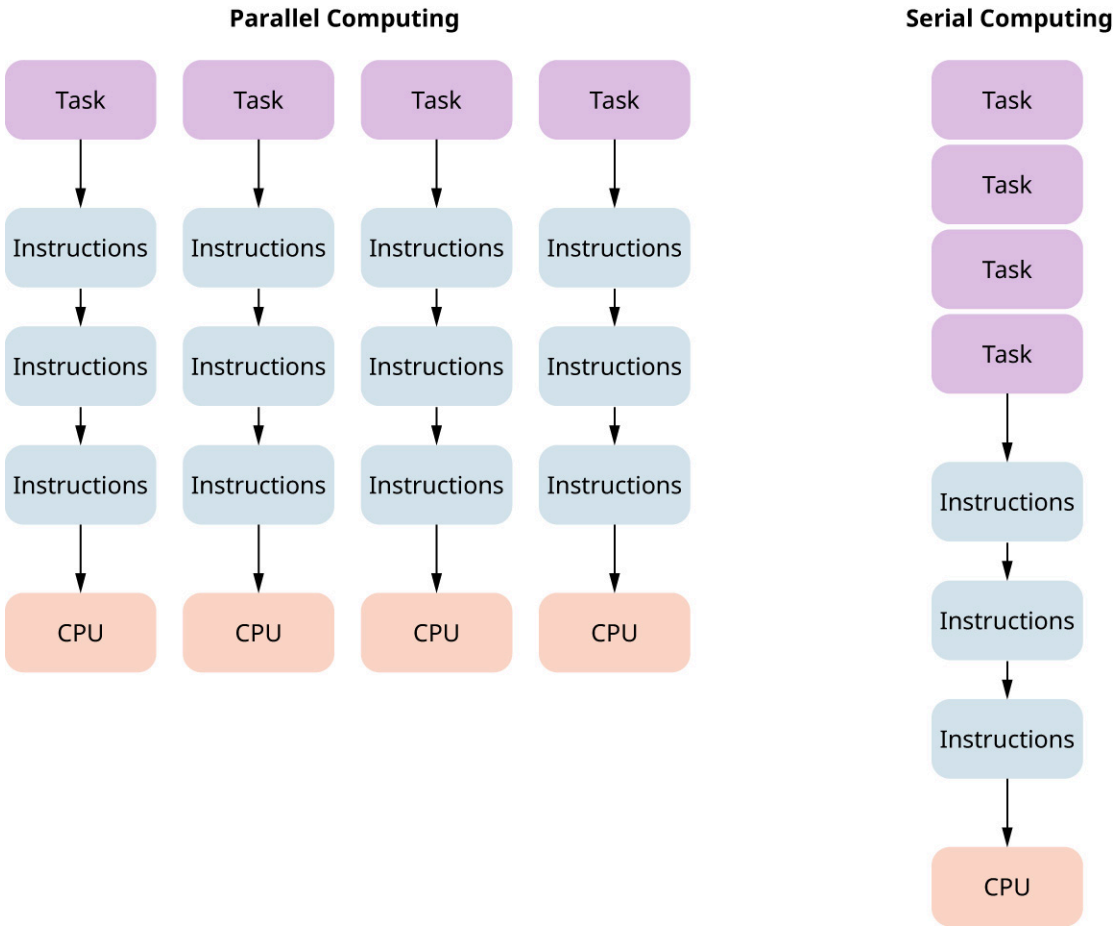


Figure 4.30 This diagram illustrates how multiple computer programs can be executed as tasks on a multi-core machine either in parallel on separate individual cores using parallel computing or in sequence (on a single or on multiple cores) using serial computing. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

There are two related but distinct terms that we should define at this point: **concurrent programming**, which refers to any situation where multiple programs or tasks are running simultaneously, regardless of whether they are using multiple processors or sharing one processor; and **distributed computing**, which is a more specific form of parallel programming where processors are working together in parallel, but the processors are in multiple connected computers, not a single computer. Concurrent programming is a broader term than parallel programming, while distributed computing usually refers to massively parallel programs that run on hundreds or thousands of servers, usually at large companies such as Amazon, Google, the NSA, and the NIH.

THINK IT THROUGH

GPU Applications

GPUs are high-performance parallel processors. Some major applications of GPUs include cryptocurrency mining, video games, and the dashboard computers embedded in automobiles. In 2021, there was a shortage of GPUs due to a “perfect storm” of world events. The COVID-19 pandemic complicated manufacturing, limiting the rate at which GPUs could be built. In response to the pandemic, demand for computers increased, as many workers were forced to work from home. Demand for video games also increased as people sought indoor entertainment. At the same time, cryptocurrency prices went up, which stimulated interest in mining cryptocurrency, so even more people tried to buy GPUs at the same time.

All these events caused a severe shortage. Customers encountered long waiting lists for GPUs, or found that they were unavailable entirely. Scalpers sold GPUs at a substantial upcharge. Some people were unable to buy video games, or computers they needed to complete work. The shortages affected heavy industry; automobile manufacturers had to idle their factories, which impacted the factory workers' livelihoods, and triggered a shortage in automobiles.

This situation pitted knowledge workers, gamers, market speculators, and manufacturers against each other in a struggle for scarce resources.

To what degree is this a problem? Do computing professionals have a responsibility to offer a technical solution, such as a technological alternative to GPUs? Do they have a responsibility to anticipate these kinds of unintended consequences? How should policy makers handle a shortage for a critical resource?

Parallel Programming

Parallel programming involves writing code that divides a program's task into parts, works in parallel on different processors, has the processors report back when they are done, and stops in an orderly fashion. C was not designed with parallel programming in mind, so we need to use third-party libraries for parallel programming in C. Some newer languages were designed with parallel programming facilities from the start.

Parallel Programming Models and Languages

A parallel programming model is a high-level conception of how the programmer can control processors and the data that moves between them.

- Shared Memory: In the **shared memory** programming model, processes/tasks share a common address space, which they read and write to asynchronously. Various mechanisms such as locks/semaphores are used to control access to the shared memory, resolve contentions and to prevent race conditions and deadlocks. One example is SHMEM.
- Threads: This programming model is a type of shared memory programming. A **thread** is a single "heavyweight" process can have multiple "lightweight", concurrent execution paths. A simple example of a thread includes a chat feature, video, or audio in an application like Microsoft Teams. Examples include Pthreads, OpenMP, Microsoft Threads, Java and Python threads, and CUDA threads for GPUs.
- Message Passing: A parallel programming approach where separate processes communicate only by sending messages, not sharing memory. Each set of tasks use their own local memory during computation. Multiple tasks can reside on the same physical machine and/or across an arbitrary number of machines. One example is the **Message Passing Interface (MPI)** that was first developed in the 1990s.
- Hybrid Model: A hybrid model combines more than one of the previously described programming models; currently, a common example of a hybrid model is the combination of the MPI with the threads model. Other examples of hybrid models include MPI with CPU-GPU using CUDA, MPI with Pthreads, and MPI with non-GPU.

To program in parallel, you can extend compilers (i.e., translate sequential programs into parallel programs), extend languages (i.e., add parallel operations on top of sequential language), add a parallel language layer on top of sequential language, and define a totally new parallel language and compiler system. The extend language strategy (2) is the most popular, and MPI/OpenMP are examples.

THINK IT THROUGH

Multi-Threading Parallel Programming

Why is it important at this time for application developers to turn to the multi-threading parallel

programming paradigm and new emerging computing technologies for their application needs?

Designing Parallel Programs

Designing and developing parallel programs has historically been a very manual process. The programmer is typically responsible for both identifying and actually implementing parallelism. Developing parallel code is often a time-consuming, complex, error-prone, and iterative process. For a number of years now, various tools have been available to assist the programmer with converting serial programs into parallel programs. The most common type of tool used to automatically parallelize a serial program is a parallelizing compiler or pre-processor. A parallelizing compiler generally works in two different ways: fully automatic or programmer directed.

In the fully automatic method, the compiler analyzes the source code and identifies opportunities for parallelism. The analysis includes identifying inhibitors to parallelism, and it may determine whether the parallelism would actually improve performance. Loops (do, for) are the most frequent target for automatic parallelization.

In the programmer-directed method, the programmer explicitly tells the compiler how to parallelize the code using "compiler directives" or possibly compiler flags. This approach may be used in conjunction with some degree of automatic parallelization. The most common compiler-generated parallelization is done using on-node shared memory and threads.

If you are beginning with an existing serial code and have time or budget constraints, then automatic parallelization may be the answer. However, there are several important caveats that apply to automatic parallelization: wrong results may be produced, performance may actually degrade, it can be much less flexible than manual parallelization, is limited to a subset (mostly loops) of code, and it may actually not parallelize code if the compiler analysis suggests there are inhibitors or the code is too complex.

The first step in developing parallel software is to (1) understand the problem that you wish to solve in parallel. Next steps include (2) partitioning, or breaking the problem into discrete "chunks" of work; (3) identifying the need for communications between tasks; (4) synchronizing the sequence of work and the tasks being performed; (5) identifying data dependencies between program statements; (6) performing load balancing to distributing approximately equal amounts of work among tasks so that all tasks are kept busy all of the time; (7) establishing granularity as the qualitative measure of the ratio of computation to communication; (8) managing I/O operations that are generally regarded as inhibitors to parallelism; (9) debugging (a technique where the program is read through line-by-line to check for any bugs) parallel code; and (10) analyzing and tuning parallel program performance. Figure 4.31 shows these steps.

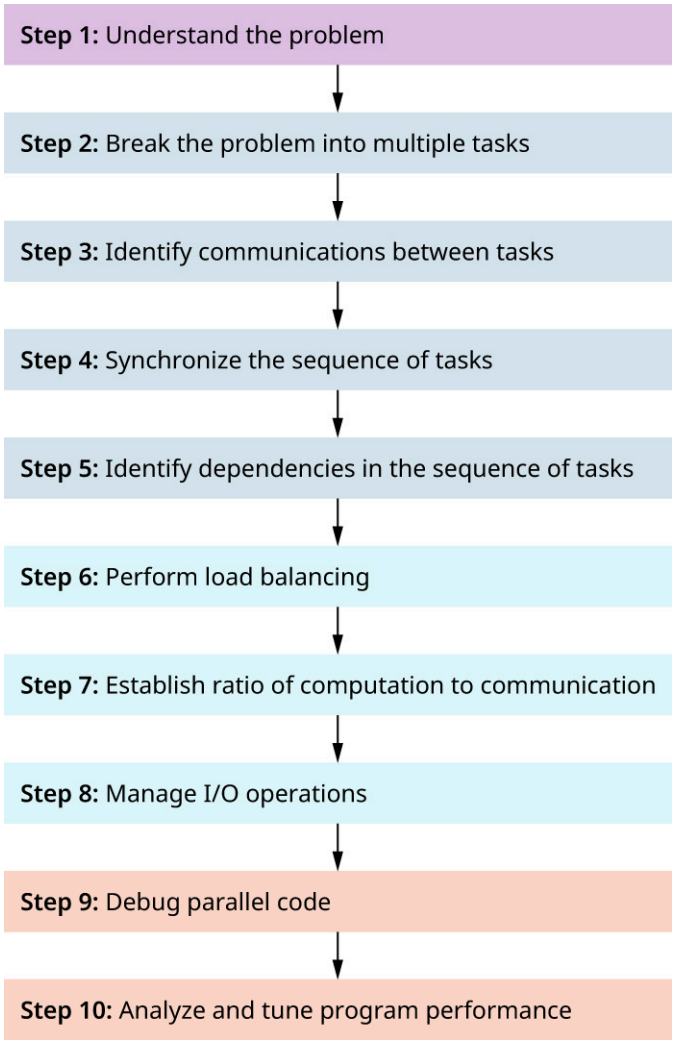


Figure 4.31 Developing parallel software follows ten important steps. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

LINK TO LEARNING

Parallel programming is a deep subject with many avenues for further study, from the low-level details of hardware and programming to high-level parallel algorithm design. Learn more about the Introduction to Parallel Computing Tutorial (https://openstax.org/r/76ParallelComp) at Lawrence Livermore National Laboratory.

Using C with MPI and OpenMP Parallel Libraries

We focus here on how parallel programs can be written in the C language using an API, which is the most popular method. Some programming languages support parallel programming and may also be used to program parallel applications using message passing features that are built into the language itself. A **message passing** feature is a parallel programming approach where separate processes communicate only by sending messages, not sharing memory. The **symmetric multiprocessor (SMP)** model applies when programming multiple processors that are practically identical. **OpenMP** is a library for parallel programming in the SMP model. When programming with OpenMP, all threads share memory and data. OpenMP supports C, C++ and Fortran. The OpenMP functions are included in a header file called omp.h. An OpenMP program has sections that are sequential and sections that are parallel. In general, an OpenMP program starts with a

sequential section in which it sets up the environment, initializes the variables, and so on. When run, an OpenMP program will use one thread in the sequential sections, and several threads in the parallel sections. The **parent thread** is the thread that runs from the program beginning through end, and starts and manages child threads. A **child thread** is started by the parent thread and only runs for a limited period in a parallel section. A section of code that is to be executed in parallel is marked by a special directive that will cause child threads to form. Each thread executes the parallel section of the code independently. When a thread finishes, it joins the parent. When all threads finish, the parent continues with code following the parallel section.

INDUSTRY SPOTLIGHT

Artificial Neural Networks

The field of artificial intelligence makes heavy use of parallel computing. Artificial neural networks (ANNs) are a widely-used technology that simulates the flow of impulses through nerve cells in a brain. An ANN needs to be "trained" by feeding it many examples of the kinds of inputs and outputs that it will deal with. This training process benefits greatly from parallel programming. A typical ANN has thousands of simulated cells, and is trained on thousands of examples. This makes for millions, or even billions, of computations; parallel computing is a great benefit because this training process can be performed in parallel. Hardware manufacturers, including NVIDIA, Intel, and Tesla, have even created GPU-based computers specifically for the task of training ANNs. Figure 4.32 illustrates the model of a neural network.

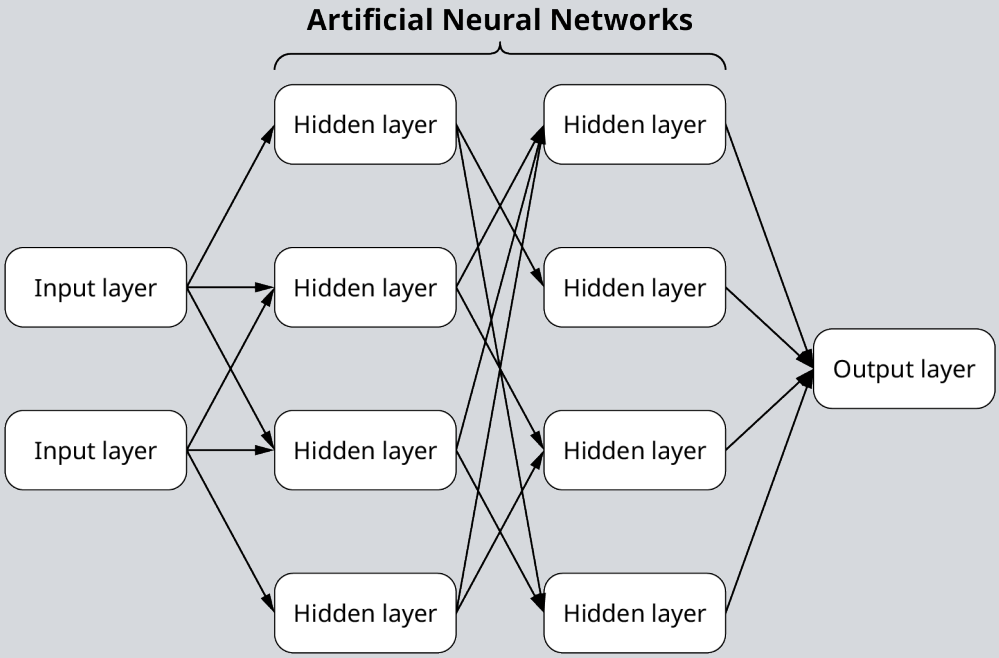


Figure 4.32 The figure illustrates a fully connected set of layers in an ANN. In this case, computing the value maintained by each node requires combining values provided by all the node's input nodes, which explains why training ANNs requires so many computations if all nodes are fully connected to other nodes. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

4.4 Applications of Programming Models

Learning Objectives

By the end of this section, you will be able to:

- Discuss the future of low-level programming
- Understand how the C language is used to develop firmware for embedded systems
- Develop kernel code using the C programming language

High-level languages are popular, and have their place, but there are certain applications where only middle-level languages such as C will do. This section showcases two such applications: firmware and kernel development.

The Future of Low-Level Programming

The economic trends that diminish interest in low-level programming are expected to continue and even accelerate. We are using wider varieties of computer hardware—not just personal computers, mobile devices, and servers, but also narrower segments such as tablets, set-top boxes, video streamers, and system-on-chip (SoC) computers such as the Raspberry Pi (Figure 4.33).



Figure 4.33 A Raspberry PI demonstrates the concept of a system-on-chip (SOC). (credit: "Raspberry-Pi-2-Bare-BR" by "Evan-Amos"/Wikipedia, Public Domain)

The IoT is the growing network of products that are not used as a computer, but nevertheless contain an Internet-connected computer. IoT devices include but are not limited to smart voice assistant speakers, thermostats, home appliances, speakers, and tap payment systems. The computer embedded inside an IoT device is limited in terms of size, energy use, and cost, so it typically has a slow CPU and small memory. This makes middle-level languages well suited to writing IoT software; writing Internet-connected applications in low-level languages is impractical, and high-level languages may not be efficient enough.

Rust

Rust is a relatively new middle-level language created by the Mozilla Foundation in the 2010s. Many of C's positive features are also found in Rust: efficient execution, portability, modularity, procedural and structured programming, and recursion. Rust has the capability to manipulate pointers but adds native safety features so that the compiler can help the programmer prevent bugs related to pointers. The language also includes some features that are more common to high-level languages and are unavailable in C, including higher-level data types (lists, maps, and sets), macros, templates, and parallel programming. These features do make Rust more complicated than C, though. Since Rust has the same positive attributes as C, with some additional desirable features, we expect to see increasing use of Rust as a middle-level language.

GLOBAL ISSUES IN TECHNOLOGY

Naming of C Procedures

It is a best practice to give procedures descriptive names that help a reader understand what the procedure does. English has been globally accepted as the language of programming, but this can cause some problems in non–English-speaking countries. For example, “open_file” is more descriptive than “fopen”; it is also difficult to tell what “fopen” might mean out of context and especially if you are not a fluent English speaker. Unfortunately, procedure names in C are almost always written in English. How can this practice of using English names affect aspiring programmers whose first language is not English? How will it make it more difficult for programmers around the world to collaborate on source code?

Firmware

Hardware is purely physical machinery, and software is purely digital code. To bridge this gap, we have **firmware**, which is very low-level code that communicates directly with hardware, and provides a convenient interface for other software. (“Firm” is the halfway point between “hard” and “soft.”)

If you want to learn to be an embedded systems engineer, it would be best to start from a simple hardware kit, rather than starting with the latest Intel or ARM chipset. **Arduino** is a hardware platform intended for creating simple, low-cost hardware for educational or hobbyist purposes. There are many series of Arduinos, but their "Arduino PLC Starter Kit" has a simple processor and comes with a guide book. Atmega328P has an 8-bit core, which is a good place to start digital circuit design and firmware development. You do not need to know how to draw schematics and layouts and assemble the chips. But you do need to know how to read schematics and understand how the chips are connected. Firmware developers should be able to read the schematics and figure out how to send data to the target device.

TECHNOLOGY IN EVERYDAY LIFE

You and the Internet of Things

As IoT technology advances and computer parts get less expensive, more and more categories of IoT device are coming to market. Fitness monitors have helped people stay in shape. Smart thermostats have conserved energy and made homes more comfortable. Other types of home automation have improved quality of life for older adults and people living with disabilities. These technologies are enabled by middle-level languages that make efficient use of computer hardware, such as C.

What is a new category of IoT device that does not exist yet, but would make your life, or the life of your loved ones, better? What kind of software would this device need? Could you write it in C?

OS Kernels and Device Drivers

The Raspberry Pi board has a Cortex-A53 Processor that supports a 64-bit instruction set. This allows you to experience a modern processor architecture with rPi. Information relating to Raspberry Pi is constantly changing, and the best way to fully understand it is to tackle making your own kernel. There are several websites where you can do this:

- OSDev Wiki (https://openstax.org/r/76OSDevWiki)
- Older toy kernel (https://openstax.org/r/76ToyKernel) that supports 64-bit long mode, paging, and very simple context switching
- The Little Book about OS Development (https://openstax.org/r/76OSDevBook)

- Operating Systems: From 0 to 1 (https://openstax.org/r/76OperSystems)

Making a toy kernel is good way to understand modern computer architecture and hardware control. In fact, you already have a powerful processor and modern hardware devices on your laptop or desktop. This may be all you need to get started.

The Qemu emulator (https://www.qemu.org/) can emulate the latest ARM processors and Intel processors, so everything you need is already on hand. There are many toy kernels and documents you can refer to. You can install Qemu emulator and make a tiny kernel that just boots, turns on paging, and prints some messages. You do not need to make a complete operating system. Join the Linux community and participate in development.

LINK TO LEARNING

This step-by-step guide teaches how to create a simple operating system (OS) kernel (https://openstax.org/r/76OSkernel) from scratch. Each lesson is designed in such a way that it first explains how some kernel feature is implemented in the rPi OS, and then it tries to demonstrate how the same functionality works in the Linux kernel.

Chapter Review

Key Terms

abstract model technique that derives simpler high-level conceptual models for a computer while exploring the science of what new algorithms can or cannot do

allocated memory memory region set aside to hold a value

archive file **.a** file that contains a static library

Arduino hardware platform intended for creating simple, low-cost hardware for educational or hobbyist purposes

assembler program that translates assembly language source code into machine code

assembly language low-level language in which every statement corresponds directly to a machine instruction

BASIC early high-level programming language

binary code program in the native format that is understood by a CPU, which is a long series of 0s and 1s

C middle-level language that has been in wide use since the 1970s

C++ middle-level object-oriented language based upon C

central processing unit (CPU) computer chip capable of executing machine code programs

child thread thread that is started by the parent thread, and only runs for a limited period in a parallel section

Church-Turing Thesis scientific theory stating that an algorithm can be converted from any reasonable computational model to another

Clang open-source C compiler developed by the LLVM project

code relocation merges separate code and data sections into single sections (one for code and one for data)

compiler (also: **interpreter**) program that translates source code from a middle-level or high-level language into something a computer can read

computational model system for defining what an algorithm does and how to run it

concurrent programming situation where multiple programs or tasks are running at the same time, regardless of whether they are using multiple processors or sharing one processor

core individual processor built into a CPU chip

declarative programming paradigm in which code dictates a desired outcome without specifying how that outcome is achieved

device driver piece of code that is responsible for connecting to a hardware component such as a video card or keyboard

distributed computing specific form of parallel programming where processors are working together in parallel, but the processors are in multiple connected computers, not a single computer

executable and linkable format (ELF) standard binary format for object code

external reference symbol that is used in a module, but not defined in that module, so is expected to be defined in some other module

firmware very low-level code that communicates directly with hardware, providing a convenient interface for other software

freed memory memory that is given back to be reused when a value is no longer needed

functional programming paradigm in which algorithms are written as mathematical functions

GCC open-source C compiler developed by the GNU Project

Git widely-used version control system

GitHub website that allows free storage of public git repositories

GOTO non-structured operation that instructs a computer to jump to an entirely different part of the program

graphics processing unit (GPU) massively-parallel processor that supplements a CPU; GPUs were originally designed for rendering real-time graphics in video games

hardware model design for a how a specific physical computer executes algorithms
high-level programming language programming language that operates at a high level of abstraction, meaning that low-level details such as the management of memory are automated
imperative programming paradigm in which the programmer writes a series of steps that must be followed in order
instruction set architecture (ISA) type of hardware model that defines a list of operations that a CPU can execute
integrated development environment (IDE) program with a graphical user interface that includes a text editor, compiler, and other tools, all in one application
interpreter (also: **compiler**) program that translates source code from a middle-level or high-level language into something a computer can read
invalid pointer pointer that does not hold a valid location
kernel core part of an operating system that is responsible for managing and interfacing with hardware components
Lambda calculus abstract computational model defined by Alonzo Church that inspired the functional programming paradigm
level of abstraction degree to which a computational model, programming language, or piece of software relates to computer hardware
library file that contains object code for functions and global variables that are intended to be reused
linker program that performs linking
linking process of collecting and combining various pieces of object code into a single program file that can be loaded into memory and executed
Linux open-source operating system kernel that is Unix-compatible
load time linking when dynamic linking happens at the same time a program executable is first run
low-level programming language programming language that operates at a low level of abstraction, meaning that code is similar to machine code
machine code sequence of binary digits (bits) that can be understood and executed directly by a computer
memory leak occurs when some memory is allocated but never freed
memory management process of allocating and freeing memory
message passing parallel programming approach where separate processes communicate only by sending messages, not sharing memory
Message Passing Interface (MPI) message-passing interface that was first developed in the 1990s
middle-level programming language programming language that is somewhat abstracted above low-level, but not as much as a high-level programming language; allows direct hardware access
modularity property of code that allows it to be divided into a small, reusable piece
multicore CPU chip that contains more than one core
object a program value that has both data, or variables, and procedures that work together to represent a specific human concept
object-oriented programming paradigm in which code is organized into objects, where each object has both data and procedures
OpenMP library for parallel programming in the SMP model
operating system software that provides a platform for applications and manages hardware components
operator fundamental programming operation that combines values
parallel computer multiple-processor system that supports parallel programming
parallel computing practice of making productive use of parallel computers
parallel programming computer programming technique that provides for executing code in parallel on multiple processors
parent thread thread that runs from the program beginning through the end, and starts and manages child threads
pointer variable that holds the memory address of another variable and points to that variable

procedural programming paradigm in which code is organized into procedures
procedure function in the context of programming
programming language paradigm philosophy and approach for organizing code
programming model design for humans to read and write
Random Access Machine abstract computational model used to analyze the efficiency of algorithms
repository container for files and related information stored in a version control tool
runtime linking when linking occurs after a program has already started running
Rust a relatively new middle-level programming language created by the Mozilla Foundation in the 2010s
segmentation fault occurs if the subscript is very far out of range
semantic error when code compiles and runs, but does not behave as it should
shared library library file that can be shared by multiple programs at the same time
shared memory programming model in which processes/tasks share a common address space, which they read and write to asynchronously
socket Internet connection between two computers
source code text of a program written in a programming language
static library simple kind of library that that copies the contents of object files into a single file called an "archive"
structured programming paradigm in which control flow is always controlled with conditionals ("if") or loops ("while") and never GOTO
symbol identifier for a function or global variable
symbol resolution during the symbol resolution step, the linker associates each symbol reference with exactly one symbol definition
symbol table array of structures in which each entry includes name, size, and location of symbol
symmetric multiprocessor (SMP) model in which there are multiple parallel processors that are practically identical
systems software programs that provide infrastructure and platforms that other programs rely upon
thread light-weight parallel execution path that shares memory with other threads
Unix operating system that has been used widely, primarily in servers and software development since the 1970s
version control tools that are used to store and improve multiple versions of project files and support team collaboration, and the ability to revert to an earlier versions
Visual C++ proprietary-license C and C++ compiler developed by Microsoft

Summary

4.1 Models of Computation

- A computational model defines what an algorithm or program does. There are hardware models, programming language models, and abstract models.
- Low-level programming means writing machine code that can be understood by a CPU directly, or something very near to that. It is laborious but yields very fast code.
- Middle-level programming is a compromise that is reasonably efficient and more convenient than low-level programming.
- High-level programming is more abstract and intuitive for humans. It is less labor-intensive, but high-level code can be slower than low-level code.
- Programming language paradigms are approaches for organizing source code.
- In the imperative paradigm, code orders the CPU to execute specific actions.
- In the declarative paradigm, the programmer declares the outcome that they need.
- In the functional paradigm, the programmer defines mathematical functions to evaluate.
- In structured programming, the flow of execution is specified with explicit syntax elements ("if-then-else," "for" loop, "while" loop) and never GOTO.
- In procedural programming, a program is divided into procedures. Each procedure performs one specific

task and has a descriptive name.
- In object-oriented programming, the basic building block is an object. An object combines data and procedures that together represent a human concept.

4.2 Building C Programs

- The C programming language is the most prominent example of a low-level language. Programs written in C typically execute as fast as assembly language and allow programmers to directly manipulate machine features such as memory via the use of pointers.
- The C programming language is used by most of the system software we depend on today (e.g., operating systems, compilers, interpreters, and device drivers) because of its efficient execution, portability, and modularity.
- The C data model supports the creation of a variety of basic types including integers and floating point numbers, as well as pointers. C also provides type constructors used to create collections using the `array`, `struct`, and `union` keywords.
- C supports the imperative and structured/procedural programming paradigms and allows for conditional and iterative statements as well as functions, which can leverage recursion.
- Program development steps in C require designing algorithms, developing programs that implement algorithms, and compiling, linking, and executing programs. All of these steps may be performed within a C development environment, which is a suite of tools that a programmer uses to create software. It must include a text editor, compiler, and linker, and may include other tools such as a version control manager, and others.
- A development environment may be an assemblage of separate command-line programs, or an IDE, which is a development environment bundled into a single app. There are a multitude of C development environments, and many of them are free to use.
- Compiling C programs involves converting C into assembly language, which can itself be translated into machine code. This process is performed by using a combination tools known respectively as a C compiler, assembler, and linker. The GCC compiler is an open-source C compiler developed by the GNU project, it include gcc, ar, and ld to implement compiling, assembly, and linking.
- Linking is the process of combining the .o files that result from separate C modules into one deliverable library or executable program.
- A library is a file that contains the object code of compiled functions. There are several variations of libraries (static, dynamic, shared) and ways of linking them (load-time, run-time).
- Version control tools manage the files created in programming, facilitating collaboration, backups, and undoing errors.

4.3 Parallel Programming Models

- Multicore computers are commonplace. Most consumer mobile devices, computers, and video game consoles have between two and eight cores. As time goes on, the number of cores in computers tends to increase.
- Parallel computing does not happen automatically. Rather, a programmer must deliberately write a program in a parallel manner in order for it to use multiple cores.
- A variety of models of parallel programming exist, including shared memory, threads, and message passing.
- OpenMP is a library for writing parallel code using the message-passing model.
- One strategy for parallel computing is to have a parent thread, which creates and controls child threads. The child threads work in parallel.

4.4 Applications of Programming Models

- Low-level and middle-level programming will continue to be important as society increasingly relies on low-powered computing devices and IoT.
- Middle-level languages including C are ideal for developing firmware and kernels.

- Rust is a relatively new middle-level language that is gaining traction.
- Arduino is an embedded computer platform. Arduino firmware can be written in C.
- Raspberry Pi kernels can be developed in C.

Review Questions

1. What is the difference between machine code and assembly language?
   a. Machine code is written in a textual format, while assembly language is written in hexadecimal.
   b. Machine code is executed directly by the CPU, while assembly language must be interpreted.
   c. Machine code is written in binary, while assembly language is written in a textual format.
   d. Machine code and assembly language both require a compiler to be executed.

2. Why are middle-level programming languages like C important?
   a. They are used exclusively for Web development.
   b. They are used to create systems software, such as operating system kernels.
   c. They do not allow access to hardware features.
   d. They are used only for academic purposes.

3. What is an advantage of high-level programming over low-level programming?
   a. High-level programming languages are less time-consuming for the programmer.
   b. High-level programming languages are slower.
   c. High-level programming languages have a lower level of abstraction.
   d. High-level languages offer less security and reliability.

4. What defines the order of code execution in the imperative programming paradigm?
   a. the compiler
   b. data flow and transformations
   c. the steps in the code
   d. There is no specific order of execution.

5. What is the difference between an IDE and a development environment that is not an IDE?
   a. An IDE is a single tool for coding, while a non-IDE environment requires no tools.
   b. An IDE bundles all of the tools into one app with a graphical interface, while a non-IDE environment is a collection of several tools.
   c. An IDE uses only command-line programs, while a non-IDE environment uses graphical tools.
   d. A non-IDE environment uses graphical tools only.

6. What is a critical step that a programmer must focus on when participating in the development of a C program?
   a. planning the development
   b. documenting the code
   c. fully testing the code
   d. generating code

7. What is “ELF” in C programming?
   a. a function library for C programs
   b. an error logging framework in C
   c. a coding standard for writing C programs
   d. a file format for object code

8. Why is linking necessary in C programming?
   a. to combine separate object files into a single executable or library
   b. to compile the source code into object code
   c. to debug the program before it is executed

d. to run the compiled program on the operating system

9. What is a core?
a. a type of memory used for storing data in a computer
b. an individual processor in a CPU chip that can execute instructions
c. a program that runs on a computer
d. a software module that manages system resources

10. What is a thread in the context of computing?
a. a single task that executes on a core
b. a component that stores data in a database
c. a network connection between two computers
d. a type of memory used in parallel computing

11. What are the roles of a parent thread and child thread in parallel computing?
a. The child thread starts other threads and manages the program's execution.
b. The parent thread performs tasks assigned by the child thread.
c. The parent thread starts child threads, monitors them, and cleans the program.
d. Both parent and child threads perform the same tasks simultaneously.

12. What are examples of parallel programming models?
a. single-threaded and multi-threaded
b. interpreter and compiler
c. input/output and file systems
d. shared memory and message passing

13. What is firmware?
a. a code that interacts with hardware devices
b. a type of high-level software with high abstraction
c. an operating system that runs on embedded devices
d. a hardware component that updates software

14. Why is it recommended to start with simple hardware kits when learning embedded systems engineering?
a. Simple kits provide advanced processors.
b. Simple kits are intended for industrial use and can be used for complex processes.
c. Simple kits allow users to build digital circuit design and explore firmware development.
d. Simple kits require extensive prior knowledge of x86-64 architecture.

15. What features does the Rust programming language offer that makes Rust more desirable then C?
a. Since the language is object oriented, there is higher level data types.
b. Rust is a high-level language, so it tends to be more readable.
c. Rust provides a standard library and C does not.
d. Has the same positive features of C and includes new features supporting higher level data types and parallel programming.

Conceptual Questions

1. Why are low-level languages like assembly language not portable?

2. Why do multiple computational models exist? Why don't we just use one model for everything?

3. What are some disadvantages with structured programming languages like C?

4. Why does the execution speed of software matter? Describe a scenario where slow execution speed would impact a computer user negatively.

5. What is an advantage of low-level programming over high-level programming?
6. What is the best way to use GOTO in the structured paradigm?
7. For each of the following programming languages: investigate the language. Is the language low-level, middle-level, or high-level? Which of the paradigms covered in this section apply to the language (imperative, declarative, functional, structured, procedural, object-oriented)? Cite your sources.
    - Fortran
    - Haskell
    - Smalltalk
8. What is the output of the following C program, and what is the program's purpose here?

```c
#include <stdio.h>
int main() {
  int a=10;
  int *b=&a;
  printf("%d", b);
  printf("\n");
  printf("%d", &a);
  printf("\n");
  printf("%d",*b);
  return 0;
}
```

9. What is the output of the following C program, and what is the program's purpose here?

```c
#include <stdio.h>
int main() {
  int arr[2][3] = { 10, 20, 30, 40, 50, 60 };
  printf("Array:\n");
  for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 3; j++) {
      printf("%d ",arr[i][j]);
    }
    printf("\n");
  }
  return 0;
}
```

10. Compare and contrast static linking of a program with dynamic load-time linking. What are the advantages and challenges of each?
11. Compare and contrast dynamic load-time linking of a program with dynamic run-time linking. What are the advantages and challenges of each?
12. Non-parallel programs work, and parallel programming can be difficult. Why is it important for programmers to make the effort to make their programs parallel? Why not stick with non-parallel programming?
13. Why is it difficult for developers to use the multi-threaded programming paradigm in order to fully utilize the capabilities of today's available multicore processors?

14. Do you need to know how to assemble hardware and draw schematics to implement firmware for embedded systems?

Practice Exercises

1. List three pros or cons about each level of language in terms of execution time, complexity, readability, abstraction, and speed of development.
2. Write 8086 assembly code to add the values 1 and 3 together. The registers abx and cdx are available to use for this operation and the result should be stored in abx.
3. Write a main function in C that calls another function to add the numbers 1 and 3 together and return the sum as an output parameter. Finally, print out the answer to the console.
4. Write a main method in Java that calls another method to add the numbers 1 and 3 together and then print out the answer to the console.
5. Write the GCC commands to compile file1.c, file2.c and file3.c and then link the object files to create a static library titled myLib.
6. Write a main function in C that calls a public function titled `PrintList()` in a module titled listOperations.
7. Write a C module that includes 2 global integer variables declared in the file titled `variables.h` and then print the global variables in a main function that exists in `main.c`.
8. Write a Git command to pull changes from a repository on your local host machine with the URL "https://localhost/MyRepository" into your working repository checkout.
9. Write a GCC command to compile a C file titled `main.c` that includes a static library titled `myStaticLib`.
10. Trace the following C code and list the contents of the array after the iteration.

```
int main() {
  int List[5];
  int a = 10;
  for (int i = 0; i < 5; i++)
  {
    List[i] = a + i;
  }
  return 0;
}
```

11. The following is a C "hello world" program that uses OpenMP. How many lines of messages will this program generate at runtime?

```
#include
int main() {
int x = 1;
int y = x + 2;
#pragma omp parallel num_threads(y * 3)
{
printf("https://helloacm.com\n");
}
return 0;
}
```

12. Write a C module that creates four threads to call a function that prints the thread number out.

Problem Set A

1. For each of the following programming languages: investigate the language. Is the language low-level, middle-level, or high-level? Which of the paradigms covered in this section apply to the language (imperative, declarative, functional, structured, procedural, object-oriented)? Cite your sources.

    A. Kotlin
    B. Lisp
    C. PASCAL

2. Explain how levels of abstraction affect speed of development and speed of execution.
3. Explain how a compiler assists in providing abstractions for high-level languages.
4. Write a module titled “triangle operations” that has two functions: one to compute the area of a triangle given a base and height and one to compute the perimeter of the triangle given three sides. Then, write a main function that iterates three times, increasing each variable by one, and then calls each function.
5. OpenMP provides the `omp_get_thread_num()` function in the header file `omp.h`. To get the number of total running threads in the parallel block, you can use function `omp_get_num_threads`. How can you modify this program to ensure that only one thread executes the “Greetings from process” `printf` statement?

```
#include <stdio.h>
#include <omp.h>

int main() {
  #pragma omp parallel num_threads(3)
  {
  int id = omp_get_thread_num();
  int data = id;
  int total = omp_get_num_threads();
  printf("Greetings from process %d out of %d with Data %d\n", id, total, data);
  }
  printf("parallel for ends.\n");
  return 0;
}
```

6. Read the documentation on device drivers (https://openstax.org/r/76PSA1) and implement the various examples provided.

Problem Set B

1. Rosetta Code (https://openstax.org/r/76RosettaCode) is an archive of computing tasks and source code written in many different languages that accomplish the same task. Explore the page about converting numbers into Roman numerals (https://openstax.org/r//76Rosetta) and study the source code written in 8080 assembly (low-level), C (middle-level), and JavaScript (high-level). Compare and contrast the following aspects of the code:

    - Length
    - Readability: how easy is it to understand how the code works?
    - Level of abstraction

- Structured or unstructured

2. Research x86, ARM, and PowerPC architectures—specifically, how each of them has different assembly language features and syntax. Then research and explain how high-level languages can be compiled on different computer architectures.
3. Provide a real-life example of abstraction and explain how it is similar to abstraction in computing.
4. Research Java Abstract classes. After researching, provide a detailed usage of abstract classes and explain why abstraction is useful in software development.
5. Let a and b denote object modules or static libraries in the current directory, and let a->b denote that a depends on b, in the sense that b defines a symbol that is referenced by a. For each of the following scenarios, show the minimal command line (i.e., one with the least number of object file and library arguments) that will allow the static linker to resolve all symbol references:

   a. `p.o -> libx.a -> p.o`
   b. `p.o -> libx.a -> liby.a and liby.a -> libx.a`
   c. `p.o -> libx.a -> liby.a -> libz.a and liby.a -> libx.a -> libz.a`

6. Write a program that utilizes parallel computing, then has a safety-critical section of code that only allows one thread, and then the remainder of the program can use the same number of threads the first section used.
7. Use QEMU (https://openstax.org/r/76QEMU) and gdb (https://openstax.org/r/76gdb) to run the kernel source line-by-line.

Thought Provokers

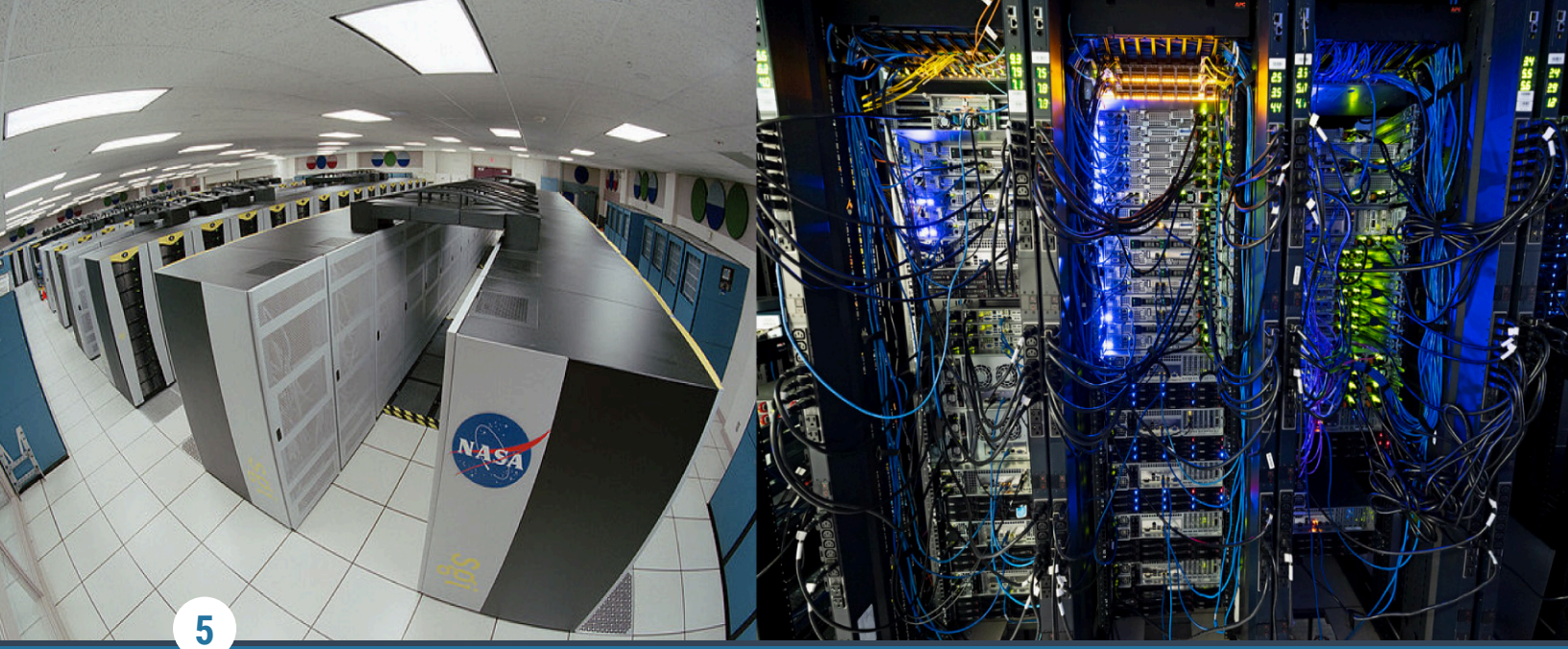
1. You are working on a project as a lead software engineer. Your team is tasked with writing a web application and a sensor that will collect and report the temperature of a room over the course of a week on a single battery charge. Your team is well versed in JavaScript and C. What language would you select to write the web application GUI and the code to operate the sensor? Explain the choice of language while connecting the level of each language.
2. TechWorks decided to implement their prosthetic control software with a low-specification (e.g., reduced instruction set) CPU and the C programming language. Alternatively, they could have used the high-level language Java, and a more powerful CPU that consumes more energy. How does this design decision impact the user's experience?
3. The TIOBE Index (https://openstax.org/r/76TIOBEIndex) is a ranking of the popularity of programming languages. How does the popularity of C, Java, and Python compare? Why do you think that is?
4. TechWorks is a small, growing startup and has four intern programmers working on their prosthetic product. Suppose that you are their manager. So far, they have made do without using any version control system. One of the intern programmers, Alice, suggests that they should set up and use Git as other programmers do in the company. She estimates that she would need to spend one day setting up the server, and all four intern programmers would need to spend one day to learn how to use Git. Is this a justified use of time? Why or why not?
5. The TechWorks prosthetic CPU has two cores, and the control software is written in C. Currently the code does not use any parallel programming. One of the programmers on the team, Bob, suggests that the software should use threads so that the function that moves the prosthetic, and the function that applies software updates, can run at the same time. What are the advantages and disadvantages of this approach?

6. When the Mozilla Foundation created Rust, C was already an established middle-level language. Why was it worth the effort for them to create an entirely new language? You may wish to consult online sources; if you do, cite them.

Labs

1. Work with a partner to collaborate using GitHub. Both students should create GitHub accounts, which are free. Student A creates a repository and adds Student B as a collaborator. B pulls the repo, makes changes, and pushes them. A pulls B's changes, makes some additional changes, and pushes them. B pulls again and sees A's changes reflected.
2. Set up a Git server and client. Install and configure the Git server on your computer; you may need to consult Internet resources. Then, use the command line `git` client tool to create a repository, add some files to it, and push the files. Confirm that you can pull the repo and view your changes from a different computer or in a different directory.
3. Write two versions of a program that takes as input a 2-D array of integers and increment each element by 1. The first version accesses the array row-wise. The second version accesses the array column-wise. Which version is faster? Why?
4. Experiment with the QEMU emulator (https://openstax.org/r/76QEMUEmulat) to emulate the latest Intel processor and run a toy kernel using one of the following:

- The Little Book about OS Development (https://openstax.org/r/76LittleBook)
- Operating Systems: From 0 to 1 (https://openstax.org/r/76OS0to1)



5

Hardware Realizations of Algorithms: Computer Systems Design

Figure 5.1 What does the word *computer* really mean? For example, supercomputers (left) and data centers (right) are types of computers. "(credit left: modification of “Columbia Supercomputer—NASA Advanced Supercomputing Facility” by Trower, NASA/ Goddard Space Flight Center, Public Domain; credit right: modification of "2020 Data Center" by Jefferson Lab/Flickr, Public Domain)"

Chapter Outline

5.1 Computer Systems Organization
5.2 Computer Levels of Abstraction
5.3 Machine-Level Information Representation
5.4 Machine-Level Program Representation
5.5 Memory Hierarchy
5.6 Processor Architectures

Introduction

We use the word *computer* a lot, but we may not know a precise definition of it. More often than not, we use it to mean our desktops and laptops. But computers exist in many different forms, like your laptop, smartphone, or tablet. Embedded processors are used to power smart home security systems. When you access your social media accounts on Facebook, Twitter, Instagram, or any website, you are using very big computers hosted by these companies. These huge and powerful computers are clusters of computers hosted in data centers and supercomputers for some applications.

These computers have many things in common, but they also differ in many aspects. In general, computers all have processors, memory, storage, and input/output devices such as keyboards, screens, and speakers. What are these components? And how do they interact with each other to form what we know as a computer? This is what we will explore in this chapter. The difference between the computer inside your watch and the one running the big sites is the number and strength of processors (computer brain), the size of memory and disks (main means of storage), and how these many pieces are connected to each other. But the main concepts are the same.

A computer’s main job, as you may have guessed, is to do computations—a lot of computations. The faster a computer, the more computations it can do per second. All computer programs you use are based on computations, whether a modern immersive graphics intensive game that leverages artificial intelligence (AI),

a text editor, or a web browser. In this chapter, you will learn how computers can do computations in a fast and correct manner using processors, memory, disks, and other related hardware.

A company called TechWorks is taking advantage of leverages such as the latest nanotechnology, processor models known as neuromorphic processors, to enable the creation of the next generation of super society intelligent autonomous solutions (e.g., advanced robotics, autonomous cars and drones, or other autonomous systems). The use of Intel's Kapoho Point 8-chip Loihi 2 board technology allows TechWorks' developers to solve larger problems by stacking large-scale workloads and enabling AI models with up to one billion parameters and solving optimization problems with up to eight million variables.

5.1 Computer Systems Organization

Learning Objectives

By the end of this section, you will be able to:

- Define a computer system
- Explain how information is stored and transferred in a computer system
- Differentiate between high-level and machine-level programs
- Identify the elements of a typical computer system

At its core, a computer system is an electronic device that does computations. These computations appear to the outside world as executing programs. When you play a game, listen to a song, or browse the web, you are instructing your computer to do computations. You may wonder how does the computer function and how do the computations performed such as browsing the web or listening to a song relate to one another?

Let us start with the second part of that question: the relationship between computation and executing a program. A song that has been digitized for storage is actually a bunch of numbers that the computer reads; it produces sound based on those numbers. The computer must calculate (i.e., compute) the frequency and volume of the sound based on the numbers read. Another example is when you open a browser and type the address of a website. The computer reads what you typed on the keyboard and leverages network capabilities to translate it to a long number called an Internet Protocol (IP) address (Figure 5.2). The IP address consists of several digits, similar to your phone number, that allow the data to reach your computer over the Internet and for other computers over the Internet to recognize your computer. Your computer passes that IP address over the Internet to another big computer asking for the content of the required website, receives the content from that big computer, and executes browser software that translates it to content such as pictures, sounds, and animations on the screen. All these steps involve computations. But how does the computer do all this? To answer this question, we first need to know the components of a computer system.

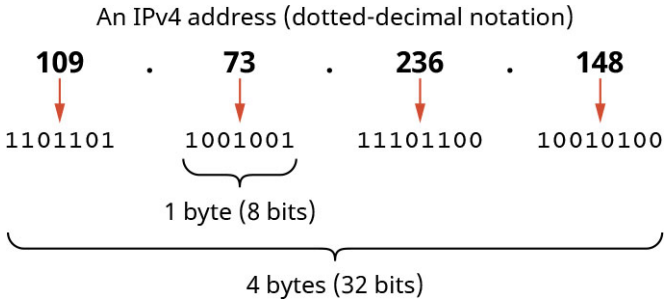


Figure 5.2 IP addresses use a standardized dotted-decimal notation that corresponds to a string of bits. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Computer Systems

A computer system consists of two major parts: hardware and software. The hardware includes the physical components of the computer system, such as hard drives, motherboards, and processor chips. The software consists of the programs you execute, such as media players or web browsers, and the data needed for these

programs to function. For example, the media player is a program, and the songs that you listen to using the media player are data. So, the hardware executes programs that are fed with data. Computers come in different shapes, from the small ones in your smartwatch, for example, to big supercomputers. Different computers may have different components besides the components that exist in all computer shapes. Bigger computers, datacenters, and supercomputers need extra components for getting data from multiple sources, more sophisticated parts to connect to the Internet, and cooling equipment. Mainframes that were created a few decades ago are big computers used for a different purpose. You don't need those in your smartphone.

Data Storage Inside a Computer

All programs and their corresponding data are stored inside a computer or downloaded from the Internet. So how are these programs and data stored? And how does a computer understand and execute them? Computers do not understand English or other natural languages; they can only relate to strings of 1s and 0s and therefore all the programs and data must be stored inside the computer as a sequence of 1s and 0s. A 1 or a 0 is called a binary digit or **bit**. Every 8 bits unit is called a **byte**. This is pretty much how computers that depend on electricity work. Other types, still not in the mainstream market but in design and testing stages, such as quantum computers, do not use bits but use something else called quantum bits or qubits.

If we say that a text editor program takes 1 megabyte of storage, it means that the editor is stored inside the computer, more specifically in the **disk** of your computer, which is a storage mechanism for data, as a sequence of almost 1 million bytes (1 megabyte is a bit larger than 1 million bytes). We call a program stored inside a computer an **executable**. Since programs and data are stored as a sequence of bits or bytes, what is the difference between a program and the data?

THINK IT THROUGH

Why Learn About Computer Systems?

Whether you want to be a software developer, a programmer, or a hardware designer, or even to efficiently use any computer system, you need a minimum knowledge of computer organization. You need to know what the different pieces in the computer systems are, how they interact, and how fast each piece is. Knowing the internal workings of computers helps you write more efficient software, design computers of different sizes (like the one in your watch, the one in your phone, or big supercomputers), and make the best decisions when buying a computer.

Application Programs and Executables

The programs we use in our computer systems, called applications, have been designed and written by software developers using computer languages, also called high-level languages (HLLs). HLLs have been designed to make the interaction between software developers and computers easier. However, computers do not understand HLLs. Computers understand only 1s and 0s. So, there is a set of programs designed to translate the programs written in an HLL into the 1s and 0s that computers can relate to. The programs written by software developers are translated into a set of instructions such as, "Add number 1 to number 2 and save the result as number 3." These instructions are stored as a series of 1s and 0s. Each group of those 1s and 0s represents a single instruction. All the instructions, in their representation as 1s and 0s, are stored in a file called an executable file on the disk.

When you click an icon or type a command, the operating system, software whose job is to manage the interaction between the user of the computer, the hardware, and the programs, loads the executable from storage into the computer memory. At this point, the computer is ready to start executing the program. Figure 5.3 shows the steps for writing programs, in various HLLs by software developers, and then translating these programs to an executable (the 1s and 0s representation of the program) using a special software toolchain

(Figure 5.3). This special software that translates HLLs to executables includes tools called compilers, assemblers, and linkers. We will not discuss these tools here, (refer to 5.2 Computer Levels of Abstraction and 5.4 Machine-Level Program Representation), but we now know what they do. Here, we will look at how the hardware executes instructions.

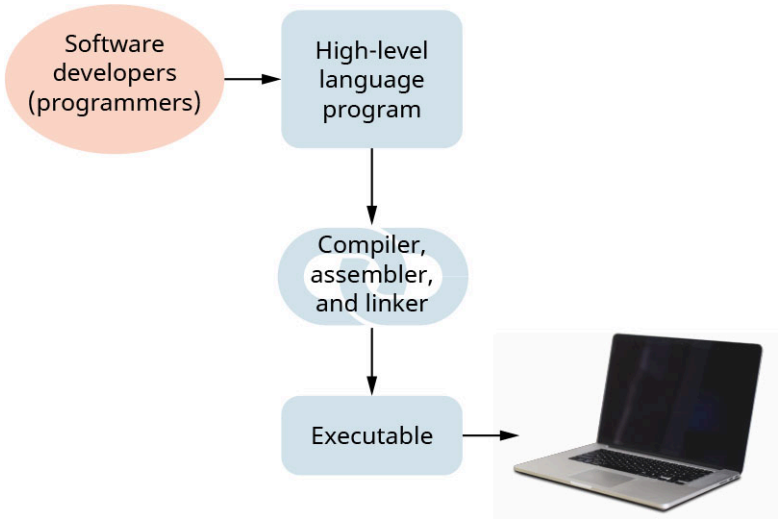


Figure 5.3 Programs developed in high-level languages (HLLs) such as C++, Java, or Python are then translated into executables that a computer can run. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Hardware Organization of Systems

Once the executable is in the memory, the hardware needs to do several things in order to execute it. Recall that an executable is a group of instructions in the form of 1s and 0s. The hardware first needs to fetch an instruction from memory by bringing it from the memory to inside the CPU, which is the brain of the computer that performs and executes the instructions of a program. Once the instruction is fetched from memory and stored in temporary storage inside the CPU, called a register, the CPU decodes this instruction; that is, it deciphers or understands the meaning of the several 1s and 0s that constitute the instruction. For example, the software program may contain an instruction such as "add this number x to the number y and put the result in z." Since computers do not understand English, this instruction is stored inside the computer in the form of 1s and 0s. The CPU has to read those 1s and 0s, understand what they mean (i.e., decode it), and then execute it.

After the instruction has been decoded, the CPU instructs the **arithmetic logic unit (ALU)** to execute it. The ALU is the piece of hardware inside the CPU that performs computations and logical operations such as comparisons. Once the instruction is executed, the result is saved into a register or sent back to the memory. The CPU is now ready to fetch the next instruction. Figure 5.4 shows a computer system's hardware. Inside the CPU we can see the ALU, a group of registers (called a **register file**), and a piece of hardware, called the memory controller, that helps the CPU talk to the memory. Inside the CPU we also find fast storage, which is faster than the memory but slower than the register, called cache memory. We will discuss cache memory in 5.5 Memory Hierarchy.

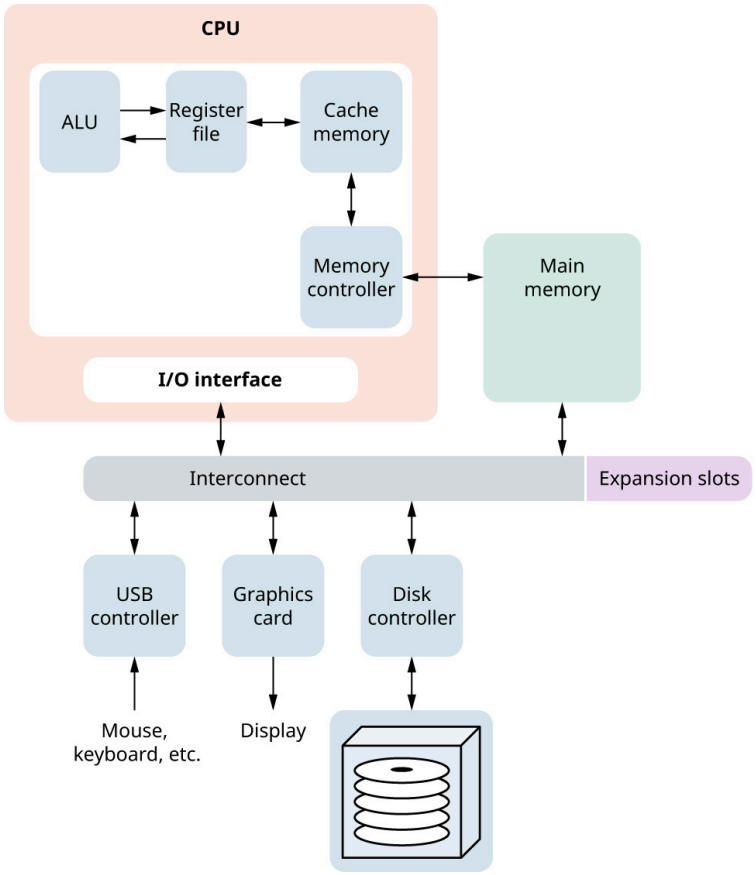


Figure 5.4 Any computer system consists of several components that, together, help the computer do its job of executing programs. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

One final piece of hardware inside the CPU is the I/O interface. The **input/output (I/O)** interface helps the CPU talk to the other I/O devices such as a keyboard or mouse. All the pieces—CPU, memory, graphics card—are connected via a collection of wires referred to as a bus. The purpose of a bus is to transfer data between the various pieces in the computer. Buses can differ in speed, which affects how fast the data are transferred. Figure 5.4 uses the generic word *interconnect* to designate buses. We can add more devices to the computer system such as a different display through expansion slots.

LINK TO LEARNING

Read this concise history of computers starting from the nineteenth century (https://openstax.org/r/76CompHistory19) to see how computers have changed and been developed over time.

Input/Output Devices

The job of input/output (I/O) devices is to take input from a user (i.e., typing on a keyboard or speaking into a microphone), transform it into 1s and 0s, and store this information in memory. It also takes some 1s and 0s, generated by some type of software program, from memory and translates them to an output format such as a picture on the screen or a sound from a speaker. Distinct devices have different speeds and varying ways of transforming the input and output to/from 1s and 0s.

Any I/O device, your keyboard for example, connects to the computer system using an interface such as a USB port that we see in all computers. The USB controller shown in Figure 5.4 is the piece of hardware that manages the USB port and allows it to detect that a device has been connected. Another important piece of hardware is the main memory.

Main Memory

For the CPU to execute programs and process data stored locally, it needs to obtain them from the computer's disk. But a disk is very slow. So, the CPU brings what it needs from the disk and temporarily stores it in faster storage referred to as main memory or random access memory (RAM). When you buy a laptop, one of the specifications is the amount of memory it has, such as 16 GB of RAM or 32 GB of RAM. The main memory is much faster than the disk and connected to the CPU with a faster interconnect, as shown in Figure 5.4. When you click an icon to start a program (e.g., your web browser), the program and its needed data are copied from the disk to the main memory. Then, the CPU reads the data and instructions from the memory into the CPU's registers and the ALU starts executing it.

Processor

The **processor**, also called the microprocessor, is another name for the CPU. It is the brain of the computer system, and its main job is to execute programs. As we discussed earlier, the processor fetches instructions from the memory, understands what each instruction wants to do, gets the data needed to execute the instruction, executes the instruction, and then stores the result in a register or in the main memory. It keeps doing so until the program ends. There have been huge advances in processor design leading to faster and more powerful computers. The processor in your smartphone today is more powerful than a big supercomputer was a few decades ago.

5.2 Computer Levels of Abstraction

Learning Objectives

By the end of this section, you will be able to:

- Describe abstraction levels from the highest to the lowest
- Explain application programs abstractions in relation to HLLs and instruction set architectures
- Discuss processor abstractions and how microarchitecture supports them
- Identify the role of the operating system within abstraction
- Discuss examples of new disruptive computer systems

When you look around, you see that complex systems can be viewed as layers of abstractions. The removal of unimportant elements of a program or computer code that distract from its process is called **abstraction**. This way of looking at complex systems makes it easier to understand them. For example, cars are very complicated inventions. At the highest level of abstraction, we look at a car as a set of devices used to operate it such as a steering wheel, brake and accelerator pedals, and so on. If we go to a lower level, we see devices that power the car such as its engine, gears, and spark plugs. If we take these parts and look at how they are designed, then we are at an even lower level where we see metals, plastics, and other materials. The same approach can be applied to computers to understand how they work. We can see computers as several layers of abstractions, as shown in Figure 5.5. For the remaining part of this section, we start from the highest level and then work through each abstraction layer to illustrate how it is used as a building block by the layer before it.

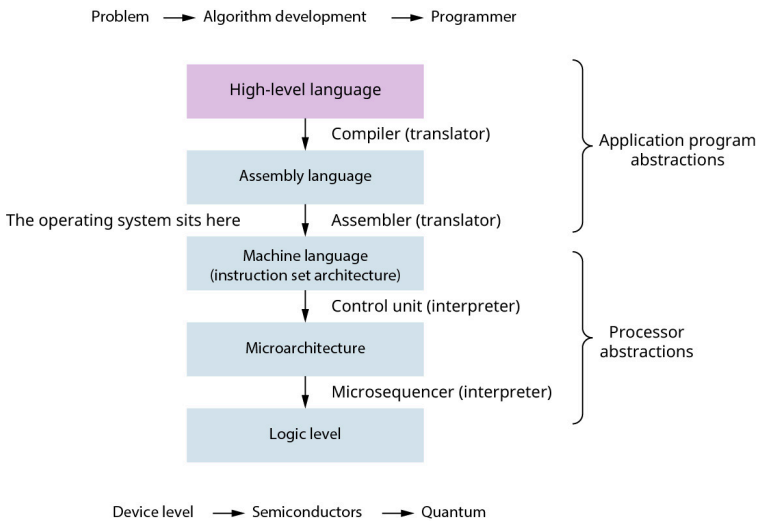


Figure 5.5 A computer system can be viewed at several levels of abstraction, each one layered on top of the other. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Computers are just a tool used to solve a problem. You may use computers to play games or listen to music, and in these cases, the problems that the computer is trying to solve are associated with programs that you use for entertainment purposes.

The top line in Figure 5.5 starts with the problem; we must have a very precise definition of the problem we are trying to solve with a computer. So, the first step in solving a problem with a computer is to know exactly, and with no vagueness, what we are trying to solve. You cannot make a computer solve a problem unless there is a defined and repeatable set of instructions to solve the problem. You may wonder then why it is necessary to use a computer in the first place if you can solve the problem yourself. Well, computers do not get bored, are precise, and can deal with very large problems. This is why, the next step after problem definition is to lay out the steps for solving the problem. This solution layout is called an algorithm. The algorithm is written in free format; that is, it can be steps written as a bulleted list, it can be a flowchart, or it can be a series of mathematical equations. Regardless of the format you choose for writing the algorithm, the algorithm needs to have a key set of characteristics.

The first characteristic is that an algorithm must be unambiguous. Each step of the algorithm must be very well defined and precise. The second characteristic is that the algorithm must be deterministic to be reproducible and repeatable so that the same set of inputs produce the same output. The third characteristic is that the algorithm, when implemented on a computer, must consume a reasonable amount of time and storage based on the problem needs. For example, an algorithm can be finite and precise, but if it requires 100 years to generate a result, it is clearly useless. For instance, an algorithm that counts the number of even numbers is not finite because we have an infinite number of even numbers.

Assuming we have an algorithm, we then need to prepare it for execution on the computer. First, we must prepare the input before it is ready to be consumed by the computer, so we give the algorithm to a programmer whose job it is to read the algorithm, understand it, and then write a program in a known computer language such as C/C++ or Python. The program tells the computer what to do, but in a formal way rather than a freeform way as an algorithm. At that point, we move to another level of abstraction.

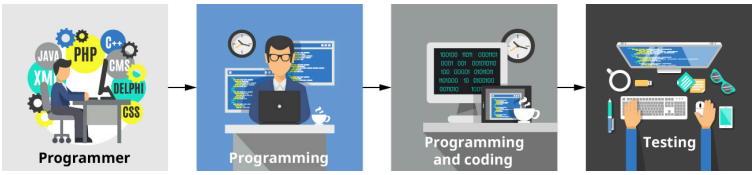


Figure 5.6 The resulting program is consumable by a computer during and after testing. (credit: “Programmer Flat Set” by Macrovector_official/FreePik, CC BY 2.0)

Application Programs Abstractions

Next, the programmer writes a program. The main difference between an algorithm and a program is that the former is written by an algorithm designer and the latter is written by a programmer so that the program can be executed by a machine. These programs are called application programs, or simply, programs, and there are billions of them in existence today. Once the programmer finishes writing the program, there are two more steps before it can be executed by a computer.

High-Level Programming Language

The program generated in the previous step is written in a programming language. There are many programming languages in the world currently. A **high-level language (HLL)** is the most evolved method by which a human can direct a computer on how to perform tasks and applications. The phrase *high-level* means that it is closer to a natural language rather than a machine level language such as strings of 1s and 0s. That is, HLL is more user-friendly, and it is made to make the life of the programmer easier regardless of the hardware or the machine. If you look at the code of a program, you find that it is still in English, yet a restricted version of English with very specific keywords and formats to remove the ambiguity that usually exists in natural human language.

Even though HLLs rely on restrictive versions of English, they still use English at a high level. The machine does not understand English and needs a low-level language; therefore, we need yet another next step: assembly language.

THINK IT THROUGH

One Hundred or One?

Since HLLs aim to make the life of the programmers easy, why do we have many HLLs? Why not one language that all programmers use?

Programming languages are typically designed to help create readable programs. However, some languages are designed with specific applications in mind. That is, some programming languages are easier to use for designing games, while other languages are meant to address mathematical problems or artificial intelligence. However, we can write any program in any language. But our task will be easier if we use the language that is designed with specific applications in mind.

Assembly Language

When you look at most mainstream programming languages, you find constructs such as functions, methods, subroutines, and objects. These concepts were invented to help human beings (the programmers) write their programs after understanding the algorithm. High-level languages make life easier for programmers. By using functions, objects, and other constructs, programmers can write programs faster and make them understandable and reusable for others. Computers, on the other hand, need specific instructions to perform tasks (for example, add this number to that other number and store the result in that place). Writing programs in this way is not easy for programmers because it is error prone, takes a lot of time to write correctly, and is not portable from one computer to the next. So, how do we deal with these conflicting requirements for programmers and computers? The answer is compilers.

A compiler is a piece of software that takes a program written in a given HLL and generates another program that does the same task as the initial program but is written in a language that is much friendlier to the computer called assembly language. Assembly language is then translated into machine language code so that the program can be executed (Figure 5.7).

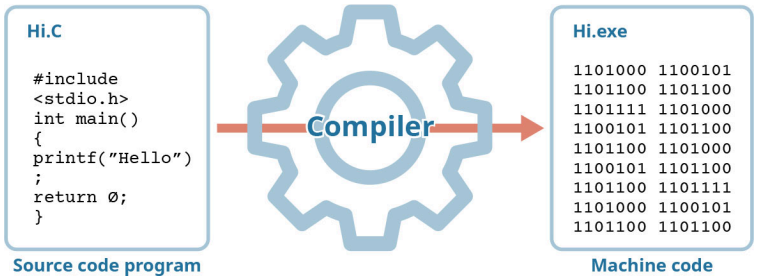


Figure 5.7 A compiler takes code written in an HLL and converts it to "computer-friendly" and simple code called assembly language. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

One low-level language designed to be computer friendly rather than human friendly is called **assembly language**. It does not use all the constructs found in HLLs such as objects or sophisticated data structure, which leads to two challenges.

The first challenge is to manage the output of the compiler, which is an assembly language program that you can open and read. It is basically a text file, so it is written in English. Remember, computers do not understand English; they only relate to 1s and 0s. To deal with this challenge we use yet another program, shown in Figure 5.5, called an assembler (note that compilers may invoke assemblers directly). The **assembler** takes, as input, the assembly program generated by the compiler, and as output, a file that contains the equivalent of that assembly program in terms of 1s and 0s. This generated file is no longer in English, and you cannot open it with your favorite text editor. It is called a machine language file. This is the file that a computer understands. Figure 5.8 shows an example of a program written in an HLL (actually, a middle-level language, which is the C programming language), which is then translated to x86 assembly language and then to binary (refer to Chapter 4 Linguistic Realization of Algorithms: Low-Level Programming Languages).

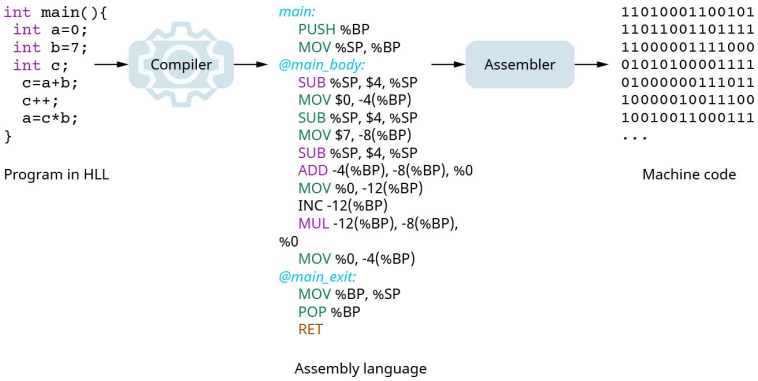


Figure 5.8 Programmers write in programs using HLLs but computers execute binary code, so we need to perform a translation. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The second challenge relates to managing the assembly language itself. If you look at the assembly language program, you find that it consists of instructions such as add, divide, and jump. But are these instructions recognized by all the processors (i.e., the main part of the computer that executes the program) in the world? If you give these instructions to a processor such as Intel, AMD, ARM, Qualcomm, or IBM Power, will they all recognize these instructions? The answer depends on a new concept referred to as the instruction set architecture.

Instruction Set Architecture

The instruction set architecture (ISA) is the set of instructions recognized by each processor family. For example, both Intel and AMD processors use the same ISA, called x86-64, which is different from the ISA recognized by ARM or IBM. This makes us revisit the concept of compilers. A compiler takes as input a program written in an HLL, and we have a compiler for each HLL. The output of the compiler is an assembly program in a specific ISA, and we have a compiler for each different ISA (e.g., ISA 1 and ISA 2). So, if we have programs written in three HLLs and we need to generate assembly for processors of two different families,

then we need six compilers as shown in Table 5.1.

Compiler#	Input to the Compiler	Output from the Compiler
1	HLL 1	ISA 1
2	HLL 2	ISA 2
3	HLL 3	ISA 1
4	HLL 1	ISA 2
5	HLL 2	ISA 1
6	HLL 3	ISA 2

Table 5.1 Compilers and Their I/O

As we saw earlier, the output of the compiler is the input to the assembler so we need assemblers for each ISA in existence to generate a machine language file (e.g., .exe file) that the processor can execute.

Processor Abstractions

As we cross the layer of ISA in Figure 5.5, we cross the boundary between software and hardware. Before we discuss hardware, we need to understand two words: translator and interpreter. Both words mean "translating from language 1 to language 2" regardless of what those languages are. The main difference is the process by which translation is done. A translator takes a whole program in language 1 and generates another program in language 2. For example, the compiler takes an HLL as input (language 1) and generates the corresponding assembly language program (language 2). The interpreter takes one line (or command) in the program in language 1 and generates one (or more) instructions in language 2. Python is a popular example of an interpreted language.

Understanding the hardware level allows us to see how computers execute programs. The main part of the computer hardware that does the execution is called the processor. The processor takes one instruction from the machine language file, executes it, and writes the results back in a designated place. Then, it fetches the following instruction and does the same. It keeps doing this until the program ends or an error occurs. This is an oversimplification, but it conveys the general idea. As you can see, it takes one instruction at a time, which is why the vertical arrow coming out of the machine language box in Figure 5.5 shows the word *interpreter*. But how does the processor do its job of fetching an instruction and executing it? To answer this question, we need to look at the main components of a processor.

LINK TO LEARNING

The transistor is the building block of the hardware of any computer. A transistor is merely an on/off switch; when it is on, it lets the electrical current pass. When it is off, it blocks the electrical current. This is very similar to the light switch you find in your room. But with these transistors, we can do more: with transistors, we build computers. Read this article to learn more about transistors (https://openstax.org/r/76Transistors) and how they work.

Microarchitecture

The architecture (i.e., design) of the microprocessor (i.e., the processor) is called **microarchitecture** (Figure 5.9). Its main job is to design the different components of the processor and decide how to connect them so that the processor can do its job. For example, one important piece inside the processor is the part that fetches the next instruction of the program. Another crucial part, called the control unit, is responsible for decoding, or understanding, what this instruction wants to do and then tells the other components of the processor to execute this instruction. So, the control unit's job is to take an instruction as input and generate signals to control the rest of the processor to make it execute the instructions. Other parts of the processor include the execution units that do the actual computations such as divide, multiply, add, and subtract.

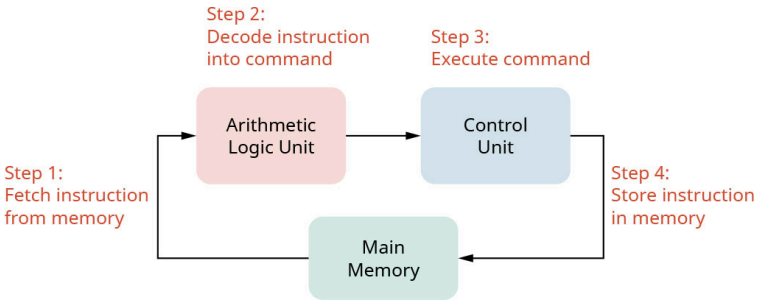


Figure 5.9 Although it may just take seconds to execute, there are several steps included in a microarchitecture process to deliver results for the user. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

To summarize, inside the processor you find those different pieces that fetch instructions, decode them, and execute them. But what is inside each one of these black boxes?

Digital Logic Abstraction

The main building block that forms the processor is called a **logic gate**. There are very few logic gate types: AND, OR, NOT, NAND, NOR, XOR, and XNOR. Using these gates, and most of the time only a subset of them, you can design all the pieces that form the processor discussed earlier. But what is inside these gates? How are they built?

The Lowest Level of Abstraction

The main building block of all logic gates, and hence of all processors, is the transistor. This is shown at the bottom of Figure 5.5 as device level. Though you may have heard the word *transistor* before, you may not know exactly what it does. Simply speaking, a transistor is an on and off switch. This is very similar to the light switch in your house that can turn a light on or off. A **transistor** lets an electrical current pass, the ON state, or can block the electrical current, the OFF state. Transistors are turned ON/OFF based on the voltage input to the transistor. If the voltage is higher than a threshold, the transistor is in the ON state. Otherwise, it is in the OFF state. There are only two states: ON/OFF, which correspond to 1 and 0. This is why computers understand only 1s and 0s. By interconnecting several transistors in some way, we build an AND gate. If we connect them in a different way, we build an OR gate, and so on.

If we try to see how transistors are built and work, then we move to the semiconductor level. At this level we use a special material such as silicon and a special, and very expensive, process to turn it into a working transistor. A single processor contains billions of transistors. How can a material like silicon make a transistor? This takes us to the level of atoms and quantum physics.

The Role of the Operating System

Now that we have rundown the problem definition to quantum physics, you may wonder where the operating system (OS) (i.e., OS X, Windows, Linux) fits in this bigger scheme. The OS is similar to any application program in the sense that it has to be written in an HLL, typically C/C++, and passed through the compiler and assembler to generate machine code. However, the OS differs from traditional application programs in that it

has more privileges in the computer system.

The **operating system (OS)**, shown in Figure 5.10 is the only piece of software that can directly access the hardware. Any other program that needs to access the hardware, such as printing something, must talk to the OS, and the OS achieves the requested task on behalf of the program. The reason all computers are designed this way is to increase security (only one program deals with the hardware so other programs cannot affect the machine) and reliability (a program cannot affect a piece of hardware, which would then affect other programs). In order for the OS to do its job efficiently, it stores the data and programs on disk in an organized way using a file system. The file system helps organize files so that it is easier to find them when needed.

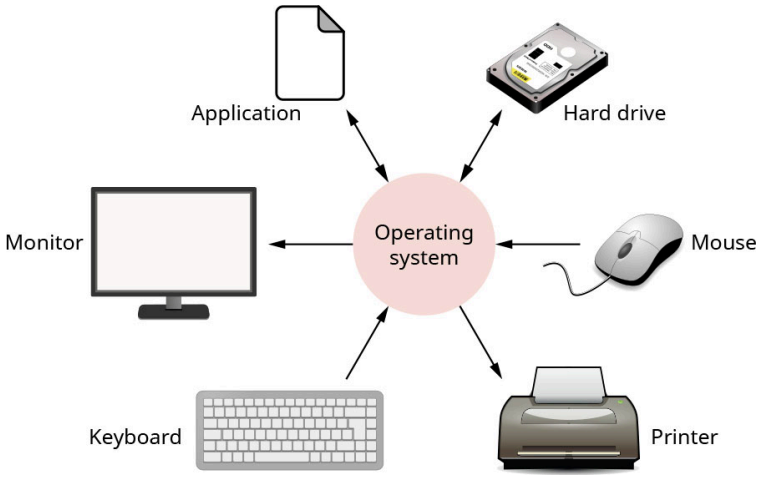


Figure 5.10 The operating system functions as a manager that connects the hardware in the computer to the software. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

A file is a generic name for any entity we want to store in the computer. For example, any program that you use consists of one or more files. Each song you listen to is a file. Any image or video you watch is a file, and so on. The process in which the OS manages which program to use with what part of the hardware at any given time is called **scheduling**. That is, the OS decides when your web browser must use the processor and when your media player uses the screen or speaker. There is a generic name for all programs running on your computer: process. So, if you are listening to music while browsing the web then you are using two processes: your media player and your browser. Therefore, part of the job of the OS is process scheduling.

The technique that the OS uses to isolate different programs from each other so that they do not overwrite each other's data or corrupt each other's files is called **virtual memory**. Additionally, the OS can leverage several computers referred to as virtual machines together to act as one big computer and to serve several uses at the same time. Users may think they have control of the whole machine, even though the reality is not so. This is why this technology used by the OS is called virtual machine.

There are many OSs in the world but the most famous are Windows from Microsoft, macOS from Apple, and Linux.

New Disruptive Computer System Abstractions

Almost all computers in the world are designed in the way we have learned so far and involve very similar levels of abstraction. However, there are very futuristic designs that scientists are tinkering with today that differ from the traditional transistors. Scientists are trying, for example, to build computers using DNA. We have DNA computing, and we have a prototype for DNA storage, too.

We have several prototypes from various companies of quantum computing where instead of using bits, 1 and 0, the machines use quantum bits (qubits) that take a value between 1 and 0. We must not forget that traditional computers in general operate in binary state (i.e., using 0 and 1). To use these computers, we need to build different types of compilers, operating systems, programming languages, and so on. Another form of

a non-traditional computer is a **neuromorphic computer**, which is built to act like a simplified version of the brain. So, it consists of hardware neurons connected together. These computers are not programmed but trained. What will computers look like 100 years from now? We do not know, yet.

5.3 Machine-Level Information Representation

Learning Objectives

By the end of this section, you will be able to:

- Interpret binary numbers
- Explain the use of standard character codes to represent letters and symbols
- Define fractional binary numbers and explain how they are used

In this section we look at two very important and widely used types of information: numbers and text. The reason we concentrate on these two types is because they are standardized, and almost all computers store them in the same format, unlike other types of information that have many different formats, some of which are proprietary.

It is important to know that a series of bits does not have a meaning by itself. For example, in order to interpret or translate the binary number 0011 as a decimal number, you need to know whether it is an unsigned or a signed integer. Therefore, we need to understand how the different types of information are presented in binary.

Integer Numbers Representation

As human beings, we always think in numbers, specifically decimals. Whenever you mention a number, it is usually in decimal form; that is, base-10. But what does base-10 mean? It means the value of the number is calculated by scanning the number from right to left. And as you pass by a digit, multiply it by 10 to the power of the position of that digit. The number at the far right, called the least significant digit, has position 0. The one after is at position 1, and so on. Figure 5.11(a) shows this process.

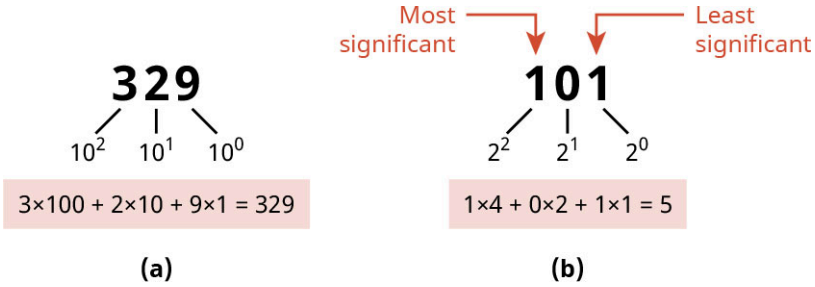


Figure 5.11 The only difference between (a) decimal and (b) binary is the base. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

What about binary numbers? Binary numbers use base-2 and are presented as a series of 1s and 0s. In order to go from binary to the equivalent decimal, we need to differentiate between the ways unsigned integers and signed integers are represented in binary.

LINK TO LEARNING

The best way to get a clear view of how numbers are translated to binary so that computers can deal with them is to see the process in action. Visit this site to convert any number from decimal to binary and vice versa (https://openstax.org/r/76DecimalBinary) and to read more about how to do it manually.

Unsigned Integer Numbers Representation

An **unsigned integer** is a non-negative integer that starts from 0. If you see a number in binary and you are told that this number represents an unsigned number, you can get its decimal equivalent in the same way as base-10 integers but using base-2 instead. Figure 5.11(b) shows the operation of getting the decimal equivalent of the binary number 101 by starting from the right and moving left. The least significant bit has a position 0, the next one has position 1, and so on. As you pass by each element, add $X \times 2^p$, where p is the position of the digit and X is the digit itself (0 or 1), to the total sum.

One important thing to keep in mind is the range of numbers that can be presented by an unsigned number. One bit can present two values only: 1 and 0. Two bits can present four values: 00, 01, 10, and 11 which correspond to 0, 1, 2, and 3. Three bits can present eight values. In general n bits can present 2^n values, which is the range from 0 to $2^n - 1$. As you can see, there are no negative numbers. To be able to present negative numbers, we need to use signed numbers.

Signed Integer Numbers Representation

A **signed integer** is an integer that can be negative or positive. The word *signed* means that they have a sign of + or –. To present negative numbers, designers experimented with several options before picking the de facto choice. The obvious option is to use the most significant bit (i.e., the leftmost bit) as a sign bit where 0 means positive and 1 means negative. The rest of the number is treated with a method called sign-magnitude. Is it a good option? Let us look at the first column of Figure 5.12. It represents the sign-magnitude for a 3-bit number. We see two things.

Sign-Magnitude:	One's Complement	Two's Complement
000 = +0	000 = +0	000 = +0
001 = +1	001 = +1	001 = +1
010 = +2	010 = +2	010 = +2
011 = +3	011 = +3	011 = +3
100 = –0	100 = –3	100 = –4
101 = –1	101 = –2	101 = –3
110 = –2	110 = –1	110 = –2
111 = –3	111 = –0	111 = –1

Figure 5.12 Two's complement is the representation of choice for signed numbers. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

First, there are two presentations of 0 (+0 and –0), which is a waste of data. Second, the numbers are not mirrored around 0. That is, usually, the number after 0 is +1 and the one before 0 is –1. This is not realized here. The importance of this mirror is to make the hardware a bit simpler.

The second method to try is a bit less intuitive. It is called the **one's complement** method of a binary number, which is obtained by flipping each 1 in the original number to 0 and each 0 to 1. For example, 101 has its one's complement as 010. The one's complement method is explained in more detail:

- Check the most significant bit and if it is 1, the number is negative and is represented in its one's complement form. To get the original number, you need to get its one's complement.
- The one's complement of the one's complement is the original number. For example, the one's complement of 101 is 010. The one's complement of 010 is 101.
- Looking at the second column of Figure 5.12, let us take a number such as 110. Its most significant bit is 1 so the number is negative and in its one's complement form. To know its original value, we need to get its one's complement again. The one's complement of 110 is 001 which corresponds to decimal number 1, so the result is –1.
- If the most significant bit is 0, then the number is positive, and to get its decimal equivalent, we treat it as if it is unsigned. With the number 010, the most significant bit is 0, so the number is positive and the decimal equivalent of 010 is 2. Therefore, the result is +2.

Is the one's complement a good method? There is a mirroring effect around the +3/–3. But we still have the two presentations of 0. The third method is called the two's complement, which is the least intuitive method. It is important to note that what is good for machines is not intuitive or easy for human beings!

The **two's complement** of a binary number is simply the one's complement with 1 added to the result. For example, to get the one's complement of 011, we do it in two steps. First, we get the one's complement: 011 is 100. Second, we add 1 to the result: 100 + 1 = 101, and 101 is the two's complement of 011. Now that we know the definition of two's complement, let us see how we can get the decimal equivalent of a binary number. We perform a very similar technique as the one's complement but use the two's complement.

Look at the most significant bit. If it is 0, the number is positive, and the decimal equivalent can be calculated as if the number is unsigned. For example, 011 is positive and the decimal equivalent is +3. If the most significant bit is 1, then the number is negative, and it is written in its two's complement form. Also, the two's complement of the two's complement brings the original number so 101 has the most significant bit of 1. The number is negative and written in its two's complement form.

The two's complement of 101 is 011, which corresponds to 3. The number 101 represents –3, as you can see in the third column of Figure 5.12. A close look at the figure shows that even though the two's complement is not the most intuitive method, it is the most efficient. Since we have only one presentation of 0, we can see from the figure that with three bits, the two's complement method can present up to –4, which we did not find in the other two methods.

Another important issue with the two's complement is that the hardware implementation of it is very efficient because we can use the same piece of hardware for both addition and subtraction and for both signed and unsigned integers. Therefore, a two's complement presentation of a signed number is the standard presentation in all computer systems.

With n bits, the range of numbers that can be presented is $[-2^{n-1}, +2^{n-1} - 1]$. You can see that this range contains 2^n different numbers, exactly like n-bit unsigned integer numbers. The only difference is that in the signed range, half the range is negative, while in an unsigned integer, the whole range is positive.

Lastly, let us look at how additions and subtractions are done. The process is similar to a decimal by starting from the right and a possible carry propagates to the left. Additionally, the subtraction is nothing but an addition; that is, A – B is the same as A + two's complement of B. When we add two numbers in decimal, it is straightforward. With binary, it is also straightforward. Just remember the information in Table 5.2; op1 and op2 are the two inputs to the addition operation.

Op1	Op2	Carry	Sum
0	0	**0**	**0**
0	1	**0**	**1**
1	0	**0**	**1**
1	1	**1**	**1**

Table 5.2 Op1 and Op2 Inputs to the Addition Operation

Let us apply this to numbers longer than one bit. In Figure 5.13, the left side of the example is a traditional decimal addition as done on paper by hand. The right side of the example shows what computers do. Computers, as we already know by now, use only 1s and 0s. So, if they add 1 to 1, for example, the result is not

2 because computers do not know 2. The result of 1 + 1 is 10, which is the binary representation of 2. From this 10, the 0 is the result and 1 is used as carry.



Figure 5.13 This illustrates the difference between traditional addition versus computer computation. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

THINK IT THROUGH

Why Integers?

A real number, represented on a computer as a single precision floating point, takes four bytes. An integer also takes four bytes. Yet, the range of numbers that a floating point can represent is much larger than the integer.

- Why do we use an integer in the first place?

Computers are much slower dealing with floating points because the hardware for a floating point is much more complicated than the one for an integer, and it is better to make things faster by performing computations with an integer. We also only use floating points if it is an absolute need.

Character Representation

Remember that what is stored inside the computer is not only integers. If you look at a keyboard you see a bunch of characters, digits, and symbols and when you press a key, that symbol appears on the screen. But how do computers recognize these characters and deal with them?

Each printable and non-printable character on the keyboard has a unit code or unique binary number. These codes are known as **American Standard Code for Information Interchange (ASCII)**. Capital letters have different codes from lowercase letters. For example, “A” has a different code than “a.” Decimal digits also have their own code. The code is 7-bit length, which covers 128 characters. This encoding has been extended to 8 bits to encompass non-printable characters as well (i.e., characters on the keyboard that cannot be printed on the screen: can you guess them?). The reason to have standardized codes across all machines is to allow computers of different brands, specifications, and sizes to work together. This code was approved in 1963, before personal computers existed, and then revised in 1965, 1967, 1968, 1977, and 1986. Now, all computers use this code to represent the characters.

You may realize that 128 characters, or even 256 ones, cannot include all written languages which is why there are new encoding standards such as Universal Coded Character Set (UCS) and Unicode that encompass all written languages and incorporate ASCII as the first 128 codes, which is backward compatible, allowing for the integration with older legacy systems.

Real Numbers (Floating Points)

There is no computer worth its salt that cannot present and manipulate real numbers. A real number, such as 3.14, is also called a **floating point number** because there is a decimal point somewhere in the middle. In most HLLs, a single precision floating point number requires four bytes of storage such as integers (signed and unsigned).

Let us start with an easy question: if given a binary floating-point number, for example 1010.010101, how do

we get the decimal equivalent? Figure 5.14 illustrates how fractional binary numbers are represented. We use the same technique as getting the decimal equivalent of an unsigned integer. The only difference is that the digits at the right, after the floating point, have negative ranks starting from –1. If we have a number such as 11.111, the rank of the first digit after the floating point is –1, the following one is –2, and the leftmost one is –3. The digits at the left of the floating point have the usual rank that starts from 0. Therefore, the decimal equivalent of $11.111 = 2^1 + 2^0 + 2^{-1} + 2^{-2} + 2^{-3} = 2 + 1 + 0.5 + 0.25 + 0.125 = 3.875$.

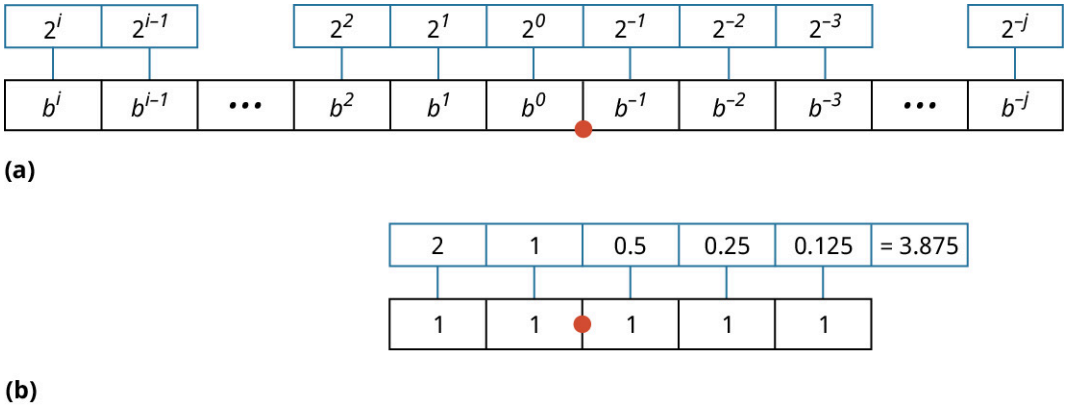


Figure 5.14 Floating point uses the same base 2, but after the decimal point the exponent is negative. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

LINK TO LEARNING

The best way to get a clear view of how numbers are translated to binary so that computers can deal with them is to see the process in action. Visit this site to convert any number from floating point to binary and vice versa (https://openstax.org/r/76FloatPoBinary) and to read more about how to do it manually.

But how do we store numbers like 11.111 inside the computer memory or register? The hard part is the position of the floating point itself because it is not known beforehand and can change during computations, which makes it very hard to decide how many bits to reserve for the digits at the right of the floating point and the digits at the left of the floating points. If we try to make it a fixed number, say 16 bits and 16 bits, we do not get good precision. Going back to the mathematical representation of decimal floating point numbers, you may recall that we can move the floating point to the left or to the right and keep the final value unchanged by multiplying by 10 to some power. For example, 1.875 is the same as 18.75×10^{-1}, which is the same as 1875×10^{-3}, which is the same as 0.1875×10, and so on. In binary, we can do the same by multiplying by 2 to the power of something. The number 11.111 is the same as 111.11×2^{-1} and so on. In fact, we can express any floating point binary number in the form $1.xxxx \times 2^y$. Except for special cases, such as 0, expressing any binary number in this format requires storing three pieces of information:

- The sign bit specifying if the whole number is positive (sign bit of 0) or negative (sign bit of 1)
- The exponent (the y in $1.xxxx \times 2^y$)
- The fraction (the xxxx in $1.xxxx \times 2^y$)

Note we don't need to store the "1." because we know that it exists except for some special cases. And this is why one of the names of the floating point format is "the hidden 1 technique." If we are talking about single precision floating point, it takes 4 bytes (32 bits) and the format is shown in Figure 5.15. One bit is needed for the sign, 8 bits for the exponent, and the rest (32 bits) for the fraction. This format is called **IEEE 754** format, developed by the Institute of Electrical and Electronics Engineers (IEEE). It is the standard format used by almost all computers that support floating points with very few exceptions.

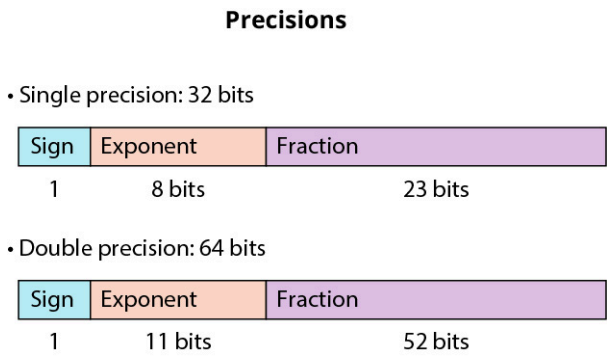


Figure 5.15 IEEE 754 is the standard of choice in most machines to present floating points. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

There is one piece of complexity regarding the exponent—the sign bit is responsible for the sign of the whole number. But, what about the exponent? We need to be able to have positive and negative exponents. If we use the two's complement format for the exponent, the whole floating point format is overly complicated. The hardware that deals with floating point operations is very complicated and much slower than the one dealing with integers, and we do not want to make it even more complicated. The solution is to shift the range of the numbers presented by the 8 bits of the exponent. What does that mean? With 8 bits, we can present numbers from 0 to 256. What if we subtract 127 from each number? That is, you read the 8 bits, get the decimal equivalent (same way as an unsigned integer), then subtract 127. The result will be the range that can be presented is now from –127 to +128. You get the idea.

Another name for the IEEE 754 format is "excess 127." Note that 127 is for the single precision. For double precision that takes 64 bits, we subtract 1023, which is roughly half the range. The double precision is shown at the bottom of Figure 5.15. Let us see an example. What is the decimal equivalent of 1011101101011000...0?

- First, let us divide it into its three main components: sign, exponent, and fraction. This makes it: **1 01110110 10110000...0**.
- Sign bit is 1, so the whole number is negative.
- Exponent = 01110110. Its decimal equivalent is 118. We subtract 127. This makes the exponent: 118 – 127 = –9.
- The rest is the fraction, but we need to add the hidden one, so 1011000...0 becomes 1.1011, which has a decimal equivalent of $1 + 2^{-1} + 2^{-3} + 2^{-4} = 1.6875$.
- This makes the whole number = -1.6875×2^{-9}.

Before we finish our discussion about floating points, there is the question of 0. How to present the 0? Even if we make all bits 0, the hidden 1 makes the final value a non-zero one. How can we deal with this problem? The representation approach that we have learned so far is called the normalized encoding of the IEEE 754 format. This is used if the exponent is non-zero and is not 11111111. If the exponent is 0 (i.e., 00000000) we are in denormalized encoding (also called subnormal). When we are in this special case, there are some differences in the translation to decimal:

- The exponent is 1-bias instead of 0-bias. The bias is 127 in single precision and 1023 in double precision.
- There is no hidden 1, so the fraction part is 0.xxxx (the 23 bits in the fraction in single precision) instead of 1.xxxx....

With these exceptions, we cannot present the 0 (but setting all 32 bits to 0) but can present very small numbers.

The case where the exponent is all 1s is called "special values encoding." If the exponent is all 1s and the fraction is all 0s, it represents infinity. If the exponent is all 1s and the fraction is non-zero, this is called NaN (Not a Number) and raises an exception. This happens when there is a bug in your program that does a division by 0 or the square root of –1, for example.

5.4 Machine-Level Program Representation

Learning Objectives

By the end of this section, you will be able to:

- Discuss x86-64 Intel processors and their architectures
- Differentiate between assembly and machine languages
- Explain basic concepts of assembly language and the types of operations

When you write a program in an HLL, there are several steps that need to be performed before the processor can start executing the code. Refer to Figure 4.18 for a high-level view of the process.

Let us assume you write your program in C and your program is spread over several source files for ease of management. As you now know, the first step is to go through the compiler. The compiler is totally oblivious to the fact that the multiple source code files belong to the same program; it just takes each file separately and generates the corresponding assembly language file for each one of them. If the input is three C files, the output of the compiler will be three assembly language files.

The next step is to take these language files and translate them to machine code files, also known as object files or binary files. Here too, the assembler is oblivious to the fact that the input assembly files do not belong to the same program, so it translates them separately. The first tool in this workflow that recognizes that all the files belong to the same program is the linker. The linker takes all the generated object files, looks for needed libraries, and links everything together into one executable file. Linked libraries are needed because it is very unlikely that programmers write self-contained code. You still use I/O, for example, for printing something on the screen, but you have not implemented those functions yourself—or you use mathematical functions someone else implemented. A library linked at this step is called a static library.

At this stage, you have an executable file residing on your disk until you decide to execute it by typing a command, clicking an icon, or even saying a command. At that moment, a part of the operating system, called the **loader**, loads the executable into the memory and arranges its content in a specific way to make it ready for execution by the processor. A **dynamic library** is when more libraries may be linked during execution or while the program is running.

This section takes a closer look at assembly language. As an example of a widely used assembly language ISA, we will look at x86 ISA used by Intel and AMD processors. But before we delve into this, we need to ask a simple question: Why learn assembly language? Will you ever need to write code in assembly language? Most likely not, except in rare cases where you are developing some part of an operating system, a device driver, or any other application that requires very low-level manipulation. However, by looking at the assembly language generated by the compiler for your code, you can find innovative ways to optimize your code, detect hidden bugs, and reason about the performance of your code once you execute it.

Intel Processors and Related Architectures

You may recall that each processor family understands a set of instructions, which is the ISA. The family of processors from Intel and AMD share the same ISA called x86 (x86-64 for the relatively newer version for later processors). This ISA has a long history that dates back to the 1970s. Figure 5.16 gives a quick glimpse at how things evolved. The figure does not show every single processor from Intel but instead focuses on some milestones.

Name	Transistors	MHz
• **8086** (1978) • First 16-bit processor. Basis for IBM PC & DOS • 1MB address space	29K	5–10
• **386** (1985) • First 16-bit processor, referred to as IA32 • Capable of running Unix	275K	16–33
• **Pentium 4** (1978) • First 64-bit processor, referred to as x86-64	125M	2800–3800
• **Core i7** (1978)	731M	2667–3333
• **Xeon E7** (1978)	2.2B	~2400
• **Core i9** (1978)	2.95B	~3900

Figure 5.16 x86 is a CISC ISA with backward compatibility dating back to the 1970s. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

First, as technology evolves and the need for faster processing power arises, we move from 16, to 32, to 64-bit processors. This number relates to the size of the registers (fast storage entities inside the processor), the width of the buses (parallel wires connecting the processor to memory), and the amount of memory the processor can access (for n-bit machines, the processor can access 2^n bytes of memory). The ISA also evolves in parallel to incorporate the larger registers (hence the move from IA32 to x86-64) and the computations with larger numbers. Second, we can see the tremendous increase in transistors in each generation. Having more transistors means more features implemented inside the processor and hence higher performance and potentially richer ISA.

The complex architecture structure that assists in executing operations such as mathematical computations and memory storage is called **complex instruction set computer (CISC)**. This is done by combining many simple instructions into a single complex one. This concept came from something called the semantic gap, which is the difference between the HLL program and its assembly equivalent. It is good for programmers to understand assembly language as this skill will help you code in any language. However, assembly programming is so much different from HLL programming that most programmers have difficulty understanding it. The wider the difference, the wider the semantic gap. To reduce this gap and make assembly language more accessible to programmers, x86 was designed to make its instructions a bit more complicated because statements in HLL are complicated. Complicated means a single assembly instruction can do several things. For example, an instruction like *addw %rax, (%rbx)* means access the memory at a specific address, get the data stored there, add that data to a number, and store that number in a specific place. So, it is accessing the memory, making an addition, and storing the result somewhere. Because the instructions are complicated, this set of these instructions is called CISC.

Complex instructions such as the ones corresponding to a for-loop in HLL were the norm until the 1980s when another viewpoint came into existence that said that complex instructions make the processor slow. Moving into the 1990s, and the appearance of portable devices with their sensitivity to power consumption and battery life, another disadvantage of CISC arose: complex instructions make the processor not only slow, but also power hungry. And, thus, the other viewpoint of simpler instructions called **reduced instruction set computer (RISC)** came to be the norm. Right now, all the processor families in the world are RISC except x86.

LINK TO LEARNING

There has been a debate among companies who are designing hardware as to whether CISC or RISC is better. Read this article chronicling this debate (https://openstax.org/r/76DebCISCvsRISC) from MicrocontrollerTips.

Assembly and Machine Code

In our discussion of Figure 4.18 we saw assembly (output of the compiler) and **object code**, binary code, and machine code, which all designate output of the assembler. Machine code is the binary presentation of the assembly code. In some cases, there are assembly instructions that do not have a counterpart in the machine code called **pseudo-assembly**. For example, there are instructions in assembly that execute a *go to* if a number is less than another number. The only conditions known in machine code are equal and not equal, but not less than, less or equal, and so on. We can see this in an instruction set like MIPS.

The assembler's job is to ensure that the machine code file only contains instructions that are native to the processor; that is, part of the ISA. So, we can think of the machine code as being a subset of the assembly code. You never find an instruction in the machine code file that is not part of the ISA of the processor for which you want to generate the binary. The reason there are pseudo-assembly instructions is to give the compiler a bit more freedom to generate efficient code. Let us assume that you write a program in C and you think about functions calling each other. If you write a program in C++ or Java, you think in terms of objects, methods, inheritance, and so on (refer to Chapter 4 Linguistic Realization of Algorithms: Low-Level Programming Languages). We call this the programmer view of the language. What if you write (or read) assembly code? What do you see? This view is summarized in Figure 5.17.

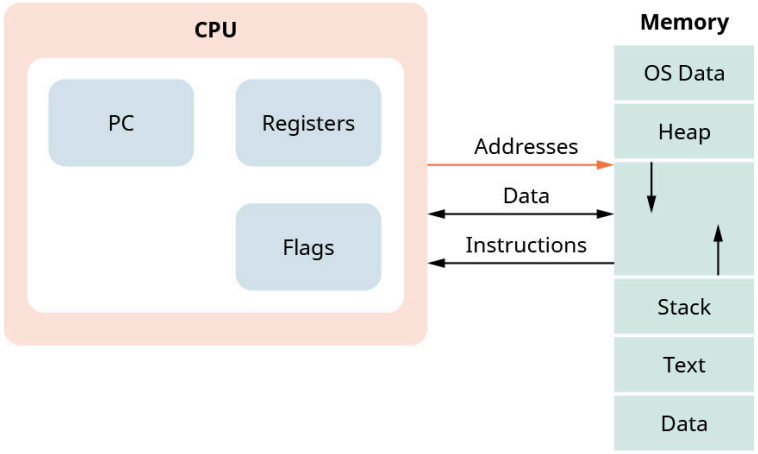


Figure 5.17 The assembly programmer sees a simpler, but more realistic view of the machine than the HLL programmer. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Figure 5.17 shows the following:

- There is a processor (CPU) and memory.
- The CPU and the memory are connected by a **bus**, which is a data pathway. When you access the memory, you may want to get data and must provide an address. Or you may want to write data to memory, so you must provide both the data and the address to which this data will be written. In both cases, the CPU must provide an address, therefore, the address bus is single directional. But the data bus is bidirectional because you can send data to memory or get data from memory.
- Data is not the only thing you need to bring from the memory to the CPU. The main job of the CPU is to execute instructions on data. For example, adding two numbers involves data (the two numbers) and the command for addition (instruction), which is why there is a single directional bus from memory to CPU for getting instructions from the memory.
- The memory holds several things: data, the instructions of the programs (shown as "Text" in Figure 5.17), some data needed by the OS to manage your program, and the resources it needs. The stack and heap are places in the memory to store data depending on the program at hand. The **stack** is used to store local variables (and some other stuff that we will discuss later), the **heap** stores dynamically allocated data, an Data in Figure 5.17 is another area in the memory to store global variables.
- Inside the CPU, there are registers which carry hardware parts that store data, instructions, addresses,

and so on. Each register stores one item. An x86 programmer has access to 16 registers, as we will see shortly. Because the CPU is executing a program, which is a series of instructions, the CPU must keep track of what the next instruction to be executed is.

- Keeping track is the job of a specific register, shown separately from the other registers, called the **program counter (PC)**. The PC is updated after executing each instruction to point to the next instruction to be executed. Also, it is useful to keep some information about the result of the instructions executed, such as whether the result generated by the current instruction is positive, negative, or zero, which is the job of the flags. A **flag** tells a program if a condition has been met.

Registers

A **register** is a memory unit that functions at very high speed. Figure 5.17 shows registers as one black box. If we open this box and see what is inside, we see 16 registers. Any instruction in x86 assembly uses only those 16 registers. The name of each register is shown in Figure 5.18 and starts with an "r." The naming convention of registers is a bit odd, but it is due to some historical naming (for the registers on the left). To keep backward compatibility, old register names cannot be changed.

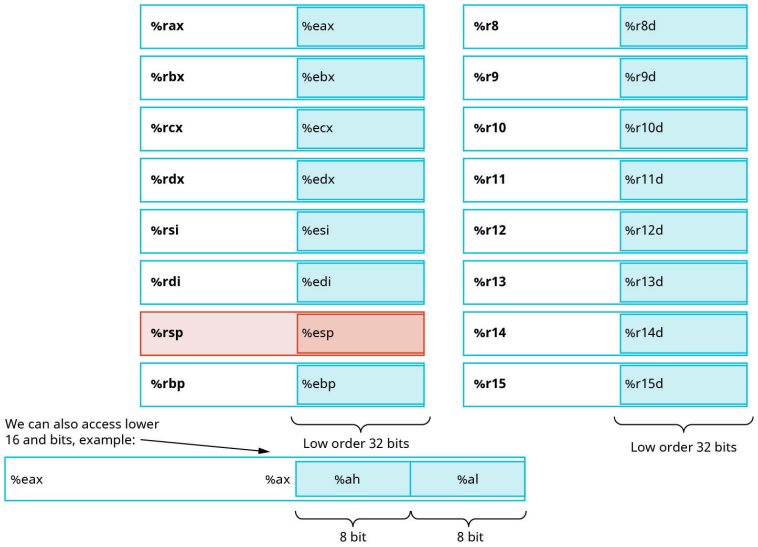


Figure 5.18 Registers in x86 are a bit complicated due to the need to keep backward compatibility and the fact that x86 is CISC. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Each one of these registers can hold 64 bits. When we had 32-bit machines, only eight registers existed, as shown on the left of Figure 5.18. In the 32-bit era, each register could hold 32-bits only and its name started with "e" (for extended). This is why we see that the lower 32 bits of the current registers hold the old names to keep backward compatibility. Not only that, in the 16-bit era, each register held 16 bits only. Those on the left had names: ax, bx, cx, and so on, which are the names of the lowest 16 bits of the current registers. Registers on the right side of the figure did not exist before the 64-bit era. If we go to the 8-bit era, we can even access the lowest 8-bits of the register. The register rax is shown as an example where parts of the register can be accessed using the naming convention: rax (16 bits), eax (32 bits), ax (16 bits), and ah and al (8 bits each). In the 8-bit era there were only four registers, which are the top four in the left column of the figure.

CONCEPTS IN PRACTICE

HLLs and Assembly

Most programmers use HLLs to write their programs. Why do you think world class programmers are very well versed in assembly? Professional programmers write in HLL but they often like to look at the assembly version of their code as well. This allows them to discover mistakes in their HLL code and also to find out

ways to enhance their HLL code.

However, there are some programs, or parts of programs, that need to be written in assembly and not in HLL for the sake of performance (assembly code written by a human in these cases is much faster than HLL translated to assembly by a compiler) and more control over the hardware. Assembly language common uses today include device drivers, low-level embedded systems, and real-time systems.

Operands

An assembly language program consists of a group of instructions. Each instruction does an operation and for this operation to be executed, it needs operands. An **operand** is a value used as input for an operator. Perhaps we want to add two numbers in an operation. To be executed it needs two numbers, which are the operands. For example: *add %RAX, %RBX* means add the content of the register RAX to the content of the register RBX and put the result in register RAX. Operands in assembly can be one of three things:

- A register
- An immediate operand (e.g., add 7 to rax; the operand here is mentioned explicitly in the instruction)
- Data from memory

Memory Addressing Modes

To get data from the memory, we need to specify the location, called the **addressing mode**, in the memory that contains the required data. Why are there several "modes" instead of just specifying an address directly? Well, the answer is related to the HLL.

In HLL programs, we use complicated data structures, such as arrays with one or more dimensions, structures, or linked lists. The compiler needs to translate this data structure to a much simpler assembly language. In assembly there are no complex data structures; there are only data items of 1, 2, 4, and 8 bytes. Then how can we map these complex data structures to the simple single dimension data items? One crucial way is to have rich addressing modes. In its most general form, an address in x86 is specified as D (Rb, Ri, S) where:

- D is a non-negative (but can be 0) integer whose range is from 0 to $2^{32} - 1$.
- Rb and Ri are registers, and they can be any one of the 16 registers.
- S is a scale that takes one of the following values: 1, 2, 4, or 8.

The address is calculated as: D + Rb + (S × Ri). While this is a general form, it can have more reduced forms such as (Rb, Ri). Here D's default is 0 and S's default is 1; D (Rb, Ri); or (Rb, Ri, S).

Now that we know about operands, let's look at the operations themselves that are implemented by the different assembly instructions.

LINK TO LEARNING

It is always good to see the concepts we learn in action. Visit this site to write an HLL program and see the corresponding assembly (https://openstax.org/r/76HLLProgram) at the same time. Any change you make to the HLL code will have an effect on the assembly.

Assembly Operations

In any ISA, all assembly instructions fall into one of three categories:

1. Data movement: from register to register, from memory to register, and from register to memory
2. Arithmetic and logic operations including addition, subtraction, AND, OR, and so on

3. Control instructions, which are the instructions that implement the “go to” operations, whether conditional or non-conditional (category also includes procedure calls)

Data Movement Operations

The data movement in assembly takes the form movx source or destination where:

- “x” specifies the number of bytes to be moved from source to destination. It can take one of the following values: “b” means 1 byte, “w” (word) means 2 bytes, “l” (long) means 4 bytes, and “q” (quad word) means 8 bytes. If you think about it, these are the sizes of all the data types we have in any HLL.
- the source and destination can be any of the operand types we mentioned earlier. There are only three combinations that are not allowed. The first is to move immediate to immediate as it does not make sense. The second is to move from memory to memory because the CPU must be involved. The last prohibited combination is when the destination is immediate as it also does not make any sense. Some examples:
  - “movq %rax, %rbx”
    Move 8 bytes from register rax and put them in register rbx, which is not really a move; it is a copy.
  - “movq (%rax, %rbx, 4), %rcx”
    This is a bit complicated. It involves three steps: first, calculate [rax + (4 × rbx)]; second, use this calculated value as an address and go to the memory at that address; and third, get 8 bytes, starting from the address you calculated (do not forget the “q” at the end of the mov instruction) and put them in register rcx. Now do you see why x86 is CISC where C means complex?

Arithmetic and Logic Operations

The arithmetic and logic operations involve very well-known operations such as add, subtract, multiply, and or, xor, shift left, and shift right. As you may have guessed, these are operations that require two operands. Look at the type of operands that we investigated earlier. For example:

```
addq %rax, %rbx
means rax = rax + rbx.
```

Additionally, there are some one operand operations such as increment and decrement. There are some complexities involved in multiplication and division where there are different instructions for signed and unsigned integers. And, for the division, yet another complexity is where to store the remainder.

Comparison and Test Operations

To be able to implement the complex data flow in HLLs (e.g., switch case or if-else), the assembly language must support conditional and unconditional *go to*. In x86 parlance, it is called a *jump* instruction. The generic form of jump is the jump instruction followed by a label. The **label** is a variable name that we give to an assembly instruction. This is needed because if you want to say (go to this instruction), how can you define “this” instruction? The label takes the following form: label: instruction. For example:

```
part: movq %rax, %rbx
. . . .
jmp part
```

In this example, “part” is the label for the move instruction. The “jmp” is a nonconditional jump; that is, it is always executed. There are some for conditional branches too. Let’s look at one of them.

For the jump if equal, or je, label, this instruction means jump to label if the zero flag is set to 1. In the programmer’s view of the assembly program, there are some flags that give information on the previous instruction (refer to Figure 5.18). One of the flags is called the zero flag, and it is set to 1 if the previous instruction has generated 0 as a result. For example, subtracting two registers and getting the result of 0.

Procedure Call Operations

One last item is related to procedure call—x86 has two instructions: CALL and RET to implement. However, the situation is more complicated than this. In HLLs, you have the concept of local variables and global variables. How is this enforced in assembly? Remember, the assembly program generated must behave in the same way you intended when you wrote the HLL program.

Assembly uses the concept of a stack, the very well-known data structure that works as last-in-first-out, to simulate the notion of local variables, passing arguments to procedure, and saving some of the registers in memory during a procedure call. Why do we need to save some registers? Because we have only 16 registers in x86, and programmers use way more variables than that in their HLL.

Vector Instructions

You may have realized that we have not mentioned floating points at all in the x86 operations. This is because there is another set of instructions and another set of registers for floating points—the **vector instructions**. These instructions operate not on individual registers, but on a vector (i.e., a group of numbers). So, an addition operation can add 32 numbers to another 32 numbers at once. That is, the first number is added to the corresponding first number in the second vector, the second number to second number, and so forth. These operations are usually very efficient in many applications.

5.5 Memory Hierarchy

Learning Objectives

By the end of this section, you will be able to:

- Discuss various memory and storage tools
- Differentiate between various types of storage technologies
- Explain how locality is used to optimize programs

For the processor to do its job, which is doing the calculations, it must be fed instructions and data. This means the overall performance depends on both the calculation's speed and the speed by which data and instructions are received. No matter how fast your processor is, you do not get good performance if the stream of instructions and data is not fast enough. Everything worked well in the early days of computing, from the 1940s until the early 1990s, and then computing hit a wall—a memory wall.

Researchers in industry and academia achieved good leaps in performance for processors by innovating ways to use transistors provided to them by Moore's law. They used those transistors, which were doubling per chip on average every 18 months, to add more features to the processor leading to better processor performance. However, the same was not done with memory, which resulted in a speed up gap between memory and processor. The gap started small and then got wider until it became a bottleneck of performance. Figure 5.19 shows the trend in the processor-memory performance gap.

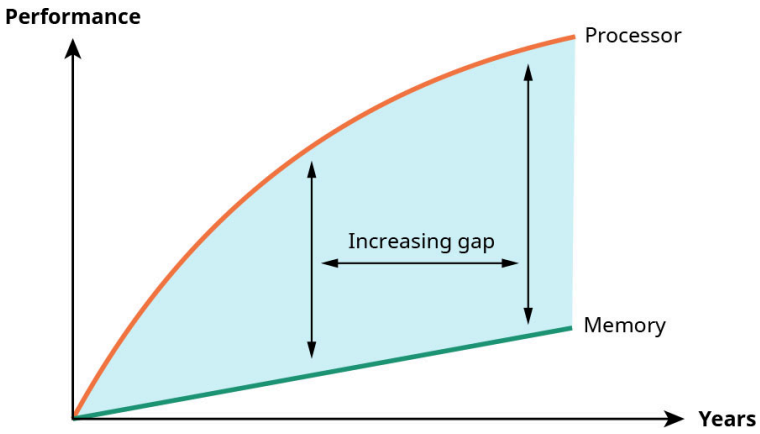


Figure 5.19 The gap between processor speed and memory speed is increasing. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

In this section, we explore memory technology. We will discuss the different technologies by which memory is built, explain what we mean by memory hierarchy, and see what researchers have done to deal with the processor-memory gap.

Memory and Storage

What is our wish list for the perfect memory? Probably speed—we want a fast memory. But be careful—memory speed is different from memory capacity. Memory capacity has increased throughout the years at a much faster rate than memory speed. We also want infinite capacity and persistence; that is, when the power is off, we want the memory to keep its content.

Next, we want to be able to pack a large amount of storage in as small an area as possible via density, which comes in handy especially for portable devices such as your smartwatch or smartphone. And what about the cost? If the memory is very expensive, the whole computer system is very expensive which means that nobody buys it; designers then have to put a smaller memory size into the computer system to keep the price low. But smaller memory means less functionality to the computer system and lower overall performance.

The reality is much less ideal. There is no single technology that excels in all these aspects. Some technologies are fast but more expensive, volatile, and less dense, while others are cheap and persistent but are relatively slower than other technologies. If we pick only one technology, we end up with a non-functional system. For example, if we pick the fast and volatile but expensive technology, the resulting computing system, which is probably very expensive, needs to be powered on indefinitely in order to retain the data. If we pick the persistent and cheap but slow technology, the system may be unusable due to its slow speed.

The word *storage* is usually used with persistence (long-term storage) while the word *memory* is used for volatility (short-term storage) even though this distinction may be blurred in future technologies. How can we get the best of both worlds?

CONCEPTS IN PRACTICE

The More You Know

Knowing about hardware is always beneficial to a software programmer. The cache, for example, is transparent to the programmer; however, if the programmer knows about the cache and how it works, they can write code that exhibits locality and gets good performance.

If you are writing a program that accesses a matrix, and you know how the matrix is stored in memory, you can adjust your code to access the matrix row by row (or column by column) to increase the locality which makes your program much faster.

The Memory Hierarchy

We have five items on our wish list for an ideal memory and, since there is no single technology that excels in all five, we must combine several technologies to come close to the ideal memory system. This ideal memory system must be fast, dense, persistent, large in capacity, and inexpensive. The technologies that we currently use have the following characteristics:

- Technology 1: very fast but expensive, less dense, and volatile
- Technology 2: faster and denser, but volatile and moderately expensive
- Technology 3: persistent and inexpensive but slow
- Technology 4: persistent and very inexpensive but very slow

We need to get the best of all of them, and the best way to combine them must ensure that we have higher capacity from technologies 3 and 4 but make technologies 1 and 2 closer to the processor so that they can respond faster to the processor. The obvious way to do this is to use an arrangement of storage available on a computer system in the form of a triangle as shown in Figure 5.20. We call this design **memory hierarchy**.

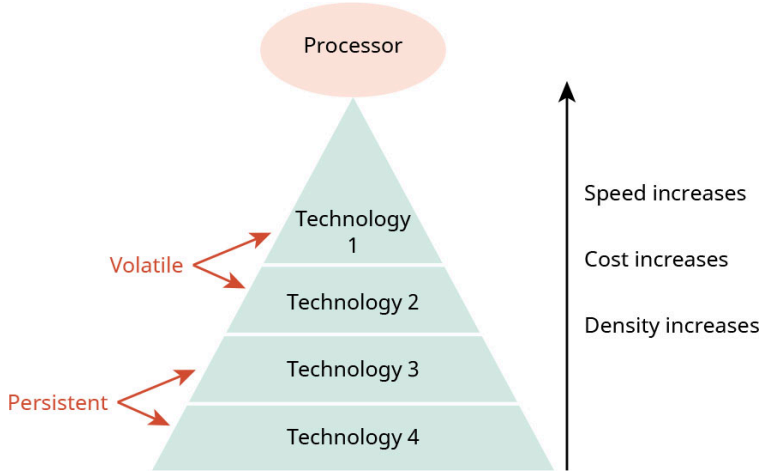


Figure 5.20 Memory hierarchy makes the best use of all technologies. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Table 5.3 gives the names of the technologies that we discuss later in this section. The processor is connected to the first cache using a set of wires called the bus. If the processor does not find what it wants, it goes to the next level, and so on.

	Technology Name	Use Case
Technology 1	Static random access memory (SRAM)	Cache memory • Volatile • Very fast • Expensive • Small in size (from few KB to few MB)
Technology 2	Dynamic random access memory (DRAM)	Main memory • Volatile • Fast • Less expensive • Average in size (1GB to 128GB)

Table 5.3 Technologies Used for Storage in a Typical Computer System

	Technology Name	Use Case
Technology 3	Solid-state drive (SSD)	Storage • Persistent • Slow • Cheap • Big in size (few GB to few TB)
Technology 4	Hard disk drive (HDD)	Storage • Persistent • Very slow • Very cheap • Several TB of storage

Table 5.3 Technologies Used for Storage in a Typical Computer System

Memory Technologies

Memory is volatile, at least for now, with research exploring other technologies for persistent memories, speed, storage, and expense. This covers technologies 1 and 2 and, therefore, they are closer to the processor. Technologies 1 and 2 cover two types of memories that both include **random access memory (RAM)**, which allows the processor to access any part of the memory in any order. Technology 2 is called DRAM, and technology 1 is called SRAM. Let us explore each one in turn.

Memory: DRAM

As you now know, the memory stores instructions and data, which are presented as 1s and 0s. One type of memory, **dynamic random access memory (DRAM)**, consists of a large number of capacitors. A **capacitor** is a very small electrical component that stores an electrical charge. A capacitor can be in one of two states: either it holds a charge, in which case we say that a 1 is stored in this capacitor, or it does not hold a charge, which means a 0. With millions of capacitors, we can store a large number of 1s and 0s. This is what you find in the specs of your laptop; when you say that you have 32GB of RAM, it means there are about 32 billion bytes in memory. Each byte consists of 8 bits. Each bit requires a capacitor.

Capacitors have a not-so-great characteristic though. When a charge is left on a capacitor for some time, the capacitor starts discharging and loses its charge. This means we lose the data stored in memory. Because of this, there is circuitry built inside the DRAM that, every few milliseconds, checks the capacitors and adds a charge to them. We call this the **refresh cycle**. It is done dynamically, hence the name dynamic RAM or DRAM.

The capacitors are not standalone by themselves. Transistors are used with them to help organize those capacitors into rows (also called word lines) and columns (also called bit lines) for addressing specific bits. Figure 5.21 shows a simplified view of DRAM. The cell, which stores 1 bit, is made up of the capacitor and some transistors.

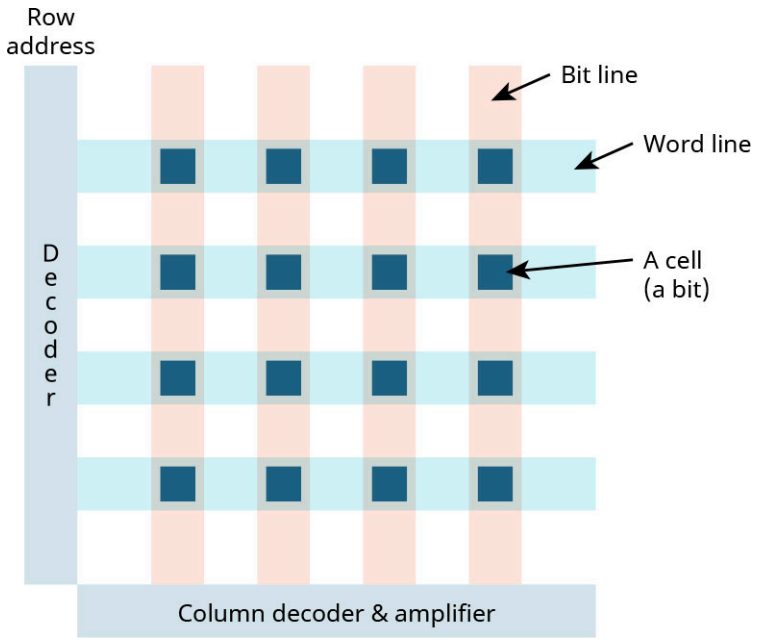


Figure 5.21 This simplified view of DRAM shows one bank. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

What is shown in Figure 5.21 is called a bank. Every few banks form a chip. Every few chips form a rank. A small memory board that contains several memory banks is a **dual in-line memory module (DIMM)**. Several DIMMs form a channel. This organization is shown in Figure 5.22. The reason for having this organization is twofold. First, if we have one huge 2-D array of memory cells, it is too complex, slow, and power hungry, so dividing it into parts makes those parts simpler and that means faster and less power hungry. Second, if there are several memory addresses that need to be accessed and they fall into different banks, for example, the memory can respond in parallel.

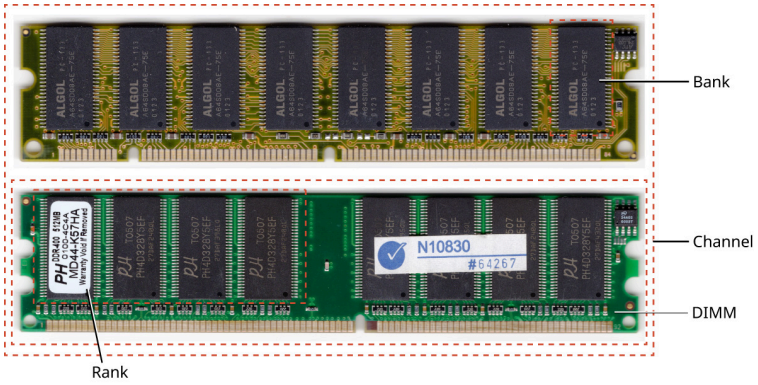


Figure 5.22 DRAM memory banks can be organized into chips, ranks, DIMMs, and channels. (credit: modification of "168 pin and 184 pin DIMM memory modules" by Veeblefetzer/Wikimedia Commons, CC BY 4.0)

You must have heard the term 64-bit machines, right? Most of our computer systems nowadays are 64 bit. One of the definitions of this term is that the connection between the processor and the memory has 64-bit width; its memory can send the data to, or receive data from, the processor in chunks of 64 bits.

The curve that we saw in Figure 5.19 shows the slow speed increase of the DRAM, which affects the performance of the overall system. If the processor must go to the memory for every instruction and every piece of data, the overall system performance is very low; therefore, computer designers speed things up by using a faster technology together with the DRAM. This faster technology, technology 1 in Figure 5.20 is called the SRAM.

Memory: SRAM

There are several reasons for the slow speed of getting the data from DRAM to processor. One is the much slower speed of DRAM technology relative to the processor. The second reason is that going off the chip that contains the processor and to the bus to reach the DRAM memory is a slow process. To overcome this, we need to have a faster memory technology inside the chip together with the processor.

The solution is to use **static random access memory (SRAM)** which keeps data in the computer's memory for as long as the machine is powered on. This means it does not need a refresh like the DRAM and is designed with a faster technology than DRAM. However, SRAMs are bigger in area; a single bit requires a large area in the chip as it needs four to six transistors, which is much larger than the capacitor in DRAM. Moreover, inside the chip we do not have a lot of space due to the existence of the processor itself. So, SRAM is a small, fast memory inside the chip that is connected to the processor from one side and to the DRAM off-chip from the other side, as shown in Figure 5.23. The DRAM is in the range of 8–64GB, while the SRAM starts from the KB range to a few MBs.

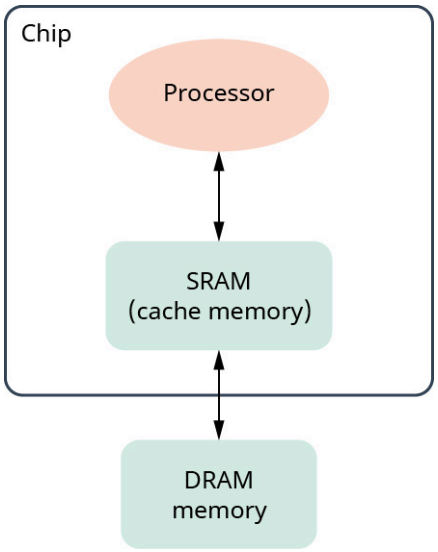


Figure 5.23 SRAM was introduced to overcome the slow speed of DRAM. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

One important distinction between the cache memory and the DRAM is that the former is programmer-transparent (i.e., its use cannot be managed by the programmer), but the latter is not. Your laptop has 32GB of RAM, which is the size of the DRAM, but you may not know how much cache your processor has. However, if you know how the cache works, you can write more efficient programs as we learn when we talk about locality.

In Figure 5.23, SRAM is more commonly referred to as **cache memory**, which is the memory that allows for high-speed retrieval of data. Let us see how the cache and the DRAM memory work together. From now on, whenever we say cache we mean the SRAM, and whenever we say memory, we mean the DRAM.

Suppose the processor executes a program that accesses array A consecutively. The processor starts with A[0] and asks the cache memory whether it has A[0]. Initially the cache is empty, so it does not have the needed data which is called a **cache miss**. The cache then gets the data from the memory. But instead of just getting A[0], it gets A[0], A[1], A[2], ..., A[x]. The number of extra elements the cache brings depends on the design of the cache. In most processors available now, the cache usually brings 64 bytes from the memory. So, if array A is an array of integers, that is, each element is 4 bytes in length, then the cache brings 16 elements from the memory, from A[0] to A[15]. The processor gets the A[0] it wants, and the extra elements brought from memory, which are in the cache. Now, if the processor wants A[1], it finds it right away in the cache, which is called a **cache hit**. However, if the processor instead needs A[17], then we have another cache miss and the cache gets to the memory again to bring several elements, including A[17].

Both the SRAM and DRAM, or cache and memory, are volatile—whenever there is a power perturbation or the machine runs out of battery, everything in the cache and the memory is gone. And we cannot build a full-fledged computer with volatile memory only; we need persistent storage too.

Storage Technologies

Storage exists in computer systems to ensure that data continues to exist even after the computer is powered off. Storage, presented as technologies 3 and 4 in Figure 5.20, has few characteristics that differ from DRAM and SRAM (technologies 1 and 2). The first, and most important one, is that they are persistent—they are non-volatile. The second characteristic is that they are slower in speed than DRAM and SRAM but have lower costs and higher capacities. It is to be noted that technologies 3 and 4 do not have to exist together in a computer system; you can have a computer with either or both.

Besides the storage that exists inside the computer system, there is a lot of storage in the cloud. That is, storage does not exist in your computer, but you can access it through the Internet. This storage is managed by big tech companies. For example, we have Azure from Microsoft, AWS from Amazon, Google Drive from Google, and so on. These companies are serving millions of users and are isolating users' data from each other. There are techniques to make each user access cloud storage and even software in the cloud.

Now, it is time to give them names. Technology 3 is called a solid state disk (SSD) and technology 4 is called hard disk drive (HDD). What are the differences and where does the commonly encountered term "flash drive" fit in? Let us start with the older technology first.

Hard Disk

Hard disks were the main storage solutions for all computers in the 1980s, 1990s, and until the mid-2000s. A **hard disk drive (HDD)** stores data on a rotating platter, has a very large capacity, and uses a small motor to rotate platters to get the data. You can easily buy an 8TB disk for a modest amount of money. The industry took about 25 years to move from 5MB disk to 1TB, and only two years to go from 1TB to 2TB, and, after that, the capacity increased by a whopping 60% per year. So, we have a very ample size with a low price but also a very slow disk. It is several orders of magnitude slower than the DRAM memory. The reason is shown in Figure 5.24. The main reason the disk is slow is due to the mechanical movement. Therefore, computer designers have been looking for storage that does not need mechanical movement drives and that does not have any moving parts.

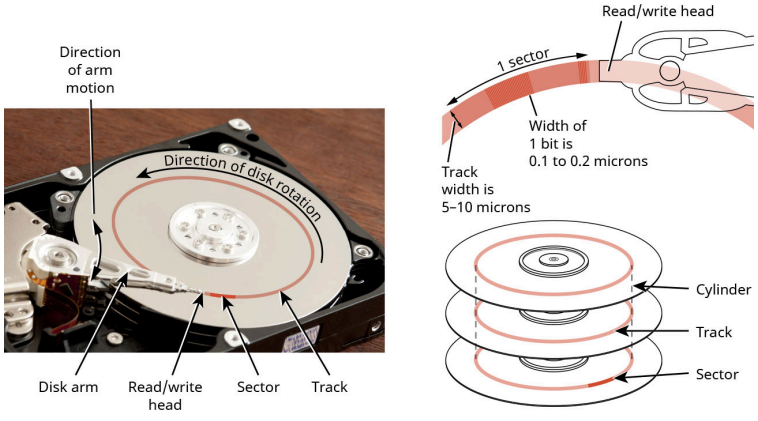


Figure 5.24 (a) The full hard disk drive (HDD) is (b) divided into platters, and each platter is divided into tracks and sectors. (credit a: modification of "Open HDD" by Gratuit/Freeimageslive, CC BY 3.0; credit b: attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Solid-State Drive

The term *solid state* means that a system does not have moving parts and is expected to be fast. A **solid-state drive (SSD)** stores data on a chip and is two to three orders of magnitude faster than HDD. SSD has two main

parts: the storage itself and the circuitry that accesses the storage. Nowadays SSDs use a storage technology called **flash memory**, which is a type of nonvolatile storage that can be electronically erased and reprogrammed. It is what you use in your thumb drives, just a bit faster. Flash memory in SSDs is based on NAND gates. A **NAND gate** is a type of logic gate used to store bits in flash memory. Flash memory is organized as pages, and a group of pages is called a **block**.

The circuitry that controls the flash memory, called the **translation layer**, has an important function. It maps the addresses to pages. One of the disadvantages of storage cells that make a page is that they wear out after 100 thousand to 1 million accesses. So, the translation layer tries to change the mapping to ensure that accesses are equally distributed among different cells. This is a complicated process and is one of the reasons SSDs are more expensive than HDDs. The sequential read from SSD reaches 7000 MB/s and the sequential write reaches 5000 MB/s. Random access to SSD has lower speed for reads and writes.

LINK TO LEARNING

Most laptops now have SSD storage, so it is good to know how SSD really works. Watch this video for a succinct explanation of how SSD works (https://openstax.org/r/76SSDHowItWorks) in the context of smartphones.

More About Cache Memory

As you have learned, cache memory is used as a fast, small memory inside the processor to close the gap between the processor speed and the memory speed. Given that latency is the rate of data transfer, assume DRAM memory's access latency is M cycles and the cache access latency is m. Also assume that for a specific program the probability of cache hits = p. Then the average latency of the combined cache + Memory = $mp + (1 - p)(M + m) = \boldsymbol{m + (1 - p)M}$.

Remember that whenever there is a cache miss, we have already spent m cycles searching the cache, then we go to the memory, which takes another M cycles. If we look at the equation $m + (1 - p)M$, we see that to get good performance, we need to do one or more of the following things:

- Have a faster cache and lower m.
- Increase cache hit rate and reduce p.
- Have a faster memory and lower M.

To have a faster cache, we must make it smaller in capacity, but smaller cache decreases **hit rate**, which is the number, usually a percentage, of times the cache was used to retrieve data. We cannot easily have faster memory to reduce M so the solution is to have more than one level of caches. Level 1 (L1), closest to the processor, is small in capacity and very fast but with a potentially low hit rate. When there is L1 cache miss, instead of going to the memory off-chip, we go to L2 cache, which is still on chip and bigger in capacity than L1. Most processors now have up to three levels of caches, as shown in Figure 5.25.

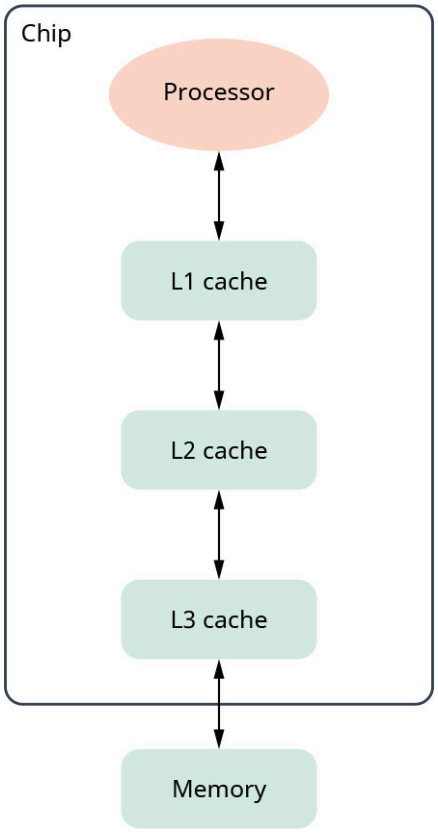


Figure 5.25 Most processors now have three levels of cache memory. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Locality

Throughout our discussion of the memory hierarchy, we have determined the following:

- Memory is read in a chunk of consecutive 64 bits which is 8 bytes.
- Whenever there is a cache miss, the cache works to bring a cache block, not the few bytes the processor needs. Most caches now have a cache block of 64 bytes. Those 64 bytes are consecutive bytes brought from the lower-level cache, that come from the even lower cache (i.e., from L3 to L2 to L1). The L3 cache brought the block from memory.

When a processor repeatedly visits the same memory locales, it has **locality**. We can surmise that if a program accesses the data in a consecutive manner, it gets better performance, called **spatial locality**. Also, if we reuse the data as much as possible, we get better performance because the data is available in the cache, and we can increase the cache hits. We call this second criteria **temporal locality**.

We can get even better performance if the programmer writes efficient code that makes the best use of the underlying hardware. An efficient program has a very important characteristic called locality. For example, if you want to add two arrays together (i.e., *A*[0] + *B*[0], *A*[1] + *B*[1]), then you get a good performance if you access these two arrays sequentially from element 0 until the end of the arrays which is an example of spatial locality. So, whenever you are writing a program, pay close attention to how you access the data.

Instructions also reside in memory. If we have a for-loop that is executed several thousand times, the instructions in the loop body are reused in every iteration which is an example of temporal locality.

5.6 Processor Architectures

Learning Objectives

By the end of this section, you will be able to:

- Discuss the history and advancements in traditional processor architectures and computation models
- Define heterogeneity and discuss its effect on computer systems

Processors have a relatively short history that starts in the late 1960s; however, there have been big jumps since then. In this section, we learn about the evolution of processors from the dawn of processor design until today. The main measures of success of processors involve correctness, speed, power, reliability, and security.

Speed has been the main measure of success for some time. But then computer designers began to worry about battery life (for portable devices) and the electricity bill (for big machines) so power became a pivotal issue. As computers have invaded almost all aspects of our lives, reliability has become a must because we do not want computers to fail. With the widespread use of the Internet and peoples' need to be connected all the time, security has also become an issue.

Homogeneous Processor Architectures

The current processors contain several CPUs inside the chip. If these CPUs are copies of each other, the design is **homogeneous**. If there are different types of CPUs, some are fast but power hungry while others are slow but power efficient, the design is called **heterogeneous**. An example of heterogeneous processors is the chip inside the latest MacBook Pro.

In its earliest version, a processor was just a big, bulky, black box that got its instruction from memory, executed it, got the next instruction, and so on. What was bad about this simple design? First, having one big bulky circuit made it slow and power hungry. Second, not all instructions took the same amount of time; a floating-point computation took as much as ten times longer than an integer computation. But since this design was one box working with one clock, the clock cycle had to be as big as the slowest instruction. The processor executed at the speed of the slowest instruction; therefore, even though the design was simple, it suffered from performance and power issues. However, physics came to the rescue.

CONCEPTS IN PRACTICE

Social Media and Supercomputers

Billions of people use social media sites such as Facebook, X (formerly Twitter), YouTube, and Instagram every day and at the same time. This means we need supercomputers that can serve all these people, store and manipulate huge amounts of data in a short time, and not go down. This cannot be done with a simple multicore; it requires millions of multicores and thousands, if not millions, of accelerators such as GPUs, TPUs, and FPGAs.

Moore's Law and Dennard Scaling

In 1965, Gordon Moore (cofounder of Intel) published a short paper that predicted that the number of devices inside the processor would double every 18 months, called Moore's law. Since transistors are the building blocks of logic gates, and logic gates are the building blocks of pieces such as adders, multipliers, and registers, then more transistors would mean more features implemented in the processor, hopefully leading to better performance. More transistors inside the processor's chip meant that transistors would get smaller in size, and smaller transistors would mean faster transistors. Not only that, but Robert Dennard from IBM found that as transistors got smaller, the power consumed and dissipated was also reduced in a phenomenon known

as Dennard scaling.

Traditional Processor Architectures

Figure 5.26 shows how the processor evolved from the single cycle implementation to more sophisticated and higher performance designs thanks to Moore's law and its enabling technology, Dennard scaling.

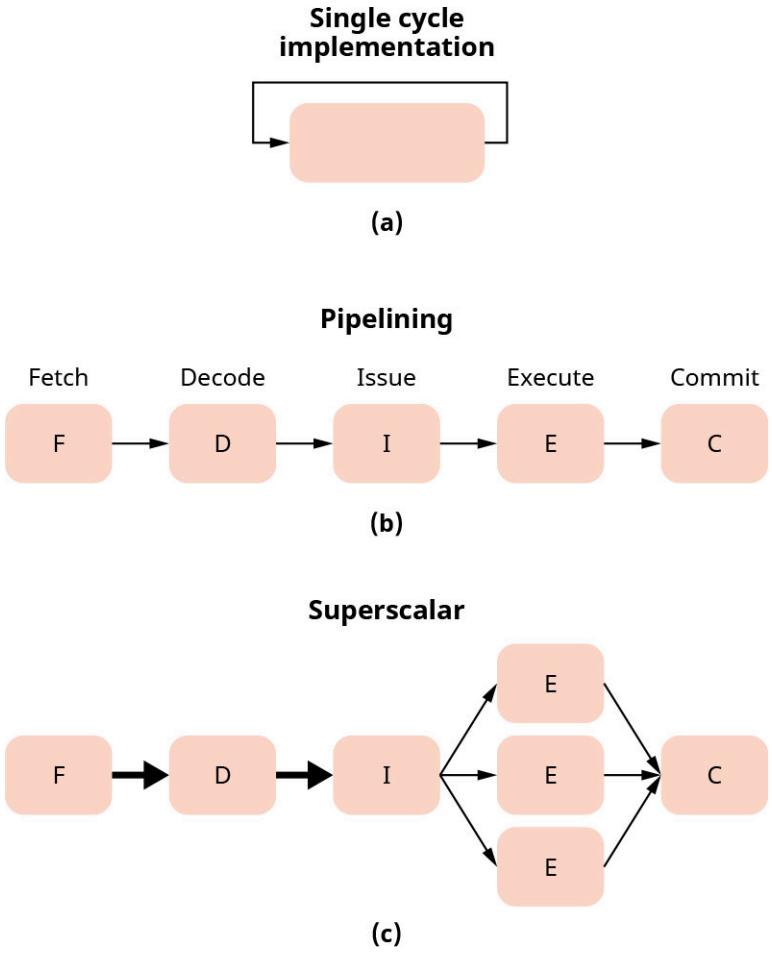


Figure 5.26 The processor has evolved from the (a) simple design single-cycle implementation to (b) pipelining to the very sophisticated (c) superscalar design in less than 70 years. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

If you think about what this bulky box in Figure 5.26(a) really does, you reach the following conclusion: it does few things repetitively with each instruction. It fetches an instruction from memory, decodes this instruction to know what needs to be done, and issues the instruction to the correct execution units (e.g., an integer operation to the integer execution unit and a floating-point operation to floating point execution unit).

After execution, it writes the result of the operation back to a destination specified by the instruction, which is called the commit. Given this description, why not take the bulky piece of hardware in Figure 5.26(a) and divide it into these pieces: fetch, decode, issue, execute, and commit? Each piece does the work needed by the following piece in a technique called **pipelining**, with each piece called a phase (Figure 5.26(b)). What do we gain from this?

First, each phase is now much less complicated, less power hungry, and faster. Second, once the fetch phase finishes fetching the first instruction and hands it to the decode phase, the fetch phase hardware is now free to fetch the second instruction. By the time the decode phase is done with instruction 1 and hands it to the issue phase, the fetch phase hands to it instruction 2 and starts fetching instruction 3, which is a form of parallelism.

If we take a snapshot of the pipeline during the execution of the program, we find different pipeline phases working on different instructions. We call this type of parallelism **temporal parallelism**. The benefit is that this parallelism—or better performance—is done with no involvement from the programmer. The hardware is doing it for you.

With more and more transistors available, designers move to executing several instructions at the same time, which means that several execution units are needed. The other phases must be modified to fetch/decode/issue several instructions at the same time, which is referred to as capability as shown in Figure 5.26(c). Another type of parallelism is **superscalar capability**, which is an execution unit that allows several instructions to be executed at the same time using another type of parallelism, **spatial parallelism**. If you look closely at Figure 5.26(c), you realize it combines both the temporal parallelism (from pipelining) and spatial parallelism (from superscalar capability).

The next step in enhancing a processor's performance is to fetch instructions from more than one thread, and you can technically execute two or more programs on the same processor at the same time. This is called **simultaneous multithreading (SMT)**, which Intel calls hyperthreading technology.

This last enhancement was introduced in the early 2000s, but something happened around 2004 that pushed both the hardware and software communities to change gears. Dennard scaling stopped and designers kept increasing the frequency of processors to make them faster. This led to a complete stall because increasing frequency resulted in drastic increases in power consumption.

Multiple Cores

With the stop of Dennard scaling and the inability to increase the frequency of the processor, it was time for a drastic change. On one chip, instead of putting one processor, designers decided to put multiple processors called cores inside the same chip which started the multicore era. All the processors we use today, from our watches to big supercomputers, are multicore processors.

The trend is to increase the number of cores per chip while decreasing their frequency. Each core in the multicore is an SMT to avoid an increase in power consumption as well as an increase in chip temperature. There is one catch though; all previous techniques (such as pipelining, superscalar, and SMT) were giving us performance without the programmers getting involved. To make use of all the cores in the chip with multicore, programmers must write parallel code that requires the use of a parallel programming language. This is bound to become the norm.

LINK TO LEARNING

Microprocessors (also known as processors) have evolved in the last half century in many ways. Visit this site to learn more about microprocessor trend data (https://openstax.org/r/76CMicroproData) from 1970 to the present.

Heterogeneous Processor Architectures

Multicore processors are designed to be good on average for most applications. However, they do not give the best performance for every single program. Designers have started introducing chips that have excellent performance, better than multicore, but for a small subset of program types. Examples of these chips are graphics processing units (GPUs), field programmable gate arrays (FPGAs), and tensors processing units (TPUs). The idea is to start the program on a multicore until there is a part where other chips excel. In that case, the multicore sends that piece of code to the other chips, and this gives rise to parallel programming for heterogeneous systems (i.e., computer systems that have chips with different capabilities). Laptops can now have a multicore plus GPU.

Multiple Nodes

Having multicore plus accelerator chips (e.g., GPUs) on the same board is now the norm and is called a node. But what about big machines that run in the cloud to give us services such as Amazon, Facebook, and X (formerly Twitter)? These big machines are built using thousands, if not millions, of nodes and they need an even more sophisticated way of programming.

Chapter Review

Key Terms

abstraction removal of unimportant elements of a program or computer code that distract from its process
addressing mode location in the memory that contains the required data
American Standard Code for Information Interchange (ASCII) unit code or unique binary number
arithmetic logic unit (ALU) piece of hardware inside the CPU that performs computations and logical operations such as comparisons
assembler takes the assembly program that takes I/O that contains the equivalent of an assembly program
assembly language low-level language that is designed to be computer friendly
bit binary digit made up of 1 or 0
block group of pages in flash memory
bus data pathway
byte 8 bits
cache hit data found in the cache needed by the processor
cache memory memory that allows for high-speed retrieval of data
cache miss when the cache is empty so it does not have the needed data
capacitor very small electrical component that stores an electric charge
complex instruction set computer (CISC) complex architecture structure that assists in executing such operations as mathematical computations and memory storage
disk storage mechanism for data
dual in-line memory module (DIMM) small memory board that contains several memory banks
dynamic library collection of libraries that may be linked during execution or while the program is running
dynamic random access memory (DRAM) consists of a large number of capacitors
executable program stored inside a computer
flag tells a program if a condition has been met
flash memory type of nonvolatile storage that can be electronically erased and reprogrammed is called
floating point number (also, real number) one with a decimal point in the middle
hard disk drive (HDD) stores data on rotating platter, has a very large capacity, and uses a small motor to rotate platters to get the data
heap stores dynamically allocated data
heterogeneous design feature where CPUs are different in speed, power, or efficiency
high-level language (HLL) most evolved method by which a human can direct a computer on how to perform tasks and applications
hit rate number, usually a percentage, of times the cache was used to retrieve data
homogeneous when CPUs are copies of each other in design
IEEE 754 standard format used by computers that support floating points with very few exceptions
input/output (I/O) interface that helps the CPU talk to other I/O devices such as a keyboard or mouse
instruction set architecture (ISA) set of instructions recognized by each processor family
label variable name that we give to an assembly instruction
library collection of files, functions, or scripts that are cited within a program's code
loader puts the executable into memory and arranges its content in a specific way to make it ready for execution by the processor
locality when a processor repeatedly visits the same memory locales
logic gate main building block that forms the processor
memory hierarchy arrangement of storage available on a computer system usually in the form of a triangle
microarchitecture architecture, or design, of the processor or microprocessor
NAND gate type of logic gate used to store bits in flash memory
neuromorphic computer nontraditional computer built to act like a simplified version of the brain

object code designates output of the assembler
one's complement obtained by flipping each 1 in the original binary number to 0 and each 0 to 1
operand value used as an input for an operator
operating system (OS) only piece of software that can directly access the hardware
pipelining technique where each piece of the process does the work needed by the following piece
processor another name for the CPU
program counter (PC) register that keeps track of instructions to be executed
pseudo-assembly assembly instruction that does not have a counterpart in the machine code
random access memory (RAM) allows the process to access any part of the memory in any order
reduced instruction set computer (RISC) simple instructions
refresh cycle regular operation that DRAM memory does by adding charges to memory cells in order to lose the data stored as charges
register memory unit that functions at a very high speed
register file group of registers
scheduling when the OS manages which program to use with what part of the hardware at any given time
signed integer integer that can be negative or positive (sign of + or –)
simultaneous multithreading (SMT) when two or more programs execute on the processor at the same time
solid-state drive (SSD) stores data on a chip and is two to three orders of magnitude faster than HDD
spatial locality idea that if a program accesses data in a consecutive manner, it gets better performance
spatial parallelism type of parallelism capability
stack used to store local variables
static random access memory (SRAM) keeps data in the computer's memory as long as the machine is powered on
superscalar capability execution unit that allows several instructions to be executed at the same time using spatial parallelism
temporal locality reuses cached data to get better performance
temporal parallelism allows for different pipeline phases to work on different instructions
transistor lets an electric current pass (on state) or blocks it (off state)
translation layer circuitry that controls the flash memory
two's complement one's complement with 1 added
unsigned integer non-negative integer that starts from 0
vector instructions another set of instructions and another set of registers for floating points
virtual memory technique the OS uses to isolate different programs from each other so they do not overwrite each other's data or corrupt each other's files

Summary

5.1 Computer Systems Organization

- The main components of a computer system include the processor, the main memory, the disk, and I/O devices.
- The components interact to execute computer programs efficiently.
- Applications are written in high-level languages that use a program to translate the language into the machine-level programs that computers understand.
- This computer system organization is mostly the same in your tablet or smartphone as it is in the huge systems running services like Facebook or Google. They just differ as it comes to how powerful each piece is, how many CPUs are there, the size of the memory and disk, and so on.

5.2 Computer Levels of Abstraction

- The levels from problem definition to assembly language are related to the computer science field. People studying computer science explore how to solve a problem using an algorithm, how to translate this

algorithm into a programming language, then how to translate this programming language into a language that computers can understand.

- Computer processors operate at various levels of abstraction going from the digital logic level up to the microarchitecture level and the machine language level. The highest level of abstraction for application programs is that obtained by writing programs in a high-level language. Using a compiler makes it possible to generate a representation of such programs at a lower level of abstraction known as assembly language. That representation uses instructions that are part of the instruction set architectures (ISA) specific to the processor family in use.
- The operating system (OS) is the only piece of software that can directly access the hardware of a computer. All other programs must interface with the OS to have it achieve a specific task in a secure fashion by scheduling the corresponding process. The OS stores data and programs on disk in an organized way, using a file system, and allocates memory to programs.
- Upcoming computer designs call for new computing abstractions that will deviate from the traditional binary logic. For example, quantum computing uses qubits and neuromorphic computing uses hardware neurons.

5.3 Machine-Level Information Representation

- The most frequently used data items are stored inside a computer.
- An integer is a number that does not have a floating point; 7 is an integer but 3.14 is not. Integers are divided into two categories: signed and unsigned. Signed integers can be positive, negative, or 0. Unsigned integers are 0 or positive; therefore, they do not need a sign because they will never be negative. This is why we call them unsigned.
- Real numbers, known as floating point numbers, are the numbers that represent fractions. Integers cannot represent numbers like 3.14 or –1.25. This is the role of floating point numbers.
- Characters and symbols on your keyboard are represented inside the computer as 1s and 0s. Every character has its own code.
- The range of numbers that *n*-bit binary number can present if interpreted as a signed or an unsigned integer is helpful to you when you write a program because if you know the values that a variable in your program may take, you can make a precise decision about the type of that variable when you declare it.

5.4 Machine-Level Program Representation

- An x86-64 Intel processor uses a complex instruction set computer architecture (CISC). This type of architecture combines many simple instructions into a simple complex one. Other processor architectures use a reduced instruction set architecture (RISC) based on simpler instructions. RISC V is a relatively new RISC ISA that is getting popular than CISC. All new high-tech companies use RISC instruction sets in their hardware. For example, ARM assembly, which is a RISC assembly, is used in most portable devices.
- Assembly language makes use of processor instructions that are part of the instruction set architecture (ISA) of the processor being used. These instructions can be converted to binary code, referred to as machine language code, using an assembler.
- Assembly language provides specific instruction to perform common operations such as addition and multiplication of signed and unsigned integers.

5.5 Memory Hierarchy

- We must use different technologies to build a near ideal memory because each technology has some good characteristics and a few shortcomings. Therefore, we use multiple technologies to get the best of all. Various types of memory, such as DRAM and SRAM, are available and differ in access speed and associated costs.
- Those technologies are organized as a hierarchy. Technologies that are fast but expensive are at the top of the hierarchy, but their storage capacity is not big. As we go down the hierarchy, the technology is slower, but cheaper; hence, we use a lot of storage from it.
- That is, characteristics of each technology determine its place in the hierarchy.

- By paying attention to locality in accessing the data, the programmer can get the best performance from the hierarchy.

5.6 Processor Architectures

- Processors evolved from a single cycle implementation to multicore.
- The norm now is to have a multicore processor working in tandem with one or more accelerator chips.
- The norm in software development is to write parallel code for such a heterogeneous system. Sequential programming will soon be dated.

Review Questions

1. What is the difference between CPU and ALU?
2. What is the difference between the processor used in your smartphone and the one used in your desktop, tablet, or laptop computer?
3. What is the definition of abstraction?
   a. a piece of software that takes a program written in a given HLL and generates another program
   b. the most evolved method by which a human can direct a computer on how to perform tasks and applications
   c. the set of instructions recognized by each processor family
   d. removal of unimportant elements of a program or computer code that distract from its process
4. What is the term for the technique the OS uses to isolate different programs from each other so they do not overwrite each other's data or corrupt each other's files?
   a. virtual memory
   b. scheduling
   c. neuromorphic computer
   d. assembler
5. Why do computers only work with 1s and 0s?
6. Why is the OS the only software allowed to access the hardware?
7. Do a big supercomputer and small budget laptop use the same levels of abstraction?
8. What type of number would we need to represent the number of marbles in a bag?
   a. signed integer
   b. unsigned integer
   c. floating point number
   d. Boolean number
9. What type of number would we need to represent the temperature outside to the nearest whole number?
   a. signed integer
   b. unsigned integer
   c. floating point number
   d. Boolean number
10. What type of number would we need to represent the amount of money in a bank account?
    a. signed integer
    b. unsigned integer
    c. floating point number
    d. Boolean number
11. What is the two's complement of the binary number 11100101?
    a. 01100101

b. 11100100
c. 00011010
d. 00011011

12. What is the one's complement of the binary number 10100010?
a. 00100010
b. 10100011
c. 01011101
d. 01011110

13. If you see a series of bits, can you know whether they present a signed or an unsigned integer, a floating point, or a character?

14. Why do we need unsigned integers?

15. What is the purpose of a linker?
a. A linker puts the executable into memory and arranges its content in a specific way to make it ready for execution by the processor.
b. A linker tells a program if a condition has been met.
c. A linker takes all the generated object files, looks for needed libraries, and then links everything together in one executable file.
d. A linker is a register that keeps track of instructions to be executed.

16. What is the output of the assembler called?
a. object code
b. source code
c. dynamic library
d. static library

17. What stores dynamically allocated data?
a. stack
b. register
c. bus
d. heap

18. Why do you need to learn assembly programming?

19. What is the main philosophy behind CISC?

20. What is the main philosophy behind RISC?

21. What technology do we use to build a cache?
a. static random access memory (SRAM)
b. NAND gate
c. dynamic random access memory (DRAM)
d. solid state drive (SSD)

22. What technology do we use to build memory?
a. static random-access memory (SRAM)
b. NAND bate
c. dynamic random-access memory (DRAM)
d. solid state drive (SSD)

23. What term means that, when the cache is empty, it does not have the needed data?
a. cache hit
b. cache memory

c. cache miss
d. cache reset

24. Why is it necessary to have multiple levels of cache memory?
a. to make it harder for the CPU to obtain instructions from RAM
b. to make it faster for the CPU to obtain data from RAM
c. just in case one level of cache fails
d. to be able to execute instructions in parallel

25. Is programming a single core or multicore easier? Why?

26. Why did the industry move to multicore?

27. What is the term used to describe when two or more programs execute on the processor at the same time?
a. temporal parallelism
b. spatial parallelism
c. simultaneous multithreading (SMT)
d. superscalar capability

Conceptual Questions

1. Why is the memory connected directly to the CPU?

2. Both the disk and memory store programs and data. Why do we need them both in a computer system?

3. Why do we need an assembler? Why don't we make compilers generate machine language directly?

4. If there are two processors that understand the same ISA, does this mean they have exactly the same microarchitecture?

5. Can the step from algorithms to HLL programs be automated instead of being done by a human being? Justify your answer.

6. The idea of compilers and assemblers made HLL more portable. Why is that?

7. Suppose we have a list of 1,000 numbers ordered in ascending order. We need to find whether a specific number is in the list or not. What is the best algorithm to accomplish this?
a. Scan the list from first number to last number until you find the number you want or reach the last number.
b. Scan the list from last number to first number until you find the number you want or reach the first number.
c. Go to the middle of the list and see whether the number you are looking for is bigger or smaller than the middle number. If it is bigger, discard the lower half. If it is smaller, discard the higher half. Then redo the same in the smaller list.
d. Take a quick look at the list and decide whether the number is present.

8. Why is the two's complement a good choice for presenting signed numbers?

9. As a programmer, do you think it is useful to know about data presentation? Why?

10. Why are computers slower in dealing with floating points than integers?

11. For portable devices, such as your smartphone, do you think processors supporting CISC ISA or RISC ISCA should be used? Why?

12. Is it possible to build a de-compiler? That is, if given an assembly code, can we bring the original HLL code? Explain.

13. Why are the different technologies organized as a hierarchy?
14. What is the relationship between SSD and flash memory?
15. Do you think it is better to have more cores in the chip or to use multiple chips with fewer cores each? Why?
16. Why is heterogeneous computing here to stay?

Practice Exercises

1. Perform a search on the Internet about the specifications of your own computer as well as other similar computers from companies like Lenovo or Dell in terms of memory size and disk size. What is the ratio disk size to memory size in each computer and why?
2. Document how you can obtain the amount of cache memory on your computer.
3. Search the web for five different ISAs. For each one, find which companies are building processors that use that ISA. Finally, for each processor from these companies, check whether this processor is used on your portable device (smartwatch, smartphone, tablet), your laptop or computer, or in big supercomputers and data centers.
4. You are using the Internet to access a website of your choice. Create a diagram showing the various levels of abstraction of a computer system to explain this particular scenario.
5. Research and explain a few key responsibilities of operating systems.

Suppose you wish to express –64 as a two's complement number.

6. What is the minimum number of bits we will need?
7. With this minimum number of bits, what is the largest positive number you can represent, assuming signed numbers of course? (Answer in both decimal and binary.)
8. With that same number of bits you used in the previous questions, what is the largest unsigned number you can represent? (Answer in both decimal and binary.)
9. Represent the decimal digit 90 in binary.
10. Add the following binary numbers: 0111000101 + 0000100101?
11. Find the one's complement of 1010001000011111.
12. Find the two's complement of 000010001111110.
13. Make a list of at least three ISAs in existence today. For each one, find out whether it is CISC or RISC and in which processors it is used. Finally, take the list of these processors and see whether they are used in high-performance computing machines or portable devices.
14. Make a list of three or four accelerators and, for each one, describe the type of applications they excel at and why.

Problem Set A

1. A cache has an access time of 1 cycle. The computer with that cache experienced an average memory access time of 4 cycles, and the hit rate is 70%. What is the access time of the main memory? Did we benefit from having a cache in this system?
2. Research and explain a few key responsibilities of operating systems.

Given $X = 01100110$,

3. What is the value of X once you logically shift X to the right by two digits?
4. What is the value of X once you arithmetically shift X to the right by two digits?

Given $X = 10100110$,

5. What is the value of X once you logically shift X to the right by two digits?
6. What is the value of X once you arithmetically shift X to the right by two digits?
7. What is the value of X if you arithmetically shift it to the right by eight digits?

Perform the following number conversions:

8. hexadecimal value 40A5F916 to binary
9. binary value 11011010011010111010 to hexadecimal
10. Considering the arguments passed in, what does the following assembly language program do?

Register	Use(s)
%rdi	Argument x
%rsi	Argument y
%rax	Return value

movq	%rdi, %rax
subq	%rsi, %rax
movq	%rsi, %rdx
subq	%rdi, %rdx
cmpq	%rsi, %rdi
cmovle	%rdx, %rax
ret	

11. Suppose that we have a system with memory access time of 8 cycles. We need to add to that system a cache with 2 cycles access time. What is the smallest hit rate needed to make that cache beneficial? (Hint: avg access time must always be an integer number not floating point.)
12. A cache has an access time of 1 cycle. The computer with that cache experienced an average memory access time of 4 cycles, and the hit rate is 70%. What is the access time of the main memory? Did we benefit from having a cache in this system?

Problem Set B

1. Suppose that we have a system with memory access time of 8 cycles. We need to add to that system a cache with 2 cycles access time. What is the smallest hit rate needed to make that cache beneficial?
2. Explain how there are multiple levels of abstraction in a computer and explain how at each level, complex implementation knowledge of the lower levels is not needed.
3. Calculate 0 1000 0001 110...0 plus 0 1000 0010 00110...0. Both are single-precision IEEE 754 representations.

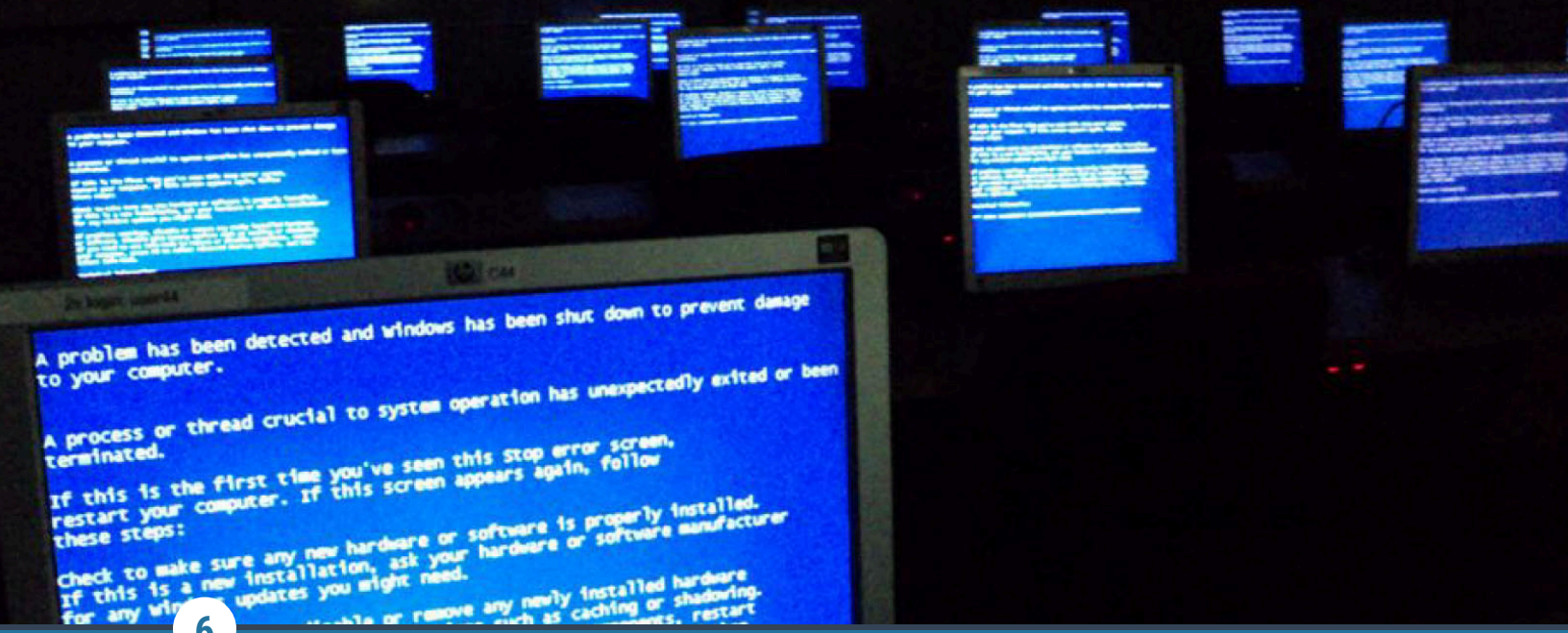
4. We derived the equation $m + (1 - p)M$ for average access latency when we have one cache and memory. Extend this equation to include two-level cache followed by memory.

Thought Provokers

1. What are the desired characteristics of a good computer system, aside from speed?
2. When a company decides to design a new processor to compete in the market, what are the criteria that this company needs to tackle to be competitive? Be careful—speed is not enough.
3. Consider our startup company that is 100% committed to leveraging innovative technologies as a business growth facilitator. Describe at a high-level how it could combine the use of Unicode and Large Language Models to translate documents in any language. What would be some of the limitations of such a solution using existing technology?
4. With several levels of cache, the data in L1 is a subset of the data in L2 which is a subset of L3. This is called the inclusion property. This makes the data a bit redundant and loses chip area to store repetitive data. Do we gain anything from violating this inclusion property? If yes, what do we gain? What are the challenges? If no, why not?
5. Now that Moore's Law is coming to an end, how do you think the computer industry can get more performance from computer systems? Suggest several solutions.

Labs

1. Make a list of the I/O devices on your laptop, and another list of the software programs installed on your laptop. This second list is a bit tricky.
2. Find out the type of processor on your computer or laptop. Once you get the type, find out the following information about it:
   - Which company designed it?
   - How many transistors does it have?
   - What ISA does it use?
3. Using the link provided earlier in this section about translating from HLL to assembly, write a sample program and generate its assembly. Look at the assembly generate, understand it well, and then write yourself another version of this assembly that does the same thing.



6

Infrastructure Abstraction Layer: Operating Systems

Figure 6.1 Operating systems operate on many computers and devices, which means that when they need an update (due to an error or virus), all those systems need the same fix. (credit: modification of "Windows Blue Screen on room full of computers," by Grj23/Wikimedia.com, CC0)

Chapter Outline

6.1 What Is an Operating System?
6.2 Fundamental OS Concepts
6.3 Processes and Concurrency
6.4 Memory Management
6.5 File Systems
6.6 Reliability and Security

Introduction

TechWorks is a start-up company that is 100% committed to leveraging innovative technologies as part of its repeatable business model and as a business growth facilitator. TechWorks has been suffering for many months from the limitations of its operating system, especially the system's lack of security, which allows access to unauthorized and unauthenticated users. This problem reflects poorly on the company's reputation. To address the security limitations, one of the company's project managers decides to implement a new mode of protection that supports all of the operating systems the company's users use, including iOS, Windows, Linux, macOS, and Android. A data scientist working for the company proposes two solutions. The first solution involves integrating an authorization method called two-factor authentication, which you'll learn more about later in the chapter. This method adds another level of security by asking users for their cell phone number so that it can send a one-time code to verify their identity every time they try to log in using their username and password. The second solution involves adding a table that defines individual user privileges such as read (R), write (W), and execute (E). This solution regulates access by leveraging features that were already part of TechWorks's operating system. One of the core features of an operating system is to manage and regulate access to the program components that are running on various machines. This is particularly important today as the program components that power modern solutions (e.g., advanced robotics, autonomous cars, and drones) are typically distributed across many different machines that communicate with each other to perform various functions. To ensure the security of such systems, a single sign-on capability is

typically required to facilitate access to all the components involved.

6.1 What Is an Operating System?

Learning Objectives

By the end of this section, you will be able to:

- Describe an operating system and its role in computing
- Explain the architecture of operating systems

An **operating system (OS)** is the core piece of software that typically manages and controls the interconnection of hardware and software on a computer. The OS is loaded upon start-up and is the key piece of software needed to operate any computerized device.

Introduction

There are many operating systems (OSs), and anyone using a modern computer is using one of them. The typical OSs for computers are Windows, macOS, and Linux and for mobile devices, iOS and Android (Figure 6.2). Microsoft Windows is a popular operating system, celebrated for its ease of use and broad software compatibility. Apple Inc.'s macOS, which is the driving force behind Apple computers, ensures tight hardware-software cohesion. Linux, a freely available open-source OS, is acclaimed for its reliability, safety, and adaptability. Linux source code has over 27 million lines of C, while Windows has over 50 million lines. Linux offers a variety of versions or distributions, such as Ubuntu for ease of use, Fedora for the latest features, Debian for stability, and Kali Linux for security tasks. Each version is designed to meet specific needs, from general computing to specialized applications. All the commercial operating systems today manage security for users.

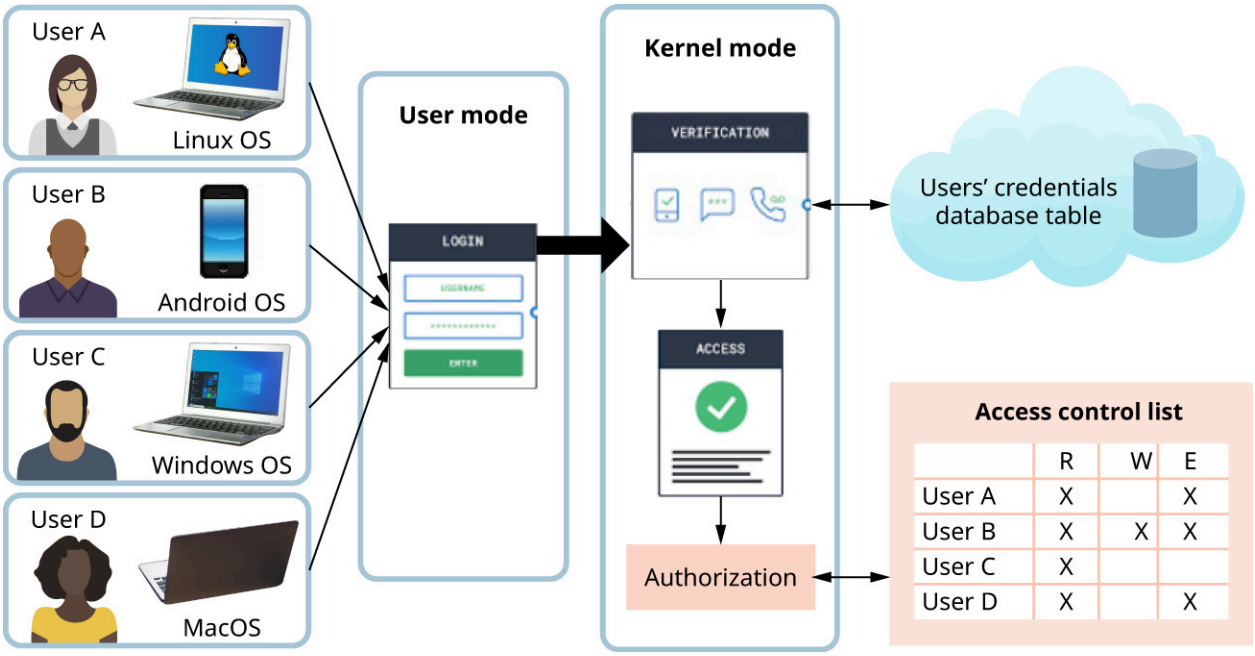


Figure 6.2 These days, organizations, such as companies and universities, must enable a variety of users using a variety of devices supported by a variety of operating systems to securely access their systems. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The goal of this section is to explain what an OS is. Traditionally, we think of the OS as the means by which communication between applications that the user wants to run and the computer hardware that is responsible for running them is facilitated. Applications typically do not directly manipulate the hardware; instead, they must ask the OS to do this on their behalf. One of the primary goals of the OS from an application software point of view is **isolation**. Isolation ensures that the multiple programs that are running

concurrently on the same CPU and memory operate independently without interfering with each other's execution or data. This maintains system stability and security. From a hardware point of view, the hardware is designed to enable programmers to write OSs and applications. In general, the OS mediates a program's access to hardware resources such as the CPU (for computation); memory (for volatile storage) and disk, flash memory, and so on (for persistent storage); TCP/IP stacks, Wi-Fi, and Ethernet network interface cards, and so forth (for network communications); and keyboard, display, touch screen, audio, game controllers, and so on (for input/output). Isolation is what allows device manufacturers to update their hardware without requiring software developers to rewrite their programs each time.

LINK TO LEARNING

To learn more about operating systems, read this informative article on understanding operating systems (https://openstax.org/r/76opsys) from How-To Geek.

Monitoring Access

Operating systems play a crucial role in managing and controlling access to hardware devices within computer systems, including peripherals such as keyboards, mice, hard drives, and other critical components. They act as gatekeepers, ensuring that interactions with these devices occur smoothly and securely. One of the key responsibilities of an OS is monitoring hardware access, which involves tracking and regulating which applications or users can communicate with the hardware and how they do so. This capability is vital for maintaining a system's overall integrity and performance.

In addition to hardware monitoring, the OS is instrumental in enforcing authentication mechanisms. This means the OS helps verify that only authorized users gain access to certain devices, especially those considered sensitive. By managing user permissions and access levels, the OS can prevent unauthorized access to critical hardware resources, safeguarding against potential security breaches or data theft. This aspect of the OS is particularly important in environments where access to information and resources needs to be tightly controlled, such as in corporate or government settings.

The role of an IT administrator is to define which hardware devices are available within a company's network and determine their sensitivity levels. This task requires a deep understanding of both the technical specifications of the hardware and the security implications of its use.

You may be wondering: How is the OS organized? How are resources shared across users? How is one user or process protected from another? Such questions relate to the characteristics or **properties** that are considered when designing an OS. Table 6.1 depicts various OS design questions and the properties they are associated with, such as structure, sharing, naming, protection, security, performance, availability, and reliability, among others.

Property	Question
Structure	How is the OS organized?
Sharing	How are resources shared across users?
Naming	How are resources named, and what is the scope?
Protection	How is one user or process protected from another?

Table 6.1 Operating System Design Considerations

Property	Question
Security	How is the integrity of the OS and its resources ensured?
Performance	How does an OS avoid making all the applications run slowly?
Availability	Can the applications always access the services they need?
Reliability	How often do things go wrong either with the hardware or with a program?
Extensibility	Can new features be added?
Communication	How do programs exchange information, including across a network?
Concurrency	How are simultaneous activities such as computation and I/O created and controlled?
Scale	What happens as demands or resources increase?
Persistence	How can data be made to last longer than program executions?
Distribution	How can a computation be allowed to span hardware, such as machine/network, boundaries?
Accounting	How can a user's resource usage of an OS be tracked, and how might the user be charged for it?
Auditing	Can actions and processes be reconstructed?

Table 6.1 Operating System Design Considerations

Efficiency Management

By managing system resources (e.g., CPU, memory, disk, and network) efficiently, the OS ensures that no resource is underutilized or overburdened. We start with this question: What level of ease or complexity is involved in developing applications with optimal efficiency on a computer? A well-performing operating system facilitates the development of efficient applications that enhance the user experience by providing faster response times. The OS also impacts user experience positively by managing resource allocation and multitasking efficiently.

At some point, you might work for an organization that is engaged in developing a new app, and the company may ask itself this question: If we were to take the time to develop all the additional software required for our application to boot and run on raw hardware, how much faster would it be? This is the same as asking what the penalty or cost is of developing what the OS provides (i.e., sharing of the hardware among apps, and limited damage when programs have defects). The answer to this question relates to runtime efficiency. With respect to coding time efficiency, the OS includes various abstractions, interface and libraries that application programmers can use to ease the burden of software development. Having to write these from scratch without the support of an operating system would greatly increase the required coding effort and extend the development time associated with application development.

Mechanisms Implementation

In operating systems, a **policy** is a way to choose which activities to perform (i.e., what needs to be done) and a **mechanism** is an activity that enforces policies (i.e., how to do it), which often depend on the hardware on which the operating system runs. If a process is granted resources using the first come, first served policy, then that policy may be implemented using a queue of requests. To understand how policies and mechanisms relate, consider this analogy. A car is a mechanism because it enables operations such as going, stopping, and turning that enable a driver to get from point A to point B. Notice that the mechanism (in this case, the car) does not say anything about how to use the mechanism to get anywhere in particular. The driver provides a policy by deciding on a route. Now compare a car with a public bus. In this case, the bus driver determines the policy. The same goes for the self-driving cars that are being developed these days. They serve as the mechanism as there are no decisions about routes being built into cars. Instead, a self-driving car uses machine learning to determine a policy, which includes deciding on a route, selecting the procedure on how to drive, and allowing the driver to override automated driving at any time.

OS-Level and Server Virtualization

The ability of a system or server to run different types of applications used by multiple users at a time on the same computer is called **virtualization**. Server virtualization places a software layer called a **hypervisor** (i.e., virtual machine monitor, or VMM) between a machine's (i.e., server's) hardware and the operating systems that run on it. The hypervisor creates and manages virtual machines. A **virtual machine (VM)** is software that is created to run like a physical computer and that operates its operating system and applications like separate physical servers. OS-level virtualization is a basic form of server virtualization. When using OS-level virtualization, there is no need for a hypervisor as the server's OS handles all resources.

Cloud computing leverages server virtualization and makes it possible for users to commission virtual machines. In this case, the servers reside in distributed data centers provided by cloud vendors. Commissioning a virtual machine is equivalent to gaining access to a virtual computer that runs a chosen operating system while leasing the underlying hardware and only paying per use (i.e., cost of CPU, storage, and network usage).

OS Architecture and Support Layers

As you've now learned, the OS is a system software program whose job it is to manage all the programs running on a computer. OSs enable users to run multiple applications at the same time and keep them from interfering with or crashing each other. OSs provide convenient abstractions to handle diverse hardware. They coordinate resources and protect users from each other using a few critical hardware mechanisms. OSs make it easier for developers to create applications by offering built-in features that help with managing errors. These features include **fault containment**, which prevents errors in one part of an application from affecting the whole system; **fault tolerance**, which allows the application to keep running even when errors occur; and **fault recovery**, which helps the system to fix itself or revert to a previous state after an error. The fact that these standard services are provided by the OS means that developers do not have to build these error management features from scratch, which simplifies the development process. Figure 6.3 shows the support layers within the UNIX/Linux system structure. The OS navigates between three layers: the user mode, where the application resides; the kernel mode, which is in effect sandwiched between the user and hardware; and the hardware, which refers to resources such as the CPU and memory. In the next sections, we will describe these modes and the features they support in detail.

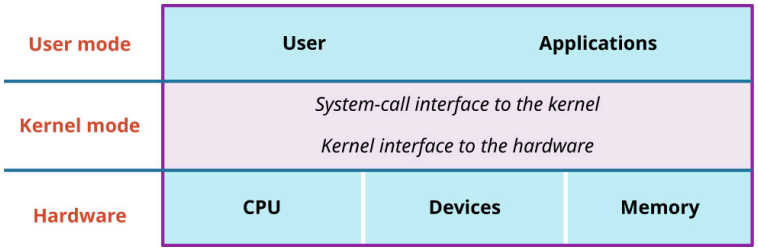


Figure 6.3 The UNIX/Linux system structure is made up of three main parts: (1) the user mode, (2) the kernel mode, and (3) the hardware. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

LINK TO LEARNING

Now that you've been introduced to the basics of operating systems, it may come as no surprise to you that operating systems come in all shapes and sizes. There are many different types of OSs (from batch operating systems to network operating systems), and each type handles many different functions (from file management to device management). This OS tutorial (https://openstax.org/r/76OStutorial) explains the different types. As you read along, consider how much you interact with an OS over the course of a single day—or how many different OSs you interact with.

OS Kernel Features

The kernel is the program that is running at all times on the computer and provides basic services for all parts of the OS. Typically, the kernel is not the only program that is running; there is usually either a system program/operating system or an application program running as well. Any program running on top of the OS is considered to be a **process** from the OS standpoint. It includes the program code, its current activity represented by the program counter, and a set of resources like open files and allocated memory. It also consists of address space, one or more threads of control executing in that address space, and an additional system state associated with it. The operating system uses processes to manage the execution of programs, ensuring they have the necessary resources while keeping them separate from each other to prevent interference.

Software applications have to be compiled and linked with system libraries before they can run as executable programs on a machine. Each running program runs in its own process, and the OS can run, switch, and isolate processes from each other even though they are actually running on the same hardware. For a given application, the "machine" is the process abstraction provided by the OS. Processes provide user-friendly interfaces, rather than raw hardware, and an execution environment with restricted rights controlled by the OS.

An OS virtualizes the machine by providing easy-to-use abstractions of physical resources while masking limitations. In this context, a **thread** is a path of execution within a process, and a process may contain multiple threads. Thus, multithreading involves executing multiple threads (i.e., execution units that are part of a process and share the same resources) concurrently, which improves overall responsiveness and efficiency.

INDUSTRY SPOTLIGHT

Operating Systems and Health Care

In the fast-moving world of IT, it is crucial to think globally. This global perspective can be particularly useful when considering how IT and OSs affect health care. Namely, it can help us understand the role of OSs in patient care, medical research, and the efforts being made to make health care more widely accessible. The challenges associated with data sharing, privacy, and health-care disparities impact regions around the

world differently.

As the delivery of health care becomes more integrated with technology, the choices made in designing how health-care IT systems use OSs can significantly impact people's lives across the globe. These choices range from deciding how to take ethical considerations into account when using AI for diagnosis to creating IT solutions that respect cultural differences.

Check out these initiatives from the World Health Organization (https://openstax.org/r/76WHO) and Centers for Disease Control and Prevention (https://openstax.org/r/76CDC) for examples in the health-care industry.

Imagine you're considering entering the health-care IT field. Can you think of some ways that thinking globally and understanding the capabilities provided by OSs could improve your ability to solve problems and create solutions that work well for health-care systems in various countries?

Hardware Management

The ISA defines how the CPU is controlled by the software by abstracting the hardware details from the applications. The OS provides an abstract machine interface to the application programs and leverages the physical machine interface to do so. The OS communicates with input/output (I/O) hardware using device drivers, I/O ports, interrupts, direct memory access (DMA), and effective I/O scheduling. It provides abstractions to manipulate files (i.e., streams) and send messages to the network (i.e., sockets). Programming languages provide application programming interfaces (APIs) that leverage these abstractions (e.g., file I/O, socket libraries, and related APIs) so that the application program can access the underlying resources that are managed by the OS. One of the main responsibilities of the OS is to isolate hardware from programs by providing common services and background management functionality, for example, storage manager, network manager, and power manager.

The OS is the only system that should be able to directly access I/O devices (i.e., disks, network cards) and manipulate memory. Moreover, the CPU hardware provides a **privileged instruction** that can only be executed by the OS. The OS can use these instructions to establish an execution environment that limits access (to, for example, memory). The application cannot remove the restrictions because it must execute privileged instructions to do so.

Certain operations are prohibited when running in user mode, such as changing the page table pointer (i.e., the pointers to memory pages that are cached for faster access), disabling interrupts (i.e., the interrupts that the OS received from I/O devices), interacting directly with hardware, and writing to kernel memory. Carefully controlled transitions between user mode and kernel mode include system calls, interrupts, and exceptions. The **system call** appears when the program requests a service from the kernel. The **system interrupt** manages the communication between the computer hardware and the system. The system throws an **exception**, which is an error that occurs at runtime.

THINK IT THROUGH

The Ethics of Open-Source OSs

The idea of making all operating systems open-source (i.e., the copyright holder releases the content or product under a license that allows any user to access, modify, and distribute it freely) has both advantages and disadvantages. On the plus side, open-source operating systems can be more secure and innovative because anyone can inspect and improve the code. This openness also encourages a global community of developers to collaborate, potentially leading to a technology that is more user-friendly and accessible for

everyone. Additionally, openness aligns with ethical principles of transparency and freedom, as it allows users to understand and control their digital environments fully.

There are, however, downsides too. Open-source projects might struggle to secure consistent funding and professional support, which can lead to them having slower updates and fixes compared to commercial software. There's also the risk of fragmentation, where too many variations of the system can create compatibility issues and confuse users.

From an ethical standpoint, the use of open-source operating systems globally can democratize access to technology, ensuring that no single company has too much control over our digital lives. It also encourages a culture of sharing and collaboration, which is essential for addressing global challenges like digital divide and ensuring equitable access to technology. However, the success of such a model depends on balancing openness with the need for sustainable development and support systems to keep the technology reliable and up to date.

Should operating systems all be open source? Discuss pros and cons.

Protected Sharing

An OS's functions should guarantee protection, isolation, and the sharing of resources efficiently via resource allocation and communication. To implement protected sharing, the OS provides common services (e.g., sharing and authorization). There are many ways to leverage OS sharing. One involves the sharing of processors to perform computations concurrently. In this case, these computations will be completed as if only one processor had been allocated to them although in reality multiple processors are performing the computations/tasks in parallel. An OS also allows a computer's memory, input and output devices, and files to be shared within the tasks and the processes. It can also allow groups of computers to work together within the network and share resources. All of the sharing capabilities are controlled using secured channels.

6.2 Fundamental OS Concepts

Learning Objectives

By the end of this section, you will be able to:

- Explain various key concepts and components of operating systems
- Discuss the various designs of operating systems

An OS manages computer resources (hardware) and provides services for computer programs (software). The OS works as an interface between the computer user and the system. The OS manages the memory, the files, the hardware, and the software; it also handles the inputs and the outputs such as the keyboard and printer (Figure 6.4).

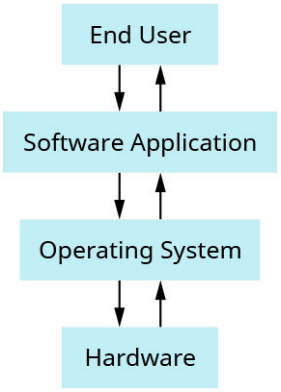


Figure 6.4 The end user initiates this process by using the software/applications, which are built on top of the operating system in

the computer. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

In this module, we study OS components such as process management and threads, memory and address space management, and device drivers and I/O devices. In addition, we cover OS structures such as monolithic OS design, layered OS design, hardware abstraction layer (HAL), and microkernels.

OS Components

OS is a complex system that is typically created using the divide and conquer mechanism. That is, the system is divided into small pieces, each of which defines a part of the system. The OS component's structure is static, and the OS and the hardware are tightly coupled together. The **application programming interface (API)** is a set of rules and tools that allows different software applications to communicate with each other. Applications make requests to the OS through API. The user can, using a keyboard and/or mouse, interact with the OS through the OS interface. The OS interface could be a **graphical user interface (GUI)**, which allows users to interact with electronic devices through graphical icons and visual indicators (e.g., Windows), or a command line (e.g., DOS). For example, in Figure 6.5, the OS interface would be the window into which the Chrome browser opens.

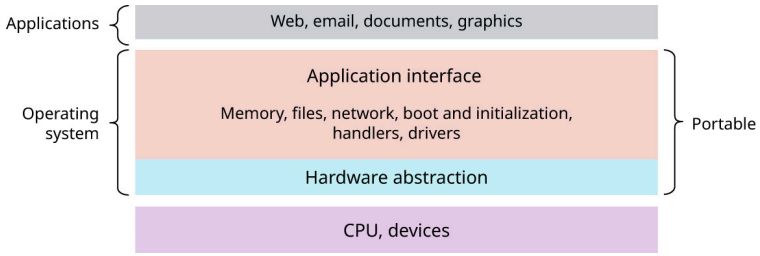


Figure 6.5 The operating system and hardware are tightly linked. The end user is using multiple applications, such as Chrome, Photoshop, Acrobat, and JVM, which they had previously downloaded in a particular OS. To use these software programs, the user will interact through the OS interface and/or API, which is connected with OS components such as file systems, memory managers, process managers, network supports, device drivers, and interrupt handlers. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Process Management and Threads

With a single click, end users seamlessly launch applications, enjoying the simplicity of the process. Have you ever wondered what manages the application once it is launched? The direct answer is the OS. An OS executes many kinds of activities starting from users' programs, background jobs or scripts, to system programs. There are many system programs such as print managers, name servers, and file servers. Each one of these programs or activities is encapsulated within a process. As you learned in the preceding section, a process is a running program that has an execution context plus the runtime instance of the program itself. Examples of an execution context are program counters (PCs), registers, VM maps, OS resources, and examples of runtime instance are code and data.

Here are some other process-related concepts that will be introduced here and developed as the chapter progresses:

- A register is a high-speed memory storing unit.
- A process can be in **running state** (when it is executed by the CPU), **ready state** (when it is waiting in the CPU), or **blocked state** (when it is waiting for an event to occur).
- The OS's process module manages the processes via creation, destruction, and scheduling. One way an OS controls access to a resource is through a data type called a **semaphore**.
- Managing the sharing of a system's resources to avoid interference and errors is called **process synchronization**.

THINK IT THROUGH

Failure Existence

Given the complexity of an OS's tasks (i.e., the management of a computer system's resources and the scheduling of tasks to make an application do what the user expects it to do) and the speed at which these tasks need to happen, it's amazing how often technology *doesn't* fail. But, of course, it does at times.

Let's suppose you ran into an OS-related failure this morning—namely, your laptop did not fully boot. What are some steps you would take to diagnose the problem on your own before seeking help or a repair?

Processes vs. Threads

As we have learned, a process is an active program. A thread is a smaller or lightweight portion of a process that is managed independently. Table 6.2 shows a comparison between the process and the thread.

Attribute	Process	Thread
Definition	An executed program	Part of the process
Weight	It could be heavy	Lightweight
Processing time	More time	Less time
Resources	Needs more resources	Needs fewer resources
Sharing	Mostly isolated	Shares memory and data

Table 6.2 Process vs Thread

Processing

A program is passive; it is just bytes on a disk that encode instructions to be run. A process is an instance of a program being executed—or processed—by a processor. Thus, processing involves a program, a process, and a processor. The processor can be a real processor or a virtual processor (i.e., CPU core assigned to a virtual machine). At any time, there may be many processes running copies of the same program (e.g., an editor); each process is separate and usually independent. An OS is responsible for managing these processes.

Different OSs approach process management in different ways. For example, the Windows operating system's approach consists of adding an operating system call to create a process and other system calls for process operation. The approach used in the UNIX OS is different from this as it consists of splitting the process into two steps using fork and exec functionalities. The fork functionality is used to set up privileges by creating a complete copy of the process, and the exec functionality brings the executable file into memory to start the execution.

Address Space and Memory Space Management

The **address space** is the set of addresses generated by programs as they reference instructions and data. The memory space holds the actual main memory locations that are directly addressable for processing. The OS is responsible for managing these memory and address spaces (Figure 6.6). To enhance performance, computers use virtual memory address space to create the illusion of a large and continuous block of memory for applications and the operating system to utilize. To do so, the computer's physical memory is used in

combination with a portion of a hard drive that contains the swap file or page file. Pages containing address space information for programs are moved in and out of physical memory as necessary to ensure that there is enough physical memory to hold the pages of programs that are running at a given time.

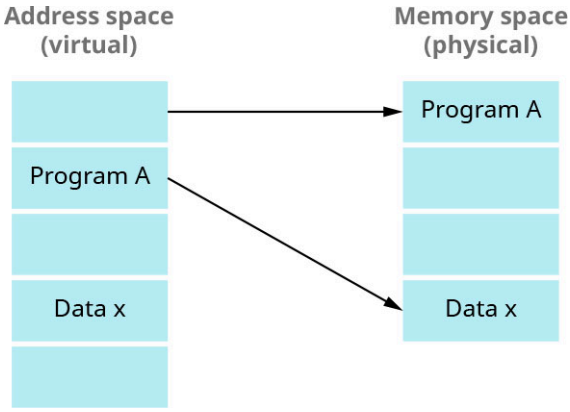


Figure 6.6 The addresses of the data will be in the address space waiting for the execution to be moved to the memory space. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Computer memory consists of two main types: primary and secondary memory. The initial point of access for a processor that also serves as direct storage for the CPU is called **primary memory**. To be executed, programs must reside in primary memory. In multiprocessor systems, primary memory can be classified into three architectures. The first, **uniform memory access (UMA)**, employs a single memory controller and, thus, access time to any memory location is the same across all processors. The second, **non-uniform memory access (NUMA)**, is a computer memory design that uses different memory controllers, thus its memory access time varies depending on the memory's location relative to a processor. The third, **cache-only memory architecture (COMA)**, uses multiple interconnected processing nodes, each equipped with a processor (i.e., a cache). This allows for the dynamic allocation of data for optimized access and performance in multiprocessor environments.

Memory that is used for long-term storage, housing the operating system, applications, and data that need to persist even when the power is off, is called **secondary memory**. Unlike primary memory, which is volatile and loses data during power interruptions, secondary memory is nonvolatile—meaning it retains data even during power failures—and thus it provides durability and data retention. Common examples of secondary memory include hard disk drives (HDDs) and solid-state drives (SSDs).

An OS must satisfy the policies of how much physical memory to allocate to each process and when to remove a process from memory. It must implement these policies using the following mechanisms: **memory allocation** and **memory deallocation**. Memory allocation is the process of setting aside sections of memory in a program to be used to store variables and instances of structures and classes. The memory allocation can be static memory allocation or dynamic memory allocation. Memory deallocation is the process of freeing the space corresponding to finished processes when that space is needed by the rest of the system. In addition, the OS must maintain mappings from virtual addresses to physical (i.e., page tables) and switch CPU context among address spaces.

INDUSTRY SPOTLIGHT

Windowing Systems

A windowing system is an OS software component that manages the display of graphical user interfaces (GUIs) on a computer screen. An investigation into the influence of windowing systems on various industries, particularly in sectors like retail marketing, has yielded insightful perspectives on how such systems impact business practices, productivity, customer engagement, and overall operational

effectiveness. Namely, it's been found that windowing systems significantly enhance user experience, which is critical to retail marketing. By making it easier to integrate marketing messages or interruptions with other media that customers are watching, a windowing system helps retailers conduct targeted marketing over the Internet. Windowing systems also improve the overall user experience for customers viewing the integrated content.

Device Drivers and I/O Devices

Computers have many input and output devices such as the keyboard, mouse, display, or USB port. Some examples of OS-specific devices include file system (disk), sockets (network), and frame buffer (video). A **frame buffer** is a portion of random access memory (RAM) containing a bitmap that drives a video display. A big chunk of the OS kernel deals with I/O (Input/Output). The OS provides a standard interface between programs/users and devices to communicate with them (Figure 6.7). A device driver's routines interact directly with specific device types and related hardware to initialize the device, request I/O, and respond to interrupts or errors. An **interrupt** is a signal to the processor from either software or hardware that indicate events that needs immediate attention. Examples of device drivers include Ethernet card drivers, video card drivers, sound card drivers, and PCIe (Peripheral Component Interconnect Express) device drivers, which are associated with graphics cards and other peripherals. Device drivers are implemented by device manufacturers or open-source contributors and support a standard, internal interface. They can execute in the OS address space and run at high privilege.

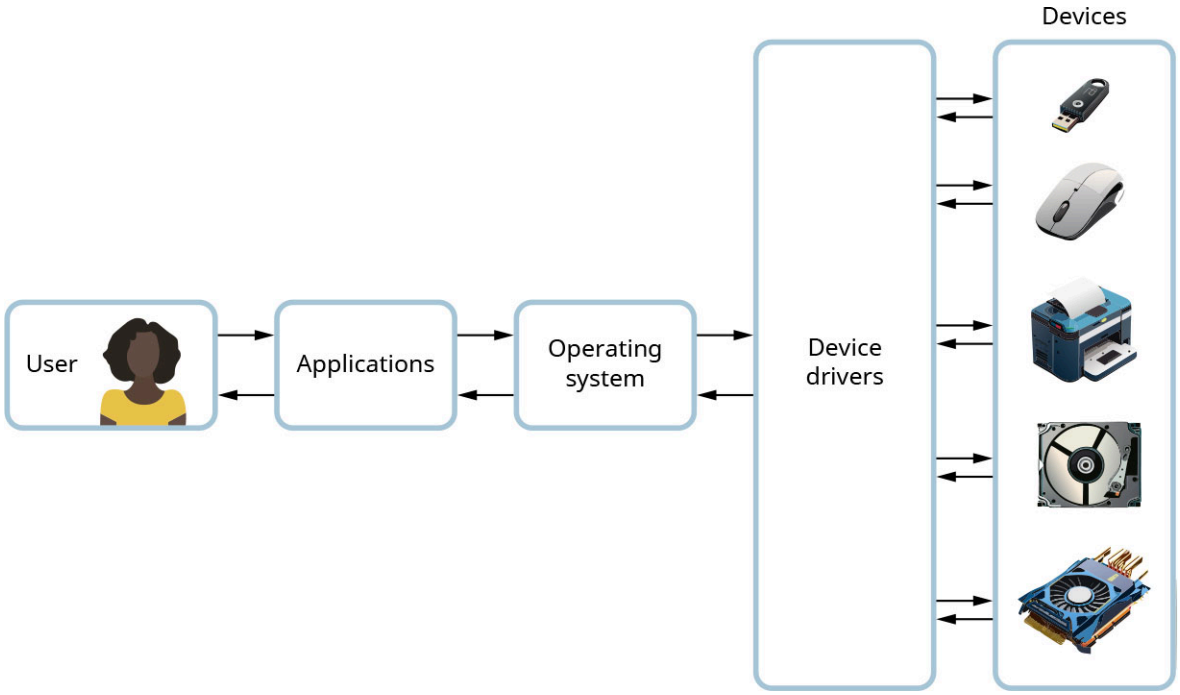


Figure 6.7 Multiple I/O devices are typically connected to a computer. When the user starts using an application, the operating system will define the devices used by the application by using each device's driver. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

LINK TO LEARNING

Are your Windows drivers up to date? Check out to see if you have the latest drivers (https://openstax.org/r/76latestdriver) for your Microsoft devices.

Device drivers are specialized software components that enable higher-level computer programs to interact with hardware devices. These drivers provide a software interface to hardware devices, allowing operating systems and other computer applications to access hardware functions without needing to know precise details about the hardware being used.

- At the heart of any computer is the CPU, the primary component responsible for executing instructions. Interestingly, CPUs themselves do not typically require external device drivers for direct operation, as the core management of CPU resources is a fundamental role of the operating system.
- Memory operates under the direct management of the operating system, which allocates and manages the system's memory resources. RAM can be accessed randomly and used for storing data temporarily while a computer is running. While standard RAM modules—both dynamic and static RAM—do not necessitate distinct drivers, specialized memory hardware, such as flash memory devices, including solid-state drives (SSDs), USB flash drives, and memory cards, interact with the system through file system drivers that manage the organization and access of stored data.
- Storage devices, encompassing a broad range of hardware from traditional hard disk drives (HDDs) to modern SSDs and removable storage media, require device drivers to facilitate data read/write operations.
- Network connectivity relies on an array of device drivers designed to manage the protocols and hardware functions of network interfaces.

Device Registers

A **device register** is the interface a device presents to a programmer where each I/O device appears in the physical address space of the machine as a memory address. The operating system reads and writes device registers to control the device. There are three types of device registers: status, command, and data. The status register provides information about the current state of the device (e.g., read), the command register issues a command to the device (e.g., writing a value to the register), and the data register is used to transfer data between the computer and the device. Device registers use bits to service three purposes: parameters provided by the CPU to the device, status bits provided by the device, and control bits set by the CPU to initiate operations. Figure 6.8 provides an example of the content of each of a device register's bits.

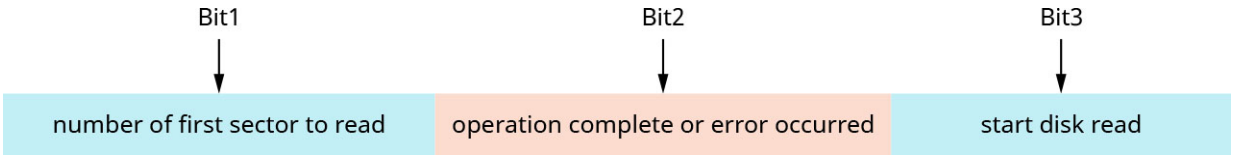


Figure 6.8 Device register bits define the first sector to read, the status of the operation, and the operation itself. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The behavior of device registers is different from that of ordinary memory locations. For example, the start operation bit may always read as 0, and the bits may change from one status to another (i.e., the operation complete bit may change without being written by the CPU).

The OS uses the device register to communicate with an I/O device as follows:

1. The CPU writes device registers to start the operation.
2. The CPU polls the ready bit in the device register.
3. The device sets the ready bit when the operation is finished.
4. The CPU loads the buffer address into a device register before starting the operation to define where to copy data read from disk.
5. Fast storage media devices move data directly to/from physical memory via direct memory access (DMA); other devices require the intervention of the CPU via programmed I/O or interrupt initiated I/O.
6. Interrupts allow the CPU to do other work while devices are operating, and the OS figures out which device interrupted.

CONCEPTS IN PRACTICE

Operating Systems, Printing, and Networking

Pretty much anyone needs to print a document these days. It is therefore important to understand how an operating system enables us to print documents so easily. When an application wants to print a document, it hands that task off to the operating system. The operating system sends instructions to the printer's drivers, which then send the correct signals to the printer.

Being able to access the network is another critical need in today's business world. It is therefore important to understand how OSs facilitate access to the network and how they control it. When a user uses an application that interacts with the Internet, the application sends messages from process to process using the OS's transport layer socket API. These messages are then split into packets within the operating system and eventually passed to a network interface card device, which transmits them to the Internet.

Figure 6.9 depicts the device register for a keyboard. Note that it has 2 bytes (16 bits). The data is in the first byte, and the second byte includes all zeros (i.e., from bit 8 to bit 15). When the user starts typing, the ready bit, which is bit number 15, sets to 1.



Figure 6.9 In this device register for a keyboard, the ready bit is set to 1 for when a user starts using the keyboard. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

OS Structure

In this section, we learn about the organization and structure of the OS. As you may recall, an OS consists of many core components but also includes other components such as privileged and non-privileged system programs. In an OS, it is important to have dual mode operations to ensure a high level of security and authority. The **dual mode** is responsible for separating the user mode from the kernel mode. A program that can run only in the kernel mode is known as a **privileged system program**. An example of a privileged system program is the bootstrap code, which is the first code that is executed when the OS starts. The OS depends on the bootstrap to be loaded and to work correctly. Figure 6.10 demonstrates the bootstrap use. If the user attempts to make any execution on privileged systems instructions, the execution will not be performed.

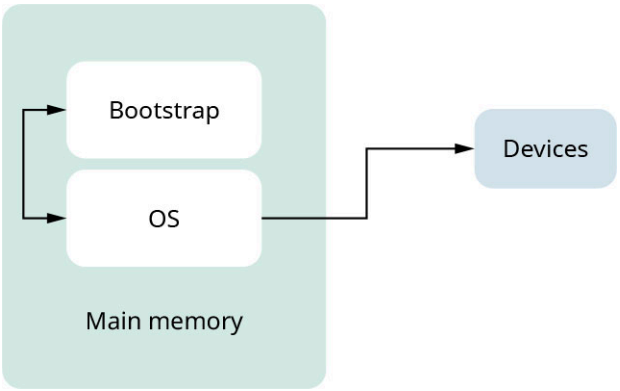


Figure 6.10 To start using any device, the OS will use the bootstrap program. (attribution: Copyright Rice University, OpenStax, under

CC BY NC-SA 4.0 license)

A program that can run only in the user mode is called a **non-privileged system program**. An example of a non-privileged system program is reading the processor status and the system time. In general, this is an instruction that any application or user can execute.

Designing a large, complex program is a major software engineering challenge because the program must not only perform well, but it must also be reliable, extensible, and backward compatible. OS design has been an evolutionary trial and error process. Successful OS designs have had a variety of architectures, such as monolithic, layered, cloud infrastructure and exokernels, microkernels, and virtual machine monitors. As the design of an OS—and even its role—are still evolving, it is simply impossible today to pick one correct way to structure an OS. The choice of OS architecture depends on various factors, including the specific requirements, trade-offs, and goals of the OS's intended use.

Monolithic OS Design

A **monolithic design** refers to a specific architecture for OSs where the entire OS operates within the kernel space, and all components and functionalities of the operating system are organized within the space. In a monolithic architecture, the entire operating system functions as a single, integrated unit, and all of its components, such as process management, file systems, device drivers, and memory management, reside and operate within the same address space, known as the kernel space. This means that the entire OS operates as a single, large program, and any module or component can directly call the functions of another without any restrictions.

While a monolithic design simplifies the communication and interaction between OS components, it has both advantages and disadvantages. One advantage is that it generally provides efficient and fast communication between different parts of the OS because they share the same address space. Another major advantage of the monolithic design is that it uses a familiar architectural style, and the cost of module interactions in terms of procedure calls is low. However, there are many disadvantages. Namely, this structure is hard to understand, modify, or maintain, and does not support fault containment. A failure in any one component can potentially crash the entire system, making it less fault-tolerant compared to more modular architectures. In this case, the alternative is to find an organizational way to simplify the OS design and implementation.

Traditional OSs such as UNIX (Figure 6.11) were built using the monolithic architecture. In contrast to monolithic designs, other OS architectures such as microkernel or hybrid designs distribute OS functionalities into separate modules or user-space processes, leading to better modularity and potentially improved system stability.

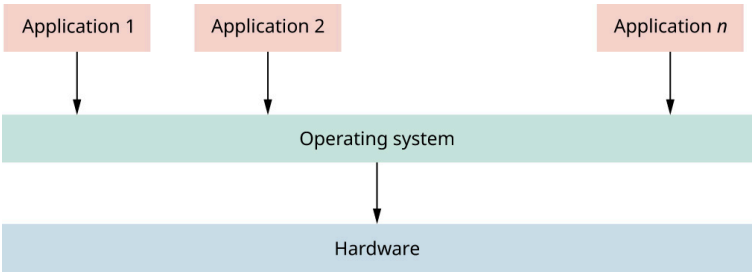


Figure 6.11 The entire OS works as one piece in the monolithic architecture. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Layered OS Design

One alternative way to achieve monolithic OS design is the **layered OS architecture**, which consists of implementing the OS as a set of layers where each layer exposes an enhanced "virtual machine" to the layer above, as illustrated in Figure 6.12.

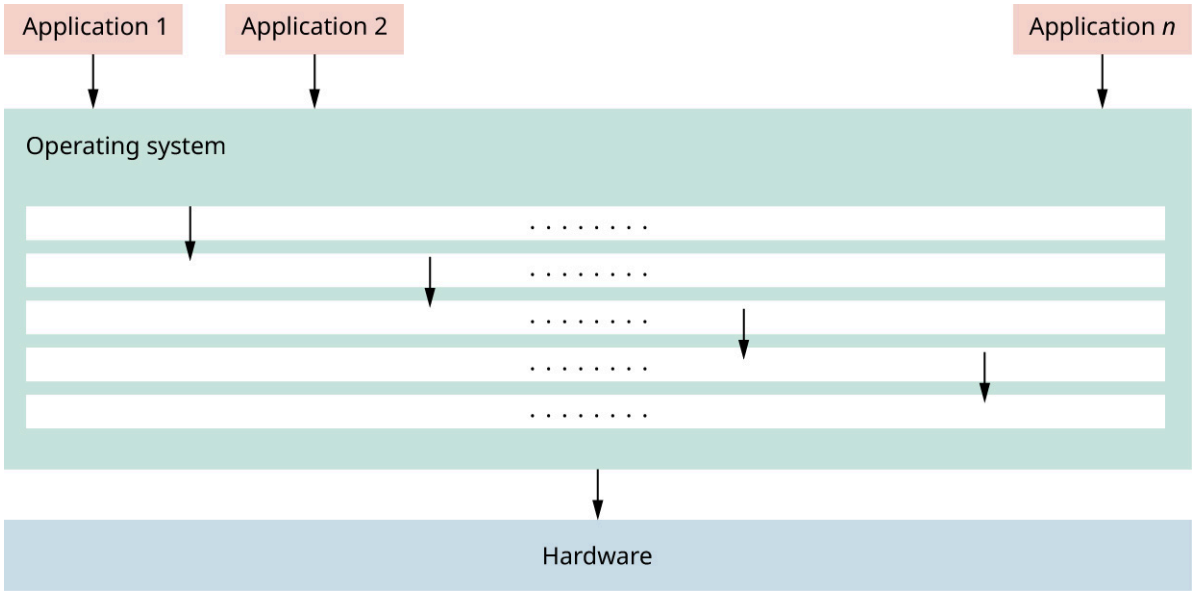


Figure 6.12 In a layered OS architecture, the OS is divided into layers, and each layer will be responsible for a specific task. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The first example of a layered approach was Dijkstra's THE system, which was designed in 1968. There were six layers, and they were organized as follows.

- Layer 5: A job manager executes users' programs.
- Layer 4: A **device manager** handles devices and provides buffering.
- Layer 3: A console manager implements virtual consoles.
- Layer 2: A **page manager** implements virtual memories for each process.
- Layer 1: A kernel implements a virtual processor for each process.
- Layer 0: Hardware is the physical components of the computer.

Each layer in this setup can be tested and verified independently. Layering also helped with implementation and supported attempts at formal verification of correctness (if you can call only layers below, it is not possible to run into a loop across the various layers).

There are, however, many disadvantages to using the layered OS architecture. Namely, it imposes a hierarchical structure, while real systems are more complex because a file system requires virtual memory (VM) services, and the VM likes to use files for its backing store. Layering also imposes a performance penalty as each layer crossing has overhead associated with static versus dynamic enforcement of invocation restrictions. There is also a disconnect between model and reality as systems modeled as layers may not really be built that way.

Hardware Abstraction

The **hardware abstraction layer (HAL)** is an example of layering in modern OSs, and it allows an OS to interact with a hardware device at a general or abstract level rather than going deep into a detailed hardware level, which improves readability and maintainability. In general, the goal of HAL is to separate hardware-specific routines from the core OS. HAL enables portability of core code across different pieces of hardware.

Microkernels

Another alternative OS design to monolithic OS design is a microkernels architecture. Within a **microkernel**, the functionality and capabilities are added to a minimal core OS as plug-ins. Microkernel architecture is also called a plug-in architecture because functionalities are added as plug-ins (Figure 6.13).

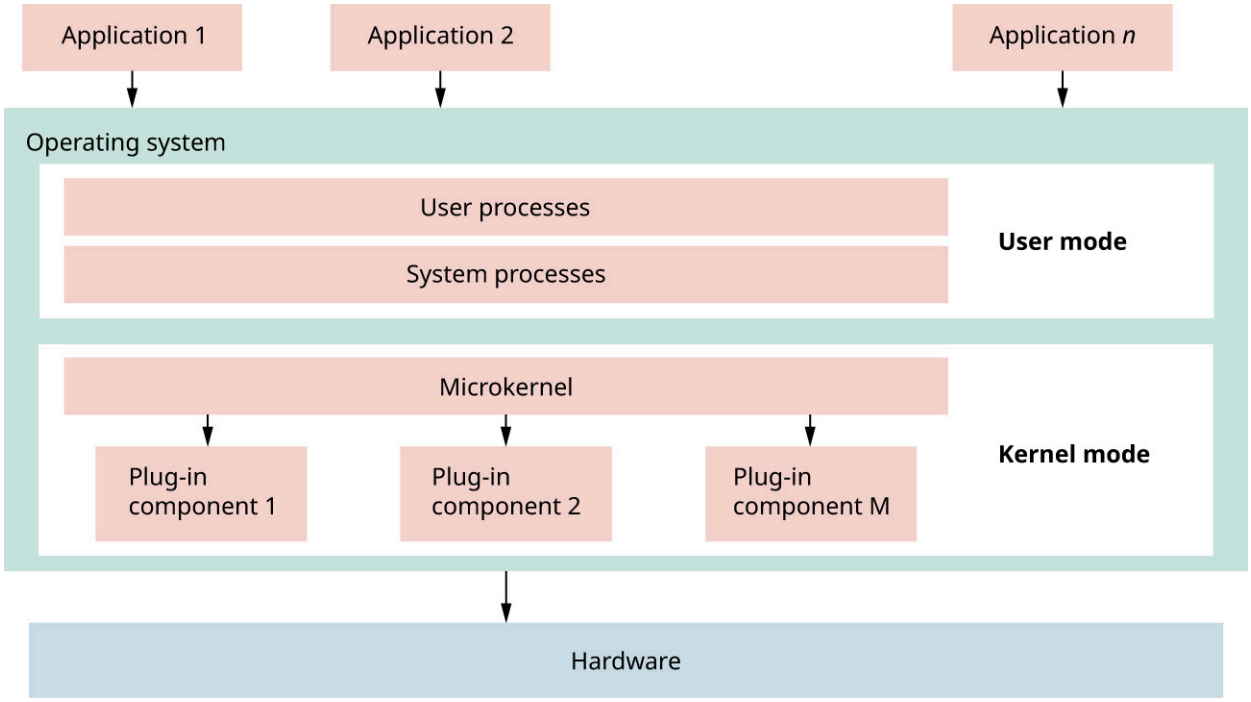


Figure 6.13 In the microkernel architecture, kernel mode is divided into multiple plug-ins to process the operations. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The goal of a microkernel architecture is to minimize what goes into the kernel and implement everything else that traditionally goes in an OS in terms of user-level processes. This results in improving reliability due to the isolation between components. Also, there is less code running at full privilege, and greater ease of extension and customization. However, performance is generally poor due to user and kernel boundary crossings, which represent a security risk when the kernel code operates on the data.

The first microkernel system was Hydra (CMU, 1970), followed by Mach (CMU), Chorus (French UNIX-like OS), and OS X (Apple). Windows OS used to use a microkernel, but now uses a hybrid kernel architecture that combines the benefits of monolithic, microkernel, and plug-in OS architectures (Figure 6.14).

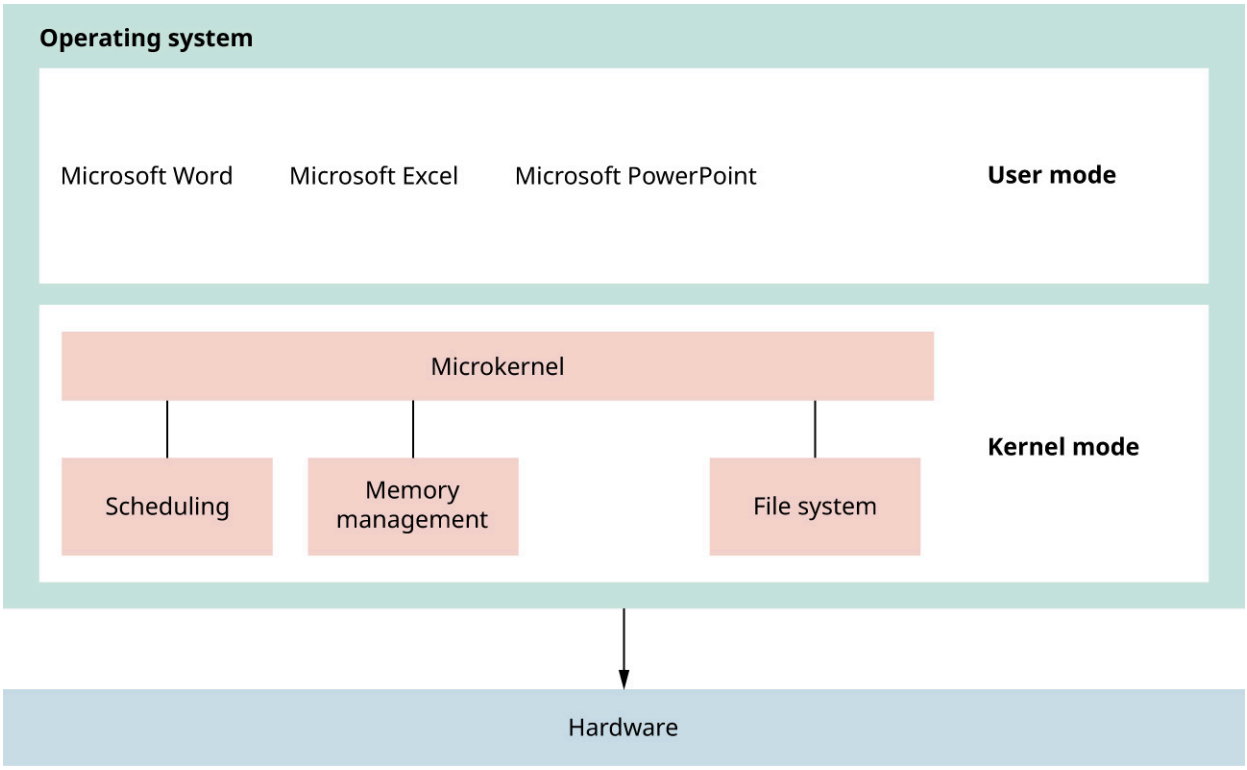


Figure 6.14 Windows OS as plug-in architecture—in Windows OS, all of the applications are in the user mode, and the OS operations are divided into plug-ins in kernel mode. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

LINK TO LEARNING

Read this article about how Apple uses kernel architecture (https://openstax.org/r/76kernelarch) in its macOS.

Cloud Infrastructure and Exokernel

Two concepts that each represent critical advancements in the field of computing by offering different methodologies for resource abstraction, allocation, and management are cloud infrastructure and the exokernel.

The **exokernel** architecture simplifies the OS by making its core (kernel) really small and letting apps have more control over the computer's hardware. Unlike usual systems that hide hardware details, an exokernel shows these details to apps, letting them use the hardware more smartly. This way, it cuts down on unnecessary steps and lets each app manage resources its own way, which can make the computer run better and faster. This design is especially good for special computing tasks where apps can really benefit from managing hardware directly, and this can lead to better performance and more efficient use of the computer's parts. There are several limitations associated with exokernels. They lack abstractions for operating systems services, which makes it difficult to achieve consistency across applications. They also require an application developer to manage resource allocation and protection and to implement security mechanisms, which raises concerns.

The virtualized and scalable hardware resources that are delivered over the Internet as a service are called **cloud infrastructure**. This infrastructure encompasses a range of components including servers, storage devices, network equipment, and virtualization software, all hosted within data centers managed by cloud service providers. Unlike traditional physical infrastructure, cloud infrastructure offers flexibility, scalability, and accessibility, allowing users to access and manage computing resources remotely through the Internet. Cloud

infrastructure is categorized into three service models:

- Infrastructure as a Service (IaaS) provides virtualized computing resources over the Internet, such as AWS EC2, Google Compute Engine, and Microsoft Azure VMs.
- Platform as a Service (PaaS) offers hardware and software tools over the Internet, typically targeting developers such as Google App Engine and Microsoft Azure.
- Software as a Service (SaaS) delivers software applications over the Internet, accessible from a web browser without installation or running on the user's personal devices such as Google Workspace and Microsoft Office 365.

The main disadvantages of cloud infrastructure are potential downtime, security and privacy concerns, vulnerability to attacks, limited control and flexibility, vendor lock-in, cost concerns, latency issues, Internet dependency, technical issues, lack of support, bandwidth issues, and varied performance.

TECHNOLOGY IN EVERYDAY LIFE

Remote Emailing

Virtual machines have an impact on every industry today as they facilitate the running of numerous business applications. For example, most companies use Google Mail or Microsoft Outlook to supply their employees with email services. These are Software as a Service (SaaS) cloud infrastructures that operate the email software by making use of virtual machines that run in Google and Microsoft Cloud data centers.

A major advantage of SaaS email might be portability. Because the application can run on any local machine with an appropriate browser, this enables users to use the app on many different platforms, including mobile phones and tablets. In addition, there is an automatic backup of saved data, and no need for complicated installations and configuration. The disadvantages of SaaS might include the lack of security of the data stored in the app with respect to unauthorized access and the inability of users to use the application when the network connection is weak or nonexistent.

6.3 Processes and Concurrency

Learning Objectives

By the end of this section, you will be able to:

- Explain what processes are and how they interact with the operating system
- Discuss how operating systems support concurrency

As we mentioned before, the OS divides the tasks it needs to perform into processes. It would be a waste of time for every process to wait until the current process completes. Instead, the OS performs more than one task at the same time, or concurrently. The computing model that improves performance when multiple processors are executing instructions simultaneously is **concurrent processing**. In this module, we learn about processes and concurrency by digging down into process management and inter-process communication (IPC), threads, scheduling and dispatching, and synchronization.

Process

To review, a process is a fundamental concept in an OS that represents an instance of a program in execution. It's an abstraction used by the OS to provide the environment a program needs to run. When we talk about a program, we are typically referring to a set of instructions stored on disk; these instructions are passive and don't do anything by themselves. However, when the program is loaded into the memory of a computer and begins execution, it becomes an active entity known as a process. This transformation is crucial for any computational task, as it moves the program from a static state into an active one where it can perform

actions, manipulate data, and interact with other processes.

CONCEPTS IN PRACTICE

Under the Cover

Most of the applications we use today on our smartphones or laptops use IPC and a client-server architecture. It is therefore important to understand what is under the cover in case these applications suddenly stop working.

In an OS, client-server communication refers to the exchange of data and services among multiple machines or processes. One process or machine acts as a client requesting a service or data, and another machine or process acts like a server providing those services or data to the client machine. The communication between server and client uses various OS protocols and mechanisms for message passing, including sockets, remote procedure calls, and inter-process communication.

Process Management

A process consists of at least an address space, a CPU state, and a set of OS resources. A process's address space is illustrated in Figure 6.15. As you learned earlier in this chapter, the address space contains the instruction code for the corresponding running program and the data needed by the running program. The data can be **static data**, which does not change within the program, or **heap data**, which serves a collection of elements and uses a tree or stack data structure. A **CPU state** consists of the program counter, the stack pointer (SP), and general-purpose registers. The PC is a CPU register located in the processor that has the next instruction address. The **stack pointer (SP)** is a register that indicates the location of the last item that was added to the stack. A **general-purpose register (GPR)** is an extra register that is used for storing operands and pointers; GPRs are where the instructions can read and write the value of their parameters mostly when the program is interrupted. There are many OS resources such as the CPU, network connections, file storage, I/O, and sound channels. An address space, CPU state, and OS resources are everything you need to run the program or to resume it if it is interrupted at some point.

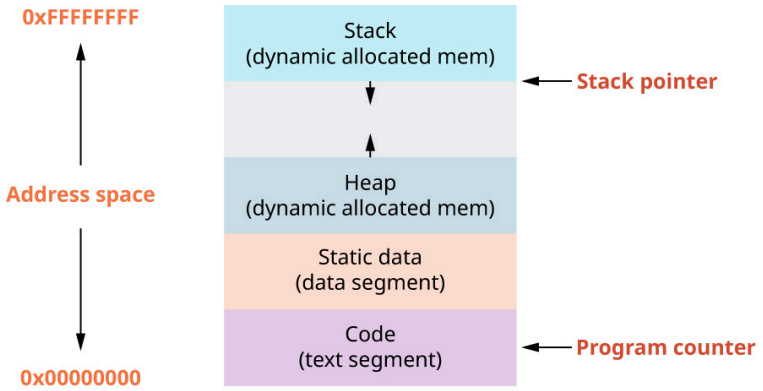


Figure 6.15 A process's address space includes the stack pointer (SP) and the program counter (PC). It has static data in the data segment and code in the text segment. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The OS's process namespace particulars depend on the specific OS, but in general, the name of a process is called a **process ID (PID)**, and it has an integer type. A PID namespace is a set of unique numbers that identify processes. The PID namespace is global to the system, and only one process at a time has a particular PID. It is possible to create a container to isolate a process, along with only the files and configurations necessary to run and operate it. This type of isolated process environment allows for greater security, consistency, and portability across different systems. PID namespaces allow containers to provide functionality such as suspending the set of processes and resuming different set of processes in the memory. The OS maintains a

data structure called the **process control block (PCB)** or process descriptor to keep track of a process state and to store all of the information identified by the PID about a process. As illustrated in Figure 6.16, the PCB contains information that serves as metadata for the process such as PID, process state, parent process ID (PPID), execution state, PC, SP, registers, address space info, and user id (uid).

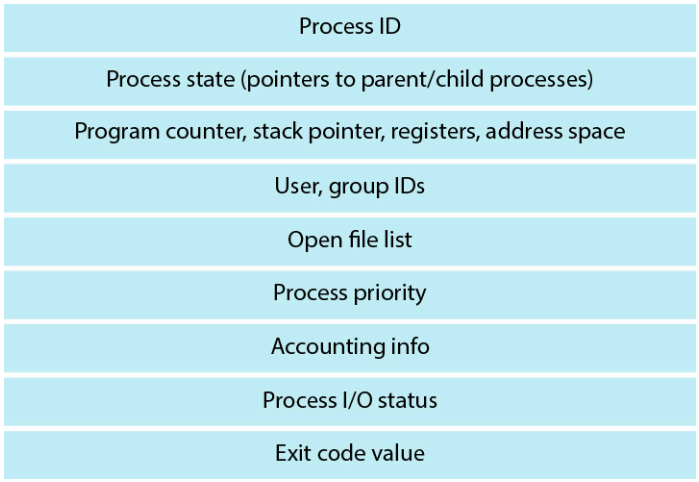


Figure 6.16 PCB are data structures that an OS uses to store detailed attributes for each process it is tracking. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Suppose that there's a process that needs input from a user. Because the OS does not use any CPU while waiting for this input, it marks the process in the PCB as suspended. When the user enters the input, the process's status in the PCB changes. The OS keeps the details relating to all of a process's execution state in the PCB when the process is not running. The CPU state (e.g., PC, SP, GPRs) is transferred out of the hardware registers into the PCB when execution is suspended. When a process is running, its state is spread between the PCB and the CPU.

Here is an example in Linux that illustrates how to use the PID when creating a child process from a parent main process, which uses the fork command in C:

```
#include "unistd.h"
#include <stdio.h>

int main() {
  // Forking to create a new process
  pid_t pid = fork();
  if (pid == 0) {
    // Child process
    printf("This is the child process with PID %d\n", getpid());
  } else if (pid > 0) {
    // Parent process
    printf("This is the parent process with PID %d\n", getpid());
  } else {
    // Fork failed
    printf("Fork failed!\n");
    return 1;
  }
  return 0;
}
```

Inter-Process Communication

Processes provide isolation to guarantee a high level of protection, but sometimes these processes need to communicate and collaborate. This is made possible via **inter-process communication (IPC)**, which is a mechanism that enables different processes running on an operating system to exchange data among themselves and thus allows these processes to communicate and collaborate. As Figure 6.17 shows, IPC allows one process, P_1, to provide input to another process, P_2, while yet another process, P_3, is also running.

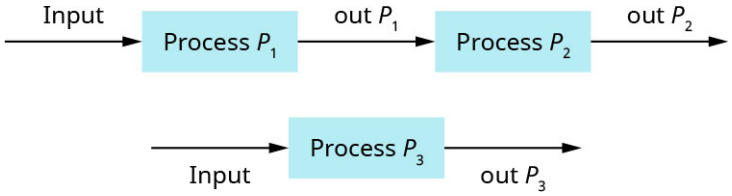


Figure 6.17 In this example of inter-process communication (IPC), the process P_1 has an output called "out P_1" that will be the input that process P_2 needs to start working. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

LINK TO LEARNING

Read this article on inter-process communication (https://openstax.org/r/76interproc) to learn more about how the processes running in a computer system can be independent or noncooperating.

Streams, Pipes, and Sockets

The IPC has a range of mechanisms that enable processes to communicate with each other such as pipes, shared memory, and sockets. A **pipe** (sometimes called a named pipe) is a data communication method between two processes that uses a specific name and standard I/O operations, and thus allows for data transfer within a file system. Shared memory allows the processes to communicate with each other without a middleman. A socket is an end point for sending and receiving data between different machines in the same network using Internet protocols.

What this means is that, for example in Figure 6.17, where process P_1 provides input to process P_2, there are many ways for the IPC to deliver this input. It could send command-line arguments that are available only to the parent process (i.e., input data is passed to a program when the program is invoked from a shell); or it could communicate via files (e.g., one process writes, the other process reads). Alternatively, the IPC could optimize file communication via the use of pipes with memory buffers (effective when processes are related), or it could utilize environment variables (i.e., variables defined within a shell that can hold a dynamically allocated value).

Concurrency

Multiple activities and processes happening at the same time—in other words, the OS handling multiple tasks at once—is called **concurrency**. Concurrent processing can be achieved via a multiprogramming environment, a multiprocessing environment, or a distributed processing environment (Figure 6.18). In a multiprogramming environment, multiple tasks are shared by one processor. In a multiprocessing environment, two or more processors that have a shared memory are used. In a distributed processing environment, two or more computers are connected by a communication network, and there is no shared memory between the processors (i.e., each computer has its own memory). Multiprocessing is used to accelerate processing by running tasks in parallel, while distributed computing environments are typically used to implement client-server or peer-to-peer architectures. As noted earlier and as will be investigated further in this section, threads are another means of achieving concurrency. However, true parallelism can only be achieved by using multiple processors to execute multiple threads simultaneously.

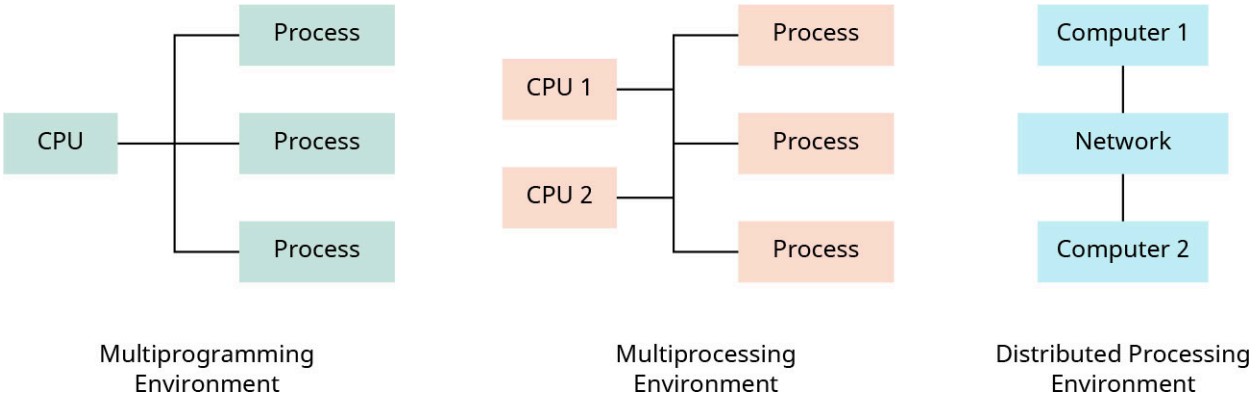


Figure 6.18 Concurrent processing, in which the OS handles multiple tasks at once, can be achieved through three different types of environments. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

LINK TO LEARNING

Read this article on concurrency in operating systems (https://openstax.org/r/76concurr) to learn more about the principles and problems associated with the concept of concurrency.

Threads

As discussed earlier in this chapter, processes represent running programs, and threads enable programs to perform multiple things at once. A process is an instance of a program that is being executed by an OS. Each process has its own memory space and resources. The OS creates a new process for every executed program and allocates the required resources for the process. For example, it allocates a specific size of memory and CPU time. The process may have one or more threads, each thread has its own context, but all of the threads within a process share the same resources.

Threads are the OS's unit of concurrency and the abstraction of a virtual CPU core. Each thread is a basic unit of execution that executes a set of instructions in a specific order. A thread is a lightweight process that shares OS resources (e.g., memory and I/O) with other threads within the same process. Threads are created by the OS kernel, and each thread has its own register stack. All the threads in a given process are sharing the same memory space. Threads are essentially paths of execution that operate within the confines of a process.

For example, consider today's web browsers. Each open tab in a web browser is its own process with its own address space, but within a tab, there might be multiple things going on. A user can scroll around and interact with a web page while a video is playing in the background. In this case, one thread could be used to manage the user interactions, while another thread is used to manage video playback on the web page.

In the early versions of the operating system used on IBM and DEC mainframe computers, concurrency was achieved via time sharing. In other words, a single task was performed using a single process with a single thread. This kind of process allowed only one user at a time to process or run a job. This old way of processing required more resources such as memory and processors to finish a single task. By the late 1970s, the more prominent approach became **multitasking**, which makes it possible for the OS to run multiple processes at the same time using time slicing. A **time slice** is a short time frame that gets assigned to a process for CPU execution. It corresponds to the time frame during which a process is allotted to run in preemptive multitasking CPUs. In that case, a scheduler runs each process every single time slice. The preemptive multitasking approach was not sufficient to improve the OS performance, so twenty years later, OSs still support multitasking using multiple threads. Figure 6.19 illustrates single and multithreaded processes.

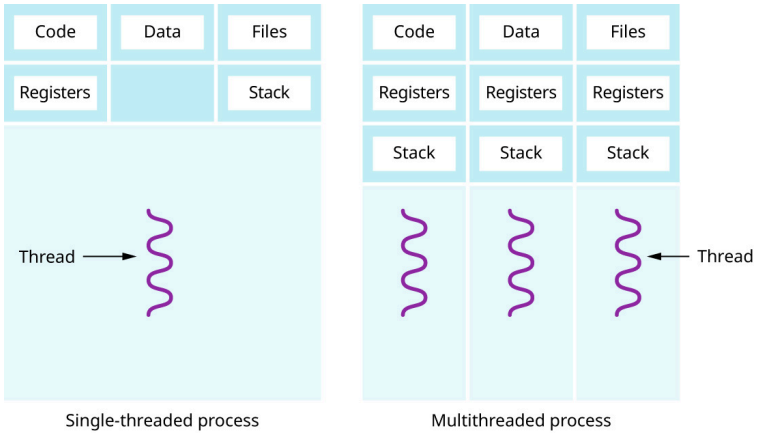


Figure 6.19 In the single-threaded process, a single thread will use its own code, data, and files along with its own registers and stack. In the multithreaded process, a multithread will have a set of registers and stacks. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

When a program is run, it operates as a process in the OS. This process can execute multiple threads simultaneously, allowing for parallel execution of tasks within the same application environment. The threads are managed and run as follows:

- Thread creation: Threads are created by the process using specific system calls to the operating system, such as pthread_create in UNIX/Linux or CreateThread in Windows. When created, each thread starts its execution at a designated start point in the program code. This is often a function passed to the thread creation call.
- Execution and scheduling: Once created, threads are scheduled by the operating system's scheduler, which allocates CPU time to them. The scheduling can be preemptive, where the OS decides when to switch between threads, or cooperative, where threads voluntarily yield control to allow other threads to run.
- Sharing and isolation: Threads share the same process resources, such as memory and file handles, making inter-process communication and data sharing more efficient than between separate processes. However, they run in their own thread of execution, meaning each has its own stack, program counter, and set of registers to keep track of its execution state.
- Synchronization: To safely manage the access to shared resources and ensure data consistency, threads often use synchronization mechanisms like mutexes, semaphores, and condition variables. These tools help prevent race conditions, where the outcome of operations depends on the sequence or timing of other uncontrollable events.
- Completion: A thread completes its execution when it exits its start function, either by returning normally or by being explicitly terminated by itself or another thread. Upon completion, any resources specifically allocated to the thread are cleaned up by the operating system.

Scheduling/Dispatching

Scheduling tasks and determining which resources should be used when are central responsibilities of the OS—they are also the means by which the OS achieves concurrent processing.

OSs may switch the CPU from process to process hundreds of thousands of times per second. On today's hardware, this takes a few microseconds. Choosing which process to run next is called scheduling. The activity of handling the removal of the running process from the CPU and the selection of another process based on a particular strategy is called scheduling. In OSs, a process can be in one of several states. These states are part of the process life cycle, and understanding them is essential for grasping how the OS manages processes. The specific names and number of states can vary between OSs, but the fundamental concepts remain the same—or at least quite similar. Each process has an execution state, which indicates what it is currently doing and can be as follows:

- ready: waiting to be assigned a core

- running: executing on a core
- blocked (also known as “waiting”): waiting for an event, so not eligible to be given a core

Figure 6.20 represents a process’s life cycle within an OS. It starts with the creation of a process, which brings a new process into existence and places it in the ready state. In this state, the process is loaded into memory and is prepared to run, but is waiting for the CPU to become available. When the scheduler dispatches the process, it transitions to the running state, where it is actively executing its instructions. If the process is interrupted, it reverts to the ready state, waiting once again for a chance to run. Certain events, such as I/O requests or page faults, can cause the running process to experience a trap or exception. When this happens, the process enters the blocked state because it can’t proceed until the event it’s waiting for, such as the completion of an I/O operation, occurs. Once the awaited event is finished, the process can leave the blocked state and reenter the ready state, once again waiting for CPU time.

When a process completes its execution or is terminated, it reaches the terminate state. However, termination doesn’t necessarily mean the process is immediately removed from the system; it may enter a zombie state. In this state, the process has finished its job but still occupies an entry in the process table, effectively being in a state of limbo until its parent process acknowledges its completion. This acknowledgment allows the OS to fully clean up any remaining information, officially ending the process’s life cycle.

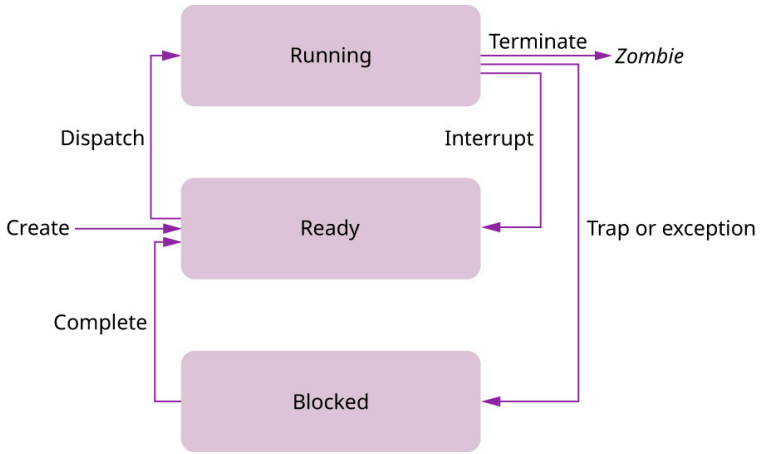


Figure 6.20 The life cycle of a process within an OS starts with the creation of the process and ends with an acknowledgment of completion. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

As noted earlier, each process is represented by a unique identifier called a process identifier (PID). An OS encapsulates the process in a data structure type called a process control block (PCB) that contains information about the process, such as current status, which could be running status, ready status, or blocked status. A PCB defines the register status, the process ID, the execution time, the memory space, as well as other information. The OS kernel scheduling function is responsible for maintaining the content of PCB and scheduling processes for the CPU to execute based on assigned priorities.

Another policy decision the OS makes is to decide whether to give out non-CPU resources such as memory and I/O. As they are data structures, PCBs are located in OS memory. When a process is created, the OS allocates a PCB for it. After initializing the PCB, the OS then does other things not related to the PCB such as allocating the PCB to the correct queue. As a process executes, the OS moves the process’s PCB from queue to queue. When a process is terminated, the PCB may be retained for a while. Eventually, the OS deallocates the PCB.

The act of switching the CPU from one process to another is called a **context switch**. Context switching is a procedure that a computer’s CPU follows to change from one task to another while ensuring that the tasks do not conflict. In Figure 6.21, the process P_0 is in the running state, and the process P_1 is in the ready state. When an interruption occurs, the process P_0 must be switched from the running to the ready state, and the process P_1 must be switched from the ready to the running state. To accomplish this, the OS performs these steps: it saves the context of the process P_0 in PCB_0, switches P_0 from the ready state, selects P_1 to be

executed, and, finally, updates PCB_1 of process P_1.

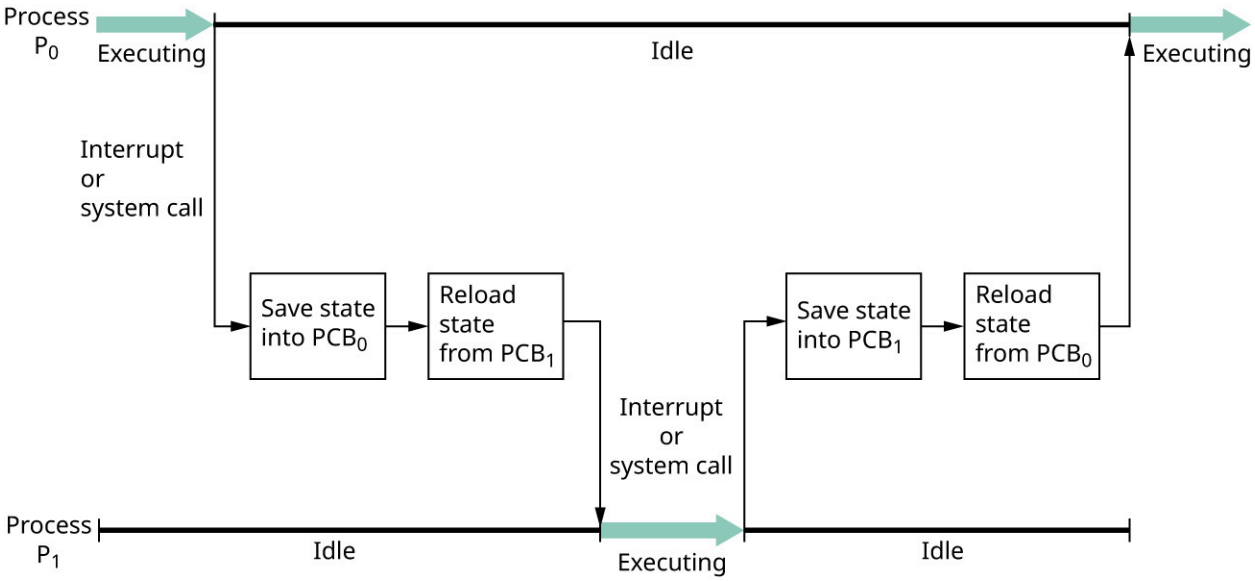


Figure 6.21 This example of context switching shows the steps involved when a CPU switches from executing process P_0 to running process P_1 and then back to P_0. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Transactions and Scheduling

As you've learned, scheduling is the act of determining which process is in the ready state and should be moved to the running state when more resources are requested than can be granted immediately and in which order such requests should be serviced. Examples are processor scheduling (i.e., one processor, many threads) and memory scheduling in virtual memory systems.

A good scheduling algorithm minimizes response time, efficiently utilizes resources (that is, it ensures full utilization by keeping cores and disks busy, with minimal context switches), and implements fairness by distributing CPU cycles equitably. Ideally, scheduling algorithms should not affect the results produced by the system. Optimal scheduling schemes would require the ability to predict the future, making adaptive algorithms the preferred choice.

Examples of simple scheduling algorithms include:

- **first come, first served (FCFS)** scheduling, also called first in, first out (FIFO): In this algorithm, the first job that comes to the processor is executed first. For example, suppose we have two processes—process P_1 with execution time 3 and process P_2 with execution time 2. The arrival time for both process P_1 and process P_2 is t_0. The system will start with process P_1 at t_0 and finish at t_3 and process P_2 will start at time t_3 to time t_5, as shown in Figure 6.22. The wait time for process P_1 is 0 and the wait time for process P_2 is 3.

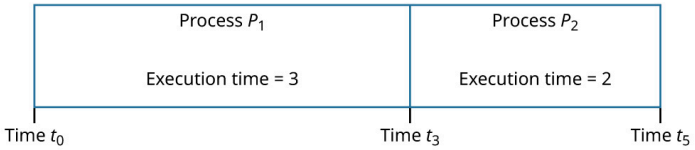


Figure 6.22 Two processes, P_1 and P_2, are being processed using the FCFS algorithm. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

- **round-robin scheduling (RR)**: This algorithm, which is widely used in time-sharing systems, is designed to ensure fairness among processes by giving each process an equal share of the CPU. Its operation is relatively straightforward but effective in environments where a large number of processes need to be handled efficiently. The core idea of RR scheduling is to assign a fixed time slice, often referred to as a

quantum, to each process in the ready queue. The CPU scheduler cycles through the queue, allocating the CPU to each process for a duration equal to 1 quantum. For the previous example, let us set the quantum to 2. Then, the processor will execute part of process P_1 and move to process P_2 then go back to process P_1 (Figure 6.23).

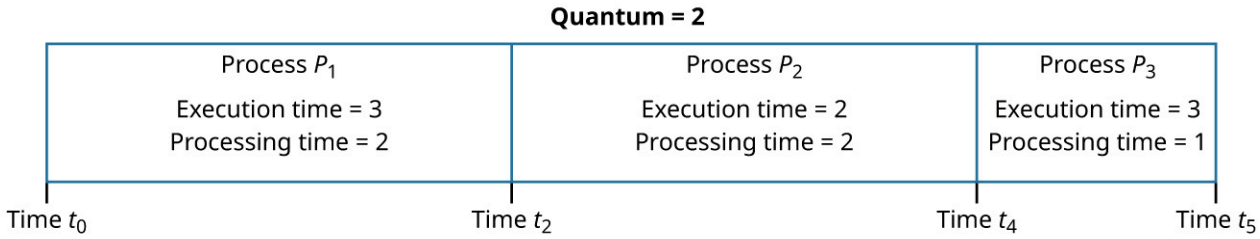


Figure 6.23 Two processes, P_1 and P_2, are being processed using the RR algorithm. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

- **shortest time to completion first (STCF)**, also called shortest job first (SJF): This algorithm takes the best approach to minimize the waiting time, but it requires that the processor knows the processing time in advance. In the previous example, the processor will run process P_2 before process P_1 because the execution time is less.
- **shortest remaining processing time (SRPT)**: In SRPT, the processor is reallocated to a newer ready job with a shorter remaining completion time whenever such a job arrives.

Synchronization

Synchronization in concurrent programming is crucial for ensuring that multiple threads or processes can work together without interfering with each other's operations on shared resources. Through mechanisms like locks, condition variables, and semaphores, developers can design systems that are both efficient and safe, avoiding issues such as data races and deadlocks. One way of coordinating multiple concurrent activities that are using shared state is **synchronization**, which groups operations together automatically to ensure cooperation between threads. To ensure that only one thread does a particular task at a time, we can use a program called **mutual exclusion** that prevents simultaneous access to a shared resource. A **critical section** is a piece of code that only one thread can execute at once. Also, only one thread at a time will get into this section of code. A critical section is the result of mutual exclusion. Critical sections and mutual exclusion are in fact two ways of describing the same thing. Critical sections are sequences of instructions that may get incorrect results if executed simultaneously. Mutual exclusion is required to ensure that a process cannot enter its critical section while another concurrent process is currently present in its critical section. Figure 6.24 illustrates a representation of two processes, process P_1 and process P_2. At Time 1, process P_1 entered the critical section by printing on a shared printer. Process P_1 will finish printing at Time 2. While process P_1 is printing, process P_2 is attempting to print, but the OS will block process P_2 from printing until process P_1 reaches Time 2.

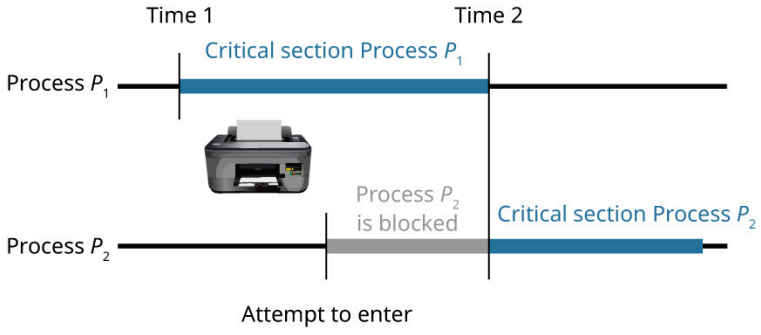


Figure 6.24 Process P_1 is using the printer starting from Time 1 and will finish printing at Time 2. Process P_2 is trying to print while process P_1 is printing. The OS will use mutual exclusion to prevent process P_2 from starting until process P_1 has ended. (attribution:

Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

A **lock** is a synchronization mechanism that is used to enforce mutual exclusion. A thread must acquire a lock before entering a critical section; if the lock is already held by another thread, the attempting thread will block until the lock becomes available. Another synchronization mechanism called a **condition variable** is used in conjunction with locks to allow threads to wait for certain conditions to become true. While a lock facilitates exclusive access to resources, a condition variable helps threads wait for specific states of the world.

A classic example of a synchronization challenge is the producers/consumers problem, where producers generate data and place it into a buffer, and consumers remove data from the buffer for processing. In a producer/consumer scenario, a consumer might wait on a condition variable if the buffer is empty, and a producer might signal this variable once it adds an item to the buffer. The key concerns associated with this scenario include ensuring that producers don't add data to a full buffer and that consumers don't try to remove data from an empty buffer. Synchronization tools like locks and condition variables are used to solve these issues. A **deadlock** occurs when a set of threads are blocked forever, waiting for each other to release resources. This can happen, for example, if thread A holds lock 1 and waits to acquire lock 2, while thread B holds lock 2 and waits to acquire lock 1. Avoiding deadlocks requires careful design, such as acquiring locks in a consistent order or using timeout mechanisms.

Some other tools used in synchronization are:

- Semaphores: Similar to locks, but allow multiple threads to access a finite number of resources
- Barriers: Enable multiple threads to wait until all have reached a certain point in their execution before any are allowed to proceed
- Read-Write Locks: Allow multiple readers to access a resource concurrently but require exclusive access for writers
- Mailboxes: Dedicated channels that connect two processes directly, allowing data to be exchanged between them; mailboxes behave like queues but use semaphores for controlled automatic access, and operate in first in, first out (FIFO) order only.

TECHNOLOGY IN EVERYDAY LIFE

Multitasking

The ability to run multiple programs at once on computers today has a huge productivity impact in all industries. In concurrent programming, managing shared resources among multiple threads or processes is crucial for maintaining data integrity and preventing race conditions. One common scenario is the producer-consumer problem, where one or more threads (producers) generate data, and others (consumers) consume it. To avoid conflicts, synchronization mechanisms like locks and condition variables are employed. Locks help ensure that only one thread accesses a critical section of code at a time. In the context of the producer-consumer problem, locks can be utilized to safeguard shared data structures, preventing simultaneous access by multiple threads and ensuring data consistency.

Condition variables are another synchronization tool that allows threads to coordinate their activities. In the context of producers and consumers, a condition variable could signal when data is available for consumption or when space is available for production. Threads can use these signals to efficiently wait for or notify others about the state of shared resources. The combination of locks and condition variables provides a powerful means to synchronize complex interactions between producers and consumers, ensuring orderly access to shared resources.

Despite the benefits of synchronization mechanisms, the improper use of locks can lead to issues like deadlocks, where two or more threads or processes are stuck in a circular wait, unable to proceed because each is waiting for the other to release a resource. This situation can bring a system to a standstill, and

careful design and coding practices are necessary to prevent or detect and recover from deadlocks. To handle deadlocks, techniques such as deadlock detection algorithms and prevention strategies are employed, contributing to the robustness of concurrent systems.

Allocation

The method that defines how data is stored in the memory by providing a set of requests for resources and identifying which processes should be given which resources to make the most efficient use of the resources is called **allocation**. Like scheduling, allocation is another kind of decision-making that an OS performs about how to use resources to support concurrency. There are three main forms of allocation: contiguous allocation, linked allocation, and indexed allocation.

In contiguous allocation, each file is assigned to a contiguous (i.e., neighboring) set of blocks in the memory. For example, if a file requires three blocks and is given a starting location x, the file will be allocated in x, $x + 1$, $x + 2$. The directory entry with contiguous allocation contains the starting block address and the length of the file. In linked allocation, each file is a linked list of memory blocks. Using the same example, the first block will be allocated in location x and it will include the address of the second block. The directory entry with linked allocation contains a pointer to the starting block and a pointer to the last block. In indexed allocation, each file has an index block containing the pointers to all blocks for that file.

6.4 Memory Management

Learning Objectives

By the end of this section, you will be able to:

- Discuss key concepts related to memory
- Evaluate dynamic storage management solutions
- Discuss the differences between virtual and physical memory

As you have learned, memory plays a huge role in OSs. Here, we discuss the memory multiplexing, linkers and dynamic linking, dynamic storage management, virtual memory, and demand paging.

Memory

Different processes and threads share the same hardware. It is therefore necessary to multiplex the CPU (i.e., temporal execution), memory (spatial access), and disks and devices. As discussed earlier, the complete working state of processes and/or kernels is defined by its data (i.e., memory, registers, and disk). For the sake of safety, security, and reliability, processes should be barred from having access to each other's memory. Dividing the capacity of the communication channel into multiple logical channels is considered **memory multiplexing**. There are several concepts that are critical to memory multiplexing, namely, isolation, sharing, virtualization, and utilization.

As you learned earlier in this chapter, isolation is important because it ensures that the multiple programs that are running concurrently on the same CPU and memory operate independently without interfering with each other's execution or data. In memory multiplexing, isolation is achieved through a set of technologies that prevent distinct process states from colliding in physical memory due to unintended overlap (i.e., overlap control). These technologies aim to, for example, prevent process P_1 from spying on process P_2. Or, if process P_1 has a bug, they ensure that this bug does not impact process P_2. There are many isolation mechanisms, including:

- User/kernel mode flag is a register that represents the CPU mode as user mode or kernel mode. As we have learned, the CPU boots in kernel mode, then it marks the flag as kernel mode. When the user starts any application, the CPU marks the flag as user mode.

- Address space boundaries protect the kernel and address space programs from each other.
- System call interface is the programming interface for application users to process a system call. As shown in Figure 6.25, a system call is executed by the user mode to request the kernel mode to perform a specific action (e.g., `syscall ()` function).

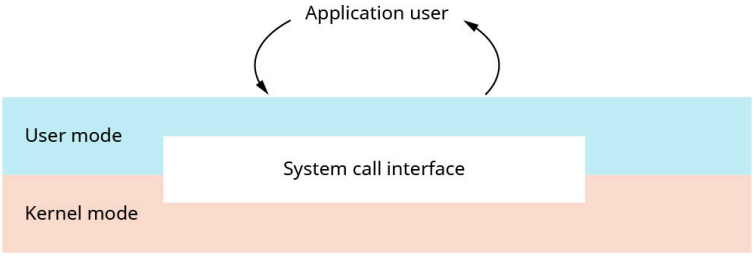


Figure 6.25 The system call interface uses an isolation mechanism to address a system call, which is a request that arises from the user mode and requires the kernel mode to perform an action. The system call interface prevents these processes from overlapping or colliding. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Time slicing provides a time frame for each process to run in a preemptive multitasking CPU such that each process runs every single time slice. If the process finishes the job before the time slice, it releases the CPU and does not need to be swapped out. If the time slice ends and the process did not finish the job, the CPU shifts it out to the end of the processes queue. For example, assume we have three processes P_1 with execution time 3 ms, P_2 with execution time 4 ms, and P_3 with execution time 2 ms, and a time slice of 2 ms. Figure 6.26 illustrates how the CPU manages the processing using time slice and indicates in which time slice each process completes execution.

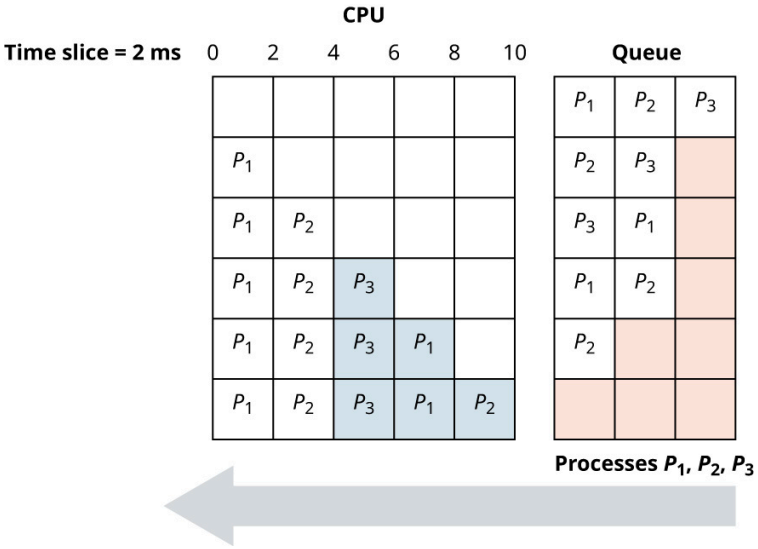


Figure 6.26 The three processes P_1, P_2, and P_3 (blue) will work as scheduled based on the time slice. The processes are scheduled in the queue and executed, so they come out of the queue (arrow). Empty spaces in the queue are shown in pink. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

When multiple processes can use the same piece of data concurrently, it is called **sharing**. The option for overlapping processes should be available when desired for efficiency and communication reasons. Memory sharing improves the performance of the system because the data is not copied from one address space to another, so memory allocation is done only once.

With respect to memory, virtualization is a technique that gives an application the impression that it has its own logical memory and that it is independent from the available physical memory. Thus, in virtualization, there is a need to create the illusion that there are more resources than those that actually exist in the underlying physical system. There are two approaches of memory virtualization: full virtualization and guest modification. When multiple operating systems run concurrently on a single physical machine, fully isolated from each other, by emulating hardware resources through a hypervisor, it is called **full virtualization**. For full

virtualizations, all OSs expect contiguous physical memory that starts from physical address 0. In the context of virtualization, **guest modification** refers to altering the guest operating system or its configuration to improve compatibility, performance, or integration with the virtualization environment or hypervisor. Guest modification modifies the OS to avoid using instructions that virtualize inefficiently. An optimal use of limited resources is warranted to guarantee a high level of utilization.

Processes use different amounts of memory, and their memory needs change over time. Whenever a new process cannot fit into contiguous space in physical memory, it results in fragmentation (specifically, external fragmentation). When the memory blocks cannot be allocated to the processes due to their small size and the blocks remain unused, this problem is called **fragmentation**. There are two types of fragmentation: internal fragmentation and external fragmentation. When the process is allocated a block and its size exceeds the process size, it leaves part of the memory allocated unused and results in internal fragmentation. In the external fragmentation, the total space that is needed for the process is available, but we can't use it because the space is not contiguous.

Linkers and Dynamic Linking

Linkers are software tools that an OS uses to combine object files into an executable file. A linker performs name resolution, matching the name of a variable or function in an application to a virtual memory address it will have when loaded and run. A linker combines many separate pieces of a program, reorganizes storage allocation so that all the pieces can fit together, and touches up addresses so that the program can run under the new memory organization. After a linker completes the task of combining multiple object files generated by a compiler into a single executable file, the executable file can be loaded and executed by the OS (Figure 6.27).

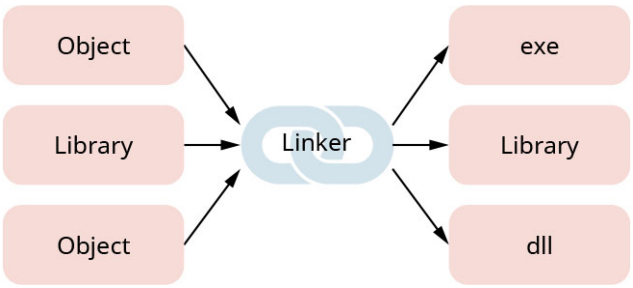


Figure 6.27 A linker process includes object files and libraries. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The mechanism that allows a program to associate external code symbols to addresses at runtime is **dynamic linking**. The allocation process starts when the process is running by dividing the memory into smaller parts called segments. For example, Linux's memory layout is divided such that the code starts at location 0, the data starts immediately above the code, and the stack starts at the highest address, as illustrated in Figure 6.28. When a process is started, the OS will load the file to the memory with the added option of sharing the memory with others. The OS facilitates the memory size at runtime by adding more assigned memory when needed.

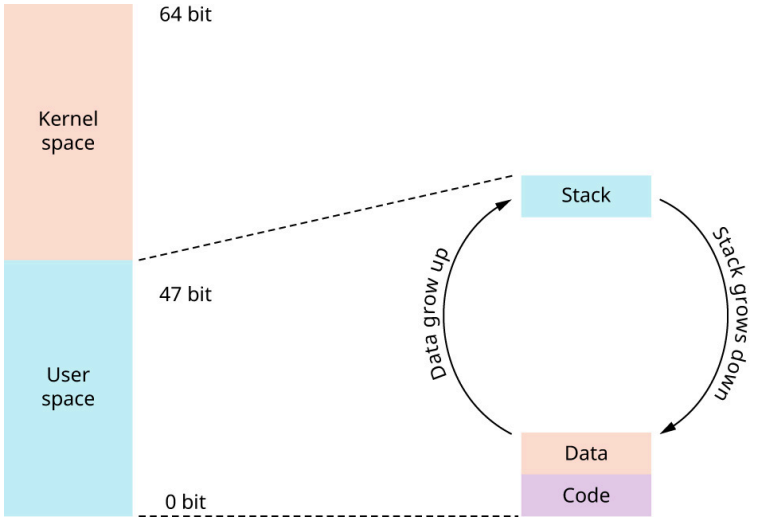


Figure 6.28 In Linux's memory layout, the code starts at location 0, the data starts immediately above the code and grows upward, and the stack starts at the highest address and grows downward. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

In dynamic linking, the code is located and loaded when the program is first run. Since the late 1980s, most systems started supporting shared libraries and dynamic linking by only keeping a single copy of common library packages in memory that is shared by all processes. This means that the system does not know where the library is loaded until runtime and must resolve references dynamically when the program runs.

Dynamic Storage Management

There are two basic operations used in dynamic storage management to manage a memory or storage to satisfy various needs: allocate a block with a given number of bytes or free a previously allocated block. There are two general approaches to applying these dynamic storage allocation operations: (1) stack allocation, which is hierarchical and restricted, but simple and efficient; and (2) heap allocation, which is more general but more difficult to implement and less efficient.

The linear data structure that follows a LIFO order (last in, first out), as in the stack data figure configuration in Figure 6.29, is called **stack allocation**. In the stack approach, the memory is freed in opposite order from allocation. For example, if procedure *X* calls *Y*, then *Y* will certainly return before returning from *X*. Stacks take advantage of this programming practice to store the state of the current procedure call. When memory allocation and freeing are partially predictable, then a stack approach can be used.

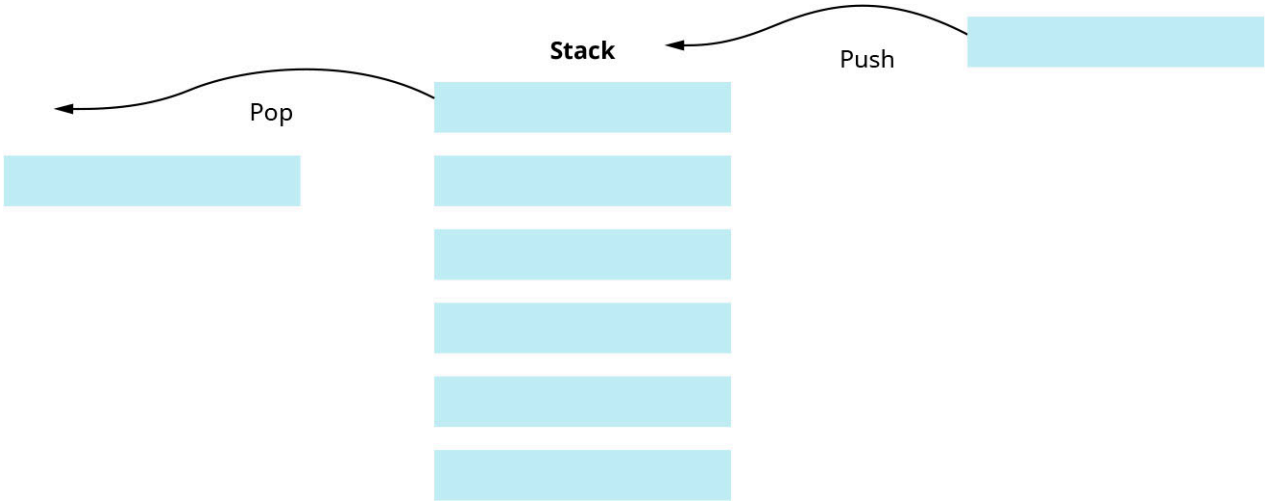


Figure 6.29 This stack data structure representation shows a last in, first out approach. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Allocating the data in a tree-based data structure called a heap is **heap allocation**. A heap is represented by a complete binary tree. As shown in Figure 6.30, a heap data structure can be of two types: max heap and min heap. Max heap presets the root node with the greatest value and the same for the sub trees. It is the opposite for the min heap, where the root will have the minimum value and the same for the sub trees.

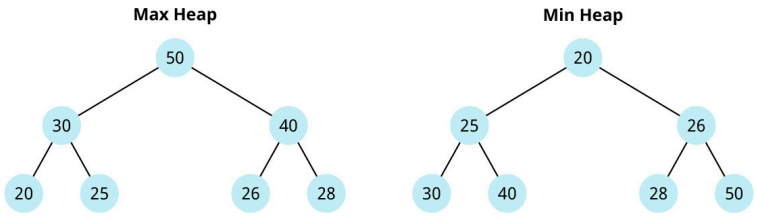


Figure 6.30 Heap allocation uses a data structure called a heap to manage memory and storage. There are two types of heap structures or trees: one for the max heap and one for the min heap. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Memory managers, such as the ones used in C and C++, do not actually store available memory in a heap data structure. Instead, they manipulate a doubly linked list of blocks, which they confusingly refer to as a "heap," and attempt to optimize memory using a "best fit" method.

Memory managers use the heap approach when the allocation and release of memory are not predictable (i.e., when it is not clear how much memory is needed until we run the program). Typically, the heap stores data structures that can change in size over time based on how many elements are added or removed from the data structure. The corresponding heap memory consists of allocated areas and free areas (or holes).

Virtual Memory

The key component of the operating system that ensures process isolation by guaranteeing that each process gets its own view of the memory is virtual memory. A running program (process) has a seemingly infinite view of memory and can access any region without worrying about other programs that might also be running on the computer. The OS seamlessly translates each process memory request into a separate region of the physical hardware memory through address translation. When the system needs to find a physical address in the memory that matches the virtual address, **address translation** occurs. The running process only deals with virtual addresses and never sees the physical address. Virtual memory is mapped to physical memory in units called "pages."

There is a time cost associated with performing virtual-to-physical memory address translation, however, and this can add up given that most programs need access to the memory to store data. To speed up address translation, the CPU has dedicated hardware for caching (storing) recent address translations called a **translation lookaside buffer (TLB)**. A TLB is a memory cache that stores the virtual memory recent transaction to physical memory. TLBs help the CPU avoid making multiple round trips to main memory just to resolve a single virtual memory access by only requiring one round trip (Figure 6.31).

A TLB contains page table entries that have been most recently used. Given a virtual address, the processor examines the TLB table. If a page table entry is present, it's a "hit." This means the frame number is retrieved, and the real address is formed. If a page table entry is not found in the TLB, then it's a "miss." In this case, the page number is used as an index while processing the page table. TLB checks if the page is already in the memory; if it's not, then a page fault is issued and the TLB is updated to include the new page entry.

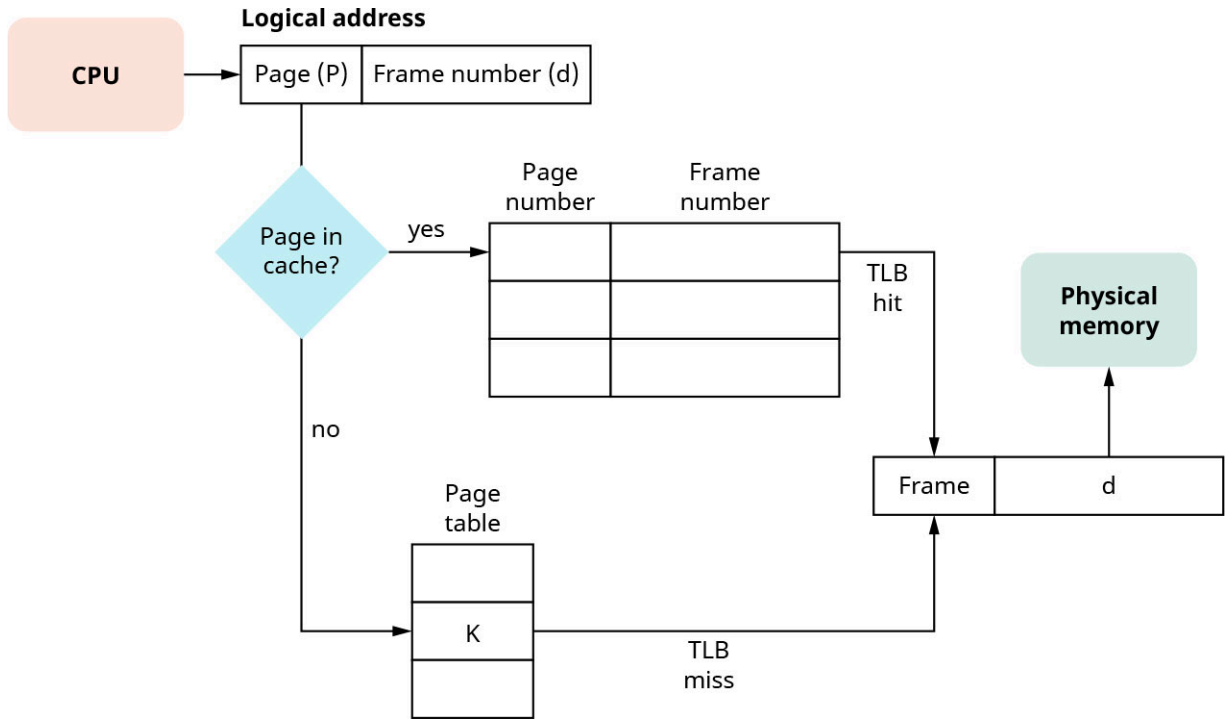


Figure 6.31 Translation lookaside buffers (TLBs) speed up address translation by using an approach that involves detecting "hits" (which means a page table entry is present in the TLB) and "misses" (a page table entry is not found in the TLB). (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

LINK TO LEARNING

What is the fundamental concept that makes it possible to implement virtual memory? Check out an explanation of how to implement virtual memory (https://openstax.org/r/76virtmem) to investigate this question.

Demand Paging

The storage mechanism that uses a page form in retrieving a process from secondary or virtual memory to main memory is called **paging**. Virtual memory presented a seemingly infinite amount of memory to the running process, but what happens when the operating system runs out of free physical memory? Modern operating systems also have a backup when DRAM runs out, which means virtual memory can be mapped to disk to meet demand. The storage mechanism in which pages should only allocate in the memory if it is required from the execution process is called **demand paging**. Figure 6.32 shows a CPU that is demanding pages from the virtual memory to the main memory (i.e., swap in) and releasing pages from the main memory to the virtual memory (i.e., swap out). The **working set size (WSS)** refers to the total amount of memory a process requires during a specific period of activity, measured as the set of pages or data blocks the process accesses. WSS is measured by tracking the unique pages a process references over a fixed interval of time. This provides an estimate of the process's active memory footprint and helps in memory management decisions like paging and swapping to optimize performance and resource allocation.

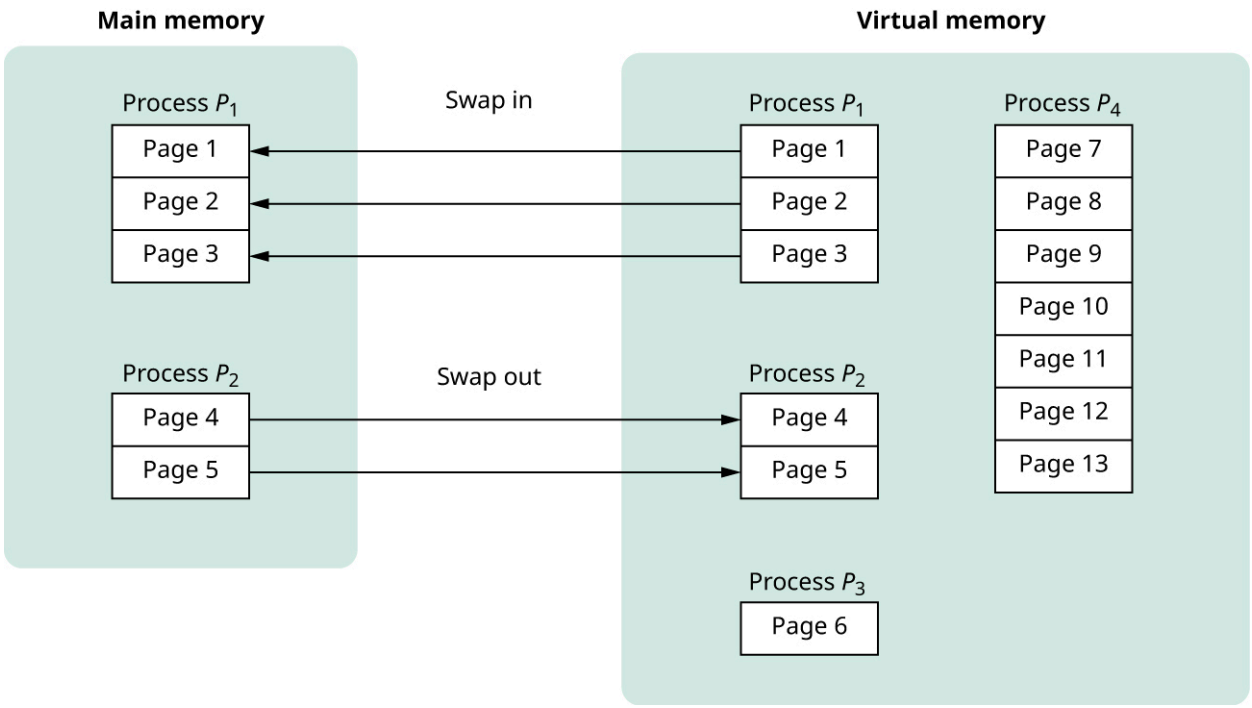


Figure 6.32 In demand paging, a CPU retrieves pages from the virtual memory to the main memory and releases pages from the main memory to the virtual memory. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

When the CPU demands a page and this page is not present in the main memory, we call this situation a **page fault**. A page fault occurs when a process references a page that is in the backing store. To handle a page fault, the CPU transfers control from the program to the OS to demand the requested page to the main memory. OS finds a free page frame in memory, loads the page from the backing store to the main memory, and resumes execution of the thread. The CPU has special hardware to assist in resuming execution after a page fault.

Given that access to the disk is much slower than DRAM, operating systems are often designed to predictively swap in-use pages into DRAM and out-of-use pages to disk. The process of bringing pages into memory (i.e., demand paging) is called **page fetching**. Most modern OSs use page fetching by starting the process with no pages loaded and do not load a page into memory until it is referenced. If a requested page is stored on the disk, prefetching, which is the act of trying to predict when pages will be needed and loading them ahead of time to avoid page faults, is performed.

If all memory is in use, it is necessary to throw out one page each time there is a page fault. This process is called **page replacement**. In page replacement, one page in the (full) DRAM is swapped to disk while the requested page is brought into DRAM. However, if too many processes need access to a lot of memory back and forth between DRAM and disk, this causes problems. For example, each page fault causes one of the active pages to be moved to disk, so another page fault soon occurs, and this leads to **thrashing**. Thrashing is when a computer's operating system becomes overwhelmed by the number of processes requesting memory. This situation leads to a cycle where the system spends more time moving data between the physical memory and disk storage (paging or swapping) than executing actual processes. It's like a busy restaurant where the staff spends more time rearranging tables than serving food. The main cause of thrashing is often that too many programs are running at the same time. These activities exceed the available memory, causing the system to constantly try to make space for new requests by moving data to and from the disk.

Strategies to prevent thrashing include limiting the number of simultaneously running programs to avoid memory overcommitment, optimizing how memory is allocated to processes, and possibly increasing the system's physical memory. By managing memory more efficiently and ensuring that the system is not overloaded with too many tasks, the system can not only avoid a slowdown, but significantly improve its

performance.

In extreme cases of thrashing, the OS can spend all its time fetching and replacing pages and will not get much work done. This is one reason why our devices can slow to a halt when they run out of memory and each thread must wait on requested pages.

CONCEPTS IN PRACTICE

Impatience with Computers

We've all been there: At our laptops, putting the final touches on a slide presentation, while checking email, while uploading a photo to our social media feed, while listening to a new playlist our friend just shared with us, while watching a video, while inputting data into a spreadsheet, and everything freezes—the screen, the keyboard, the trackpad. Not only does nothing do what we ask it to do, it all just goes still. Or worse, the little pinwheel starts spinning and never stops. Now having read about memory, dynamic storage management, resource allocation, and thrashing, think about what's happening inside your CPU.

If you're using a computer that runs Windows, check out this resource to see how you could help your operating system (https://openstax.org/r/76WindowsOS) operate better.

6.5 File Systems

Learning Objectives

By the end of this section, you will be able to:

- Explain features of various file systems
- Discuss file system structures and layers

In this module, we learn about files, file management, disk devices, file systems, file system interface, and distributed file systems.

Files, File Systems, Directories, and File Management

A **file** is a collection of related information that is stored on a storage instrument such as a disk or secondary/virtual storage. It is the smallest storage unit from the user's perspective. The file name includes two parts: name and extension (e.g., filename.txt). Each extension is for a specific purpose such as .exe (in Windows OS) to run a program and .txt for text files.

A **file system** is responsible for defining file names, storing files on a storage device, and retrieving files from a storage device. When designing a file system for managing many files, some issues to consider are as follows: most files are small so per-file overheads must be low; most of the disk space is in large files; many of the I/O operations are for large files so performance must be good for large files; files may grow unpredictably over time; users want to use text names to refer to files.

Special disk structures called directories are used to map names to support hierarchical directory structures. A **directory** is a set of files that is managed by the OS, and it also contains all the required information about the files, such as attributes, location, and ownership. The UNIX/Linux approach is as follows: directories are stored on disk just like regular files except with extra information to indicate that it is a directory. Each directory contains <name, address> pairs. The file referred to by the address may be another directory; hence, we can have nested and hierarchical directory structures.

The problems facing modern file systems include disk management, naming, and protection. File systems are often trying to improve access to files by minimizing seeks, sharing space between users, and making efficient use of disk space. A system's ability to reduce faults and ensure that the information in the system survives OS

crashes and hardware failures is called its **reliability**. In addition to improving reliability, a file system should guarantee a high level of protection by maintaining isolation between users and controlling the sharing of resources.

Disk Devices

While file systems are a layer of abstraction that provides structured storage and defines logical objects such as files and directories, disk devices are considered raw storage. Data that can be directly accessed by the CPU with minimum or no delay and does not survive a power failure is held in **primary storage**. Persistent memory that survives power failures most of the time, such as spinning disks, SSDs, and USB drives, is considered **secondary storage**. Routines that interact with disks are typically at a very low level in the OS and are used by many components such as file systems and virtual machines. These secondary storage devices may handle the scheduling of disk operations, error handling, and often the management of space on disks. A trend is for disks to do more of this themselves.

File System Architectures

Operating systems use various methods to locate files by their names, and the methodology often depends on their underlying file system architecture. To illustrate these concepts, here are some examples from UNIX-like systems and Windows:

- UNIX/Linux (Inodes): In UNIX-like systems, the file system uses a structure called an **inode** to represent files and directories. An inode contains metadata about a file or directory but not its name. The name-to-inode mapping is stored in directories, which are special files that list names of files and their corresponding inodes. When searching for a file by name, the OS starts at the root directory and follows the path specified in the file name. Each part of the path is looked up in the current directory's list of names and their associated inodes. The OS reads the directory file, finds the name, and retrieves the inode number, which then leads to the inode itself. The inode provides the location of the data blocks, allowing the OS to access the file's data. This process may involve multiple steps if the file is in a nested directory structure.
- Windows (File Allocation Table and NTFS): In File Allocation Table (FAT) format, files are located using a table that maps file names to the clusters (blocks) on the disk where their data is stored. The FAT is essentially a list, with each entry containing the location of the next part of the file. This creates a chain that the OS follows to read the entire file. In New Technology File System (NTFS), files are located using a Master File Table (MFT), and each file and directory on an NTFS volume has an entry in the MFT containing data, including the file name, size, time stamps, permissions, and the locations of the file's data on disk. When searching for a file, the OS consults the MFT to find the entry corresponding to the file name, which then provides the information necessary to access the file's data.

As you may recall, files represent values stored on disk and directories represent file metadata. File systems define operations on objects such as create, read, and write, and they may also provide higher-level services such as accounting and quotas, incremental backup indexing or search, file versioning, and encryption. A **quota** is the amount of space to store files based on the available memory space. Quotas are used to protect the system from unnecessary load and help in organizing the data in the storage. An **incremental backup** is a backup image containing the pages that have been updated from the time of the previous backup. The method that converts the data into secret code that hides the data's true meaning is called **encryption**. The system that allows a file to exist in several versions at the same time, which gives the user complete control over file creation as in the file versioning example, is called **file versioning** (Figure 6.33).

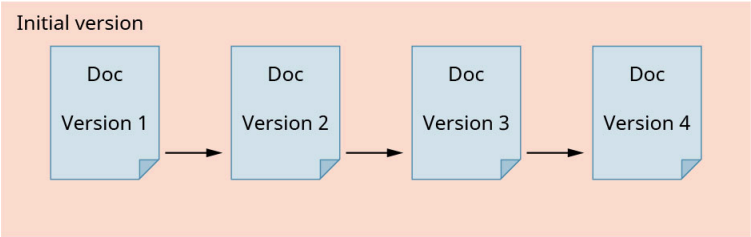


Figure 6.33 In file versioning, the OS saves all copies of a file (in this case, a doc or document file) . (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

File systems are concerned with lower-level characteristics such as performance and failure resilience. Both performance and failure resilience may be strongly affected by hardware characteristics.

File System Interface

In general, the file system interface defines standard operations such as file (or directory) creation and deletion, manipulation of files and directories, copy, and lock. Remember the various file attributes are name, type, size, and protection. The file system uses these attributes to provide system calls for the following operations:

- Create: Find a space for the file in the disk and enter the new file information in the directory.
- Write: Search the directory for a specific file and start writing from where the writing pointer is pointing to.
- Read: Specify the name of the file and start reading from where the reading pointer is pointing to.
- Seek: Search for the specific byte position in the file.
- Delete: Search for the file in the directory and erase the file from the directory.
- Truncate: Reset the file length to zero and release the allocated space for the file.
- Append: Add new information to the end of the file.
- Copy: Create a new file and read the data from an old file, then write it to the new one.

If multiple processes are trying to open a file at the same time, then there is a role for file management that should be applied—namely, lock.

LINK TO LEARNING

As people and businesses use various types of computers today, the ability to interchange files between various file systems is critical. In fact, insurance companies often ask their clients to sign insurance contracts over the Internet and rely on their digital signature in online documents. Teachers use Google Classroom to enable their students to collaborate on assignments or class presentations. Read this overview of common file systems (https://openstax.org/r/76comfile) to see how file systems enable us to seamlessly share documents and files.

Inodes

As mentioned earlier, inodes are OS data structures used to represent information about files and folders stored on disk along with file data and kept in memory when the file is open. An inode contains information including file size, sectors occupied by file, access times (e.g., last read and last write), and access information (e.g., owner id and group id). In Linux, whenever the system creates a new file, it gives it an inode unique number called **i-number**. Internally, the OS uses the i-number as an identifier for the file—in effect, as its name. When a file is open, its inode is kept in main memory. When the file is closed, the inode is stored back to disk. If you are using Linux, you can check the total number of inodes on disk using the df command and –i option, as shown in Figure 6.34.

Input:

```
$df –i /dev/sda
```

Output:

Filesystem	Inodes	IUsed	IFree	IUse%
/dev/sda	1624000	128000	1496000	9%

Figure 6.34 In the Linux OS, the total number of inodes on the directory /dev/sda can be viewed using the command df and the option –i. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

File systems are responsible for managing parts of the disk that are used (inodes) and parts of the disk that are not used (free blocks). Each file system has different strategies and approaches for managing this information, with different trade-offs. Additional features of file systems include file system–level encryption, compression, and data integrity assurances.

Distributed File Systems

A **distributed file system (DFS)** is a file system that is distributed on multiple file servers or multiple locations that support network-wide sharing of files and devices. The presentation of a DFS is similar to the traditional view (i.e., client is using a file system). The main idea of a DFS is that it uses a namespace, which means all clients see a single namespace where files and directories are shared across the network. In a DFS, clients can read and write files on a remote machine as if they were accessing their local disks. A DFS provides an abstraction over physical disks that is akin to the abstraction virtual memory provides over physical memory (Figure 6.35).

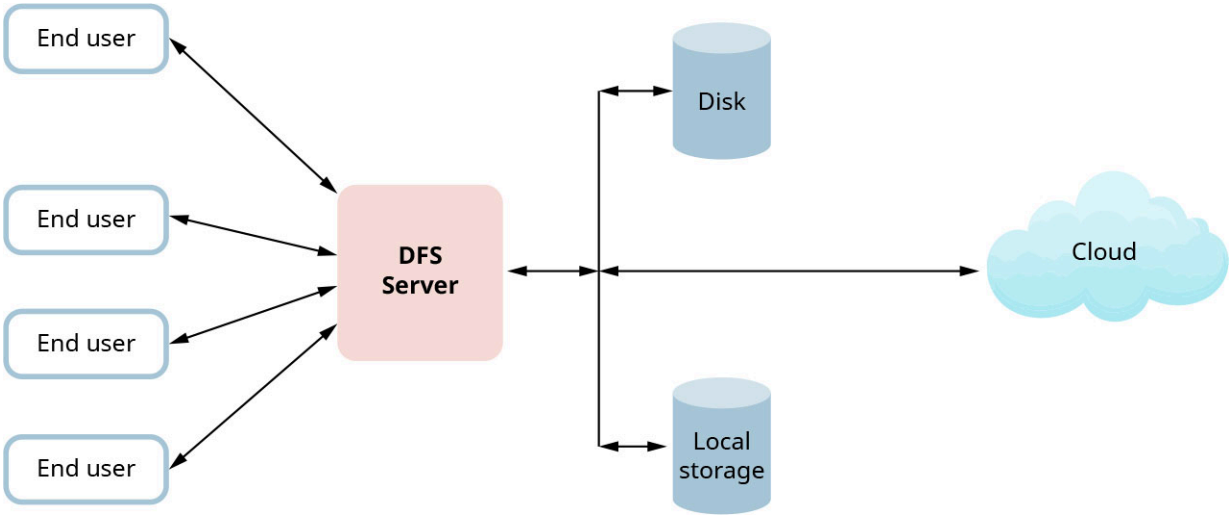


Figure 6.35 In a distributed file system architecture, the DFS server works like a middleman between the end user and the data, which can be in any storage format. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

DFS technologies like Google's GFS (Google File System), Apache Hadoop's HDFS (Hadoop Distributed File System), and Apache Spark's RDDs (Resilient Distributed Datasets) have revolutionized the way we handle and process large volumes of data. These systems are designed to accommodate High Throughput Computing (HTC), complementing the capabilities of High-Performance Computing (HPC) by focusing on the efficient processing of vast datasets across clusters of computers.

- Google File System (GFS) is a prime example of a DFS that is highly optimized for large-scale data processing. It is designed to provide high fault tolerance while running on low-cost commodity hardware.
- Hadoop Distributed File System (HDFS) follows a similar principle but is open-source and commonly associated with the Hadoop ecosystem. It's designed to store very large files across machines in a large cluster and to stream those files at high bandwidth to user applications. By breaking down files into blocks

and distributing them across a network of computers, HDFS can process data in parallel, significantly speeding up computations and data analysis tasks.
- Resilient Distributed Datasets (RDDs) in Apache Spark are a further step in distributed computing, offering an abstraction that represents read-only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost. Spark's use of RDDs allows it to process data in-memory, which is much faster than the disk-based processing used by Hadoop, making Spark an excellent choice for applications requiring quick iterations over large datasets.

To facilitate the communication necessary in these distributed environments, protocols such as Remote Procedure Call (RPC) and Distributed Hash Tables (DHTs) are employed. RPC is a protocol that a program can use to request a service from another program located in another computer in another network without having to understand the network's details. DHTs are a class of decentralized distributed systems that provide a lookup service similar to a hash table; keys are mapped to nodes, and a node can retrieve the content associated with a given key.

Beyond these, the concept of N-Tier distributed file systems, such as the Network File System (NFS), plays a foundational role. NFS allows a system to share directories and files with others over a network. By using NFS, users and programs can access files on remote systems almost as if they were local files.

The basic abstraction of a remote file system is via open, close, read, and write. As it comes to naming, the names are location transparent. Location transparency hides the location where in the network the file is stored. The procedure that allows multiple copies of a file to exist in the network is called **replication**. This improves performance and availability. DFS handles the updates, checks if clients are working on separate copies, and performs reconciliation.

LINK TO LEARNING

Distributed file systems are used worldwide in a range of industries, from banking to health care. Read this brief tutorial on distributed file systems (https://openstax.org/r/76distrfile) and name two advantages as well as two disadvantages of using DFSs.

Flash Memory

Flash memory is used for general storage and the transfer of data between computers and other digital products. Many of today's storage devices, such as SSDs, utilize flash memory, which offers considerable performance improvements over traditional mechanical hard disk drives (HDDs). The performance improvements of flash-based storage devices like SSDs come from their ability to access data much faster than mechanical drives. Here's why:

- No moving parts: Unlike HDDs that use rotating disks and read/write heads, SSDs have no mechanical parts. This not only increases durability, but also means that data can be read from and written to the drive much faster.
- Random access: Flash memory allows random access to any location on the storage, making it much quicker at reading data that is scattered across the drive. HDDs need to physically move the read/write head to the data location, which takes more time.
- Faster read and write speeds: SSDs can handle rapid read and write operations. This is especially beneficial for applications that require quick access to large amounts of data, such as video editing, gaming, and high-speed databases.
- Lower latency: Because they lack a physical read/write head that needs to be positioned, SSDs significantly reduce the time it takes for a storage device to begin transferring data following an I/O request.
- Improved durability and reliability: With no moving parts to wear out or fail, SSDs are generally more reliable and can better withstand being dropped or subjected to sudden impacts.

- Lower power consumption: SSDs consume less power, which can contribute to longer battery life in laptops and less energy use in data centers.

GLOBAL ISSUES IN TECHNOLOGY

Global Distributed File Systems

Distributed file systems enable companies that operate globally and handle vast amounts of data from many different sources and in many different ways, such as the following:

- To store and manage that data in a cloud
- To scale up their operations as needed
- To enable users across the world to access the data seamlessly
- To use encryption and other protection mechanisms to secure sensitive data
- To ensure that data is regularly backed up and can be recovered if there's a disaster

6.6 Reliability and Security

Learning Objectives

By the end of this section, you will be able to:

- Explain how OSs protect computer systems
- Discuss key security-related functions of the OS
- Explain how the OS helps the computer system recover from failures
- Discuss how advances in technology affect the longevity of an OS

Remember that we consider an OS to be reliable if it delivers service without errors or interruptions. In addition to reliability, an OS should provide a high level of protection, security, and stability. Here, we learn about OS protection, security, recovery, and longevity.

Protection

The general mechanism that is used throughout the OS for all resources that need to be protected, such as memory, processes, files, devices, CPU time, and network bandwidth is called **protection**. The objectives of the protection mechanism are to allow sharing (which in this context means using the hardware to do more than one thing at a time), help detect and contain accidental or unintentional errors, and prevent intentional/malicious abuses. The main challenge when it comes to protection is that intentional abuse is much more difficult to eliminate than accidents.

There are three aspects to a protection mechanism: authentication, authorization, and access enforcement. Authentication identifies a responsible party or principal behind each action, authorization determines which principals are allowed to perform which actions, and access enforcement controls access using authentication and authorization information. A tiny flaw in any of these areas can compromise the entire protection mechanism. It is extremely difficult to make all these protection mechanism techniques operate in such a way that there are no loopholes that can be exploited by adversaries. Figure 6.36 illustrates the relationship between authentication, authorization, and access enforcement.

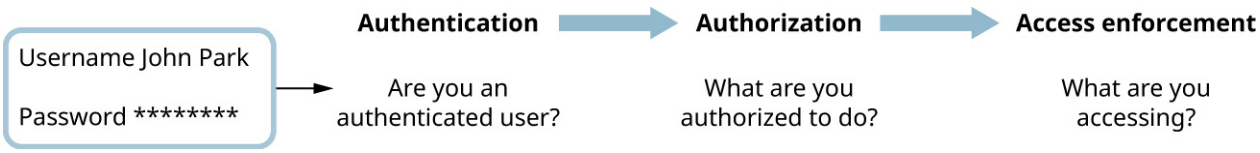


Figure 6.36 The first step of the protection mechanism is authentication, which checks the username and password; then comes authorization, which checks the privileges; and, finally, there is access enforcement, which controls access. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Security

The process of checking to see if a user's credentials match the credentials in a database of authorized users or in a data authentication server is called **authentication**. The traditional means of authentication involves the user providing a password, which is a secret piece of information that is used to establish the identity of a user and should be relatively long and hard to guess. Most systems store the passwords in a password database. A password database must be protected because it is vulnerable most of the time. For example, both organizations and users should avoid storing passwords in a directly readable form.

An alternate form of authentication involves using a badge or key. The **badge** is a logical access system. The badge does not have to be kept secret. It can be counterfeit, but if it is, the owner will know. The badge must be cheap to make but hard to duplicate.

Another form of authentication is **two-factor authentication**, which involves two factors: the system calls or texts a user's phone for the traditional password during login, employing the cell phone as a key. For example, a site sends a text message to a user's phone with a one-time passcode. The user must read the passcode from the phone and type it into the login page.

In two-factor authentication, an attacker must have both your password and cell phone to hijack your account. This approach is particularly effective for authenticating to websites, as the requiring of both the password and the physical cell phone is a sufficient deterrent. To enhance efficiency, the two-factor authentication process can be optimized for websites. Once the authentication is completed, a cookie is loaded into your browser. This cookie then transforms your browser into a type of key, granting you the ability to log in with the password as long as the cookie persists.

Whenever a user needs to log in from a different browser or different machine, two-factor authentication is used again. After logging in, the user id is associated with every process executed under that login because the user id is stored in the process control block and children inherit the user id from their parents. Once authentication is complete, the next step of protection is authorization.

The process of determining the relationship between principals, operations, and objects by defining the kind of principals that are allowed to carry out a specific activity with a defined set of objects is called **authorization**. These principals are represented using a matrix that includes an entry for each principal and a column for each object as a representation of authorization information on given operations. For example, defining who has the authorization to read/view, edit, or delete the file. Each entry in the access matrix describes the capability of each principal over each one of the objects. When the matrix includes all of the principals and all of the assigned objects, it can become complex and hard to manage. The ideal way to solve this problem is to use a guideline such as an access control list. An **access control list (ACL)** is a set of guidelines that outline the authority of each user (i.e., which user is permitted access to given resources). An ACL controls the access and privileges using a matrix design.

The ACL from Oracle features the assignment of users to roles such as basic users, advanced users, customer administrator, among others. A role is configured to confer privileges on objects rather than attaching privileges to individual users, as this would be much more difficult to set up and maintain. The most general form of setting privileges is creating a list of user and privilege pairs, which is called a capability list. A **capability list** is a list of objects and operations for each user that defines the user rights and capabilities. Typically, capabilities also act as names for objects, which means the list cannot even name objects not referred to in your capability list. For simplicity, users can be organized into groups with a single ACL entry for an entire group, and each group can be made to share the same privileges. While in Windows OS, ACLs are very general, they are relatively simple in UNIX/Linux. For example, in UNIX/Linux, access can be read, write, and execute, and it can be granted to the file owner, the file owner's group, or "the world" of all users. In many cases, the user root has full privilege for all of the operations and has access to all of the permissions. For example, the user root can view, edit, and delete a file. ACLs are straightforward and can be utilized by any file

systems in Windows. The utilization involves sharing a namespace at a high level of visibility by making it public, while defining another namespace as private—akin to the encapsulation of objects in object-oriented programming.

One component of an OS must be in charge of enforcing access rules and safeguarding authentication and authorization to provide a high level of security. The system's **access enforcement** mechanism has complete authority; therefore, it must be simple in programming and small in size. The security kernel is a substitute approach that is composed of hardware and software and serves as the OS's inner protection layer. Generally, any kind of management such as memory and interrupt management are provided by a security kernel.

LINK TO LEARNING

Every once in a while, you may get a notification on your computer asking you to update your OS, and the update may include a "patch" to address a security issue. Often, this security issue is related to a cyberattack that is exploiting some vulnerability in the OS. Check out this tutorial on OS vulnerabilities (https://openstax.org/r/76OSvulnera) to gain a deeper sense of the kinds of OS vulnerabilities that these attacks target.

Recovery

Like any other system, an OS can crash in the middle of critical sections or while the system is running. These crashes may result in lost data, unexpected results, and inconsistency. For example, if the crash happened before the system had stored a user's information in the main memory, the system will have lost this information. Unexpected results provide the wrong output and may affect other calculations.

An **inconsistency** is a situation that causes the system to produce errors or hardware failure. Inconsistencies may occur when a modification affects multiple blocks; a crash may occur when some of the blocks have been written to disk but not the others. For example, when the system adds a block to a file, it updates the free list to indicate that the block is in use, but if the inode is not yet written to point to the block, this will result in an inconsistency. Another inconsistency can occur when the system while creating the link to a file to make a new directory entry refers to an inode, but the reference count has not yet been updated in the inode.

The process of resolving OS faults or errors is called **recovery**. Three approaches that can address inconsistency issues include:

- Check consistency during reboot, and repair problem. A good example of checking for inconsistency is the file system check (fsck) command implementation for UNIX and UNIX-like file systems. The system executes fsck as part of every system boot sequence so it can check whether the system was shut down correctly or not. If it was properly shut down, it proceeds normally. In the alternative (e.g., crash, power failure, or any other reason), the recovery process will start. The recovery process will scan disk contents, identify inconsistencies, and repair them. The limitations of fsck are as follows: it will restore disk to consistency, but does not prevent information loss. This loss of information can lead to instability. Also, the fsck has security issues because a block could migrate from the password file to some other random file, which could make it visible to unauthorized users. In addition, running fsck may take a long time, and the user will not be able to restart the system until fsck completes. The recovery process with fsck will take more time if the disk size is big. Figure 6.37 illustrates an example of the code errors produced from fsck and the meaning of each code in the Linux OS.

```
# man fsck

0       No errors
1       Filesystem errors corrected
2       System should be rebooted
4       Filesystem errors left uncorrected
8       Operational error
16      Usage or syntax error
32      Checking canceled by user request
128     Shared-library error
```

Figure 6.37 Running a recovery process with fsck resulted in these code errors. The meaning of each code in Linux OS is also given. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

- Check the order of the writes. This approach avoids some discrepancies by applying changes in a specific write sequence. For instance, to ensure the free list doesn't still contain the file's new block, write the content of the list before adding that block to the file. After ensuring that the list is not including that block, create a reference for the new block in the inode. Using this approach, you'll guarantee that you'll never write a pointer prior to initializing the block to which it points without validation. The validation will force the system to never clear the last pointer prior to setting a new pointer. The advantage of this approach is that it reduces the time spent waiting, as there is no need to wait for fsck while rebooting. However, there are several drawbacks, such as the potential for resource leaks (e.g., when the system runs fsck to recover some lost resources). Another drawback is that this approach slows file operation because writing while running the system requires considerable metadata.
- Perform write-ahead logging. This term is known as journaling file system and refers to the practice of recording the changes in the information in a separate log file sometimes called a journal file. These changes will be recorded prior to any new change or update on the system. Windows NTFS and Linux ext3 implement this kind of log file. The log procedure is analogous to the way log files are used in a database system to enable the correction of updated inconsistencies, which enables the healing quickly in case of any error. Prior to performing any operation, the recovery process will initially store information regarding the operation in a special log file. The next step is to flush the information to the disk before updating any other blocks. For example, a log entry such as "I'll add block 100101 to inode 313 at index 90" will be added to the system's log in case the operation involves adding a block to a file. This will guarantee that the actual block modifications can be performed. The system will restore the log in case of any crash to ensure that all of the updates have been saved in the disk. There are many benefits to employing logging such as reducing the time needed to recover from any failure. Also, improving the ability of localizing logs in the disk will result in improving the system's performance. However, this approach has some drawbacks, namely, the size of the log file will grow over time, and this will affect the system's processing time. This problem can, however, be resolved by performing periodic checkpoints.

Longevity

How long does an OS last? Did companies stop developing new OSs? How can the current OS survive? To answer these questions, we need to discuss concepts such as paging, TLBs, disks, storage latency, and multicores as well as virtual machines (VM).

Technology and OSs

Many of the basic ideas in OSs were developed 30–50 years ago, when technology was very different. The question is not only whether these ideas will still be relevant in the future, but whether they are relevant even today. After all, technology has changed considerably over the last thirty or so years. For example, CPU speeds went from 15 MHz in 1980 to 2.5+ GHz in 2024, a 167-fold increase. Memory size went from 8 MB to 16+ GB, a 2,000-fold increase. Disk capacity went from 30 MB to 2+ TB, a 6,667-fold increase. Disk transfer rate went from

2 MB/sec to 200+ MB/sec, a 100-fold increase. Network speeds went from 10 Mb/sec to 10+ Gb/sec, a 1,000-fold increase. As you can see, there were huge increases in size, speed, and other capabilities.

As you may recall, paging is a storage mechanism that allows processes to be retrieved from secondary memory and moved to main memory using pages. In the 1960s, paging originally touted disk speed latency of 80 ms, a data transfer rate of 250 KBs/sec, memory size of 256 Kbytes. Thus, for 64 pages, it took 6.4 sec to replace all of the memory to address individual page faults and 1 sec to address sequential page faults. Today, we have disk speed latency of 10 ms, a data transfer rate of 150+ MB/sec, and memory size of 64+ GB. For 16,000,000+ pages, it takes 44+ hours to replace all of memory to address individual page faults, and 320+ sec to address sequential page faults. Therefore, we cannot afford to page something out unless the system is going to be idle for a long time. But the real question is: does paging make sense anymore as a mechanism for the incremental loading of processes? The answer is yes, but by reading the entire binary at once because 15 MB of binary takes 0.1 sec to read.

TLBs have not kept up with memory sizes; 64 entries provide 256 KB coverage. In the mid-1980s, this was a substantial fraction of memory (i.e., 8 Mbytes). Today, TLBs can only cover a tiny fraction of memory. Some TLBs support larger page sizes of 1 Mbyte or 1 GB, but this complicates kernel memory management.

Disk capacity has increased faster than access time; storage access latency for disks is around 10 ms, and it is around 100 µs for flash memory. There are new nonvolatile memories, such as Intel's 3D XPoint, that improve the latency to 100ns–300ns.

Chip technology improvements allowed processor clock rates to improve rapidly. Unfortunately, however, faster clock rates mean more power dissipation, and now power limitations limit improvements in clock rate. Chip designers are now using technology to put more processors (cores) on a chip. In general, all OSs must now be multiprocessor OSs. However, it is not clear how to utilize these cores, and application developers must write parallel programs, which is very hard.

Lastly, the current/hot trend for OS development is the data center, which coordinates thousands of machines working together trying to achieve very low-latency communication.

LINK TO LEARNING

As nearly every person and business on the planet uses computers today, their reliability and security are increasingly essential. At the same time, there is growing concern about whether the underlying technologies we are relying on to power OSs will become obsolete soon. And there are also questions about what will replace OSs. Check out the debate on what will replace OS (https://openstax.org/r/76replaceOS) and see whether you share any of the concerns.

Virtual Machines

As you learned earlier in this chapter, a virtual machine is a software emulation of a physical computer that creates an environment that can execute programs and manage operations as if it were a separate physical entity. This emulation allows multiple operating systems that are isolated from each other to run concurrently on a single physical machine. In essence, a VM provides the functionality of a physical computer, including a virtual CPU, memory, hard disk, network interface, and other devices.

Recall that the underlying technology enabling VMs is called a hypervisor or virtual machine monitor (VMM). This technology resides either directly on the hardware (Type 1 or bare-metal hypervisor) or on top of an operating system (Type 2 or hosted hypervisor). The hypervisor is responsible for allocating physical resources to each VM and ensuring that they remain isolated from each other. This isolation ensures that processes running in one VM do not interfere with those running in another and thereby enhances security and stability. VMs are widely used for a variety of purposes, including server virtualization, software testing and

development, and desktop virtualization. Virtual machines have become a fundamental component of cloud computing, as they allow cloud providers to offer scalable and flexible computing resources to users on a pay-as-you-go basis.

Figure 6.38 illustrates the difference between a Type 1 virtual machine monitor and container environment such as via Docker. A **container** is a standardized unit of software that logically isolates an application, enabling it to run independently of physical resources.

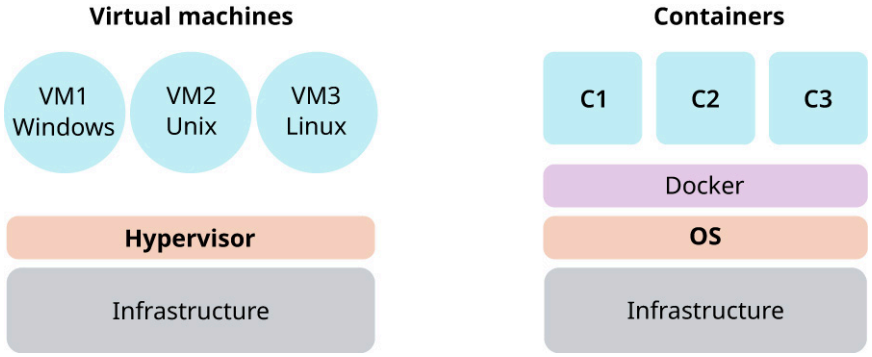


Figure 6.38 One notable difference between the virtual machine and containers is that VMs allow for the use of multiple operating systems, whereas containers share a single OS. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

When the complete OS is running within a VM, then the system will be called a **guest operating system**. VMs are heavily used in cloud computing such as Microsoft Azure, Amazon Web Services, Google Cloud Platform, and IBM Cloud.

THINK IT THROUGH

VMs vs. On-Premise Computing

VMs on the cloud represent a paradigm shift in how we utilize computing resources, offering compelling advantages over traditional on-premises computing. Cloud-based VMs provide scalability, flexibility, and cost-efficiency, making them a promising technology for businesses and individuals alike. In a traditional on-premises setup, a company or user must invest in physical hardware, maintain that hardware, and often overprovision resources to handle peak demand periods. This approach ties up capital and resources in equipment that may quickly become outdated or underutilized.

Why are virtual machines on the cloud a promising technology as compared to on-premises use of a computer?

Chapter Review

Key Terms

access control list (ACL) list of rules that specifies which users are granted the access to a specific object or resource

access enforcement part of the OS that is responsible for enforcing access controls and protecting authentication and authorization information

address space set of addresses generated by programs as they reference instructions and data

address translation stage of virtualization that occurs when the system needs to find a physical address in the memory that matches the virtual address

allocation method that defines how data is stored in the memory by providing a set of requests for resources and identifying which processes should be given which resources to make the most efficient use of the resources

application programming interface (API) set of rules and tools that allows different software applications to communicate with each other

authentication process of checking to see if a user's credentials match the credentials in a database of authorized users or in a data authentication server

authorization process of determining the relationship between principals, operations, and objects by defining which principals can perform which operations on which objects

badge logical access system that serves as a form of authentication

blocked state when a process is waiting for an event to occur

cache-only memory architecture (COMA) computer memory architecture where all memory is treated as cache, which allows for the dynamic allocation of data for optimized access and performance in multiprocessor environments

capability list list of objects and operations for each user that defines the user rights and capabilities

cloud infrastructure virtualized and scalable hardware resources delivered over the Internet as a service

concurrency multiple activities and processes happening at the same time—in other words, the OS handling multiple tasks at once

concurrent processing computing model that improves the performance when multiple processors execute instructions simultaneously

condition variable synchronization mechanism that is used in conjunction with locks to allow threads to wait for certain conditions to become true

container standardized unit of software that logically isolates an application enabling it to run independently of physical resources

context switch procedure that a computer's CPU follows to change from one task to another while ensuring that the tasks do not conflict

CPU state consists of the program counter (PC), the stack pointer (SP), and general-purpose registers

critical section piece of code that only one thread can execute at once; also, only one thread at a time will get into this section of code

deadlock synchronization challenge that occurs when a set of threads are blocked forever, waiting for each other to release resources

demand paging storage mechanism in which pages should only allocate in the memory if it is required from the execution process

device manager layer in the layered OS architecture that handles devices and provides buffering

device register interface view any device presents to a programmer where each I/O device appears in the physical address space of the machine as a few words

directory set of files that contains all the required information about the files such as attributes, location, and ownership, which is managed by the OS

distributed file system (DFS) file system that is distributed on multiple file servers or multiple locations that

support network-wide sharing of files and devices

dual mode OS structure that is responsible for separating the user mode from the system mode

dynamic linking code is located and loaded when the program is first run

encryption method that converts the data into secret code that hides the data's true meaning

exception error that occurs at runtime

exokernel OS architecture that simplifies the operating system by making its core (kernel) really small and letting apps have more control over the computer's hardware

fault containment feature of an OS that prevents errors in one part of an application from affecting the whole system

fault recovery feature of an OS that helps the system to fix itself or revert to a previous state after an error

fault tolerance feature of an OS that allows the application to keep running even when errors occur

file collection of related information that is stored on secondary/virtual storage; it's the smallest storage unit from the user's perspective

file system responsible for defining file names, storing files to a storage device, and retrieving files from storage devices

file versioning system that allows a file to exist in several versions at the same time, which gives the user complete control over file creation

first come, first served (FCFS) scheduling algorithm that operates on a simple queue mechanism where the first process to request the CPU is the first to receive it (or the first element added to the queue is the first one to be removed); commonly used in resource scheduling and data buffering and also known as FIFO (first in, first out)

fragmentation problem where the memory blocks cannot be allocated to the processes due to their small size and the blocks remain unused

frame buffer portion of random access memory (RAM) containing a bitmap that drives a video display

full virtualization memory virtualization approach that allows multiple operating systems to run concurrently on a single physical machine, fully isolated from each other, by emulating hardware resources through a hypervisor

general-purpose register (GPR) extra register that is used for storing operands and pointers

graphical user interface (GUI) visual interface that allows users to interact with electronic devices through graphical icons and visual indicators

guest modification in the context of virtualization, altering the guest operating system or its configuration to improve compatibility, performance, or integration with the virtualization environment or hypervisor

guest operating system complete operating system inside a virtual machine

hardware abstraction layer (HAL) example of layering in modern OS that allows an OS to interact with a hardware device at a general or abstract level rather than going deep into a detailed hardware level; this improves readability

heap allocation dynamic storage management approach that allocates the data in a tree-based data structure

heap data tree-based data in process management

hypervisor software layer between machine hardware and the operating systems that run on it

i-number unique number given to an inode whenever an OS that uses inodes creates a new file, which, in effect, functions as the file's name

inconsistency situation that causes the system to produce errors or hardware failure

incremental backup backup image containing the pages that have been updated from the time of the previous backup

inode structure used by file system that contains metadata about a file or directory but not its name

inter-process communication (IPC) mechanism that enables processes to exchange data among different processes running on an operating system

interrupt signal to the processor from either software or hardware that indicate events that need immediate attention

isolation ensures that the multiple programs that are running concurrently on the same CPU and memory operate independently without interfering with each other's execution or data

layered OS architecture OS architecture where the OS is implemented as a set of layers where each layer exposes an enhanced virtual machine to the layer above

lock synchronization mechanism that is used to enforce mutual exclusion

mechanism activities that enforce policies and often depend on the hardware on which the operating system runs

memory allocation process of setting aside sections of memory in a program to be used to store variables and instances of structures and classes

memory deallocation process of freeing the space corresponding to finished processes when that space is needed by the rest of the system

memory multiplexing dividing the capacity of the communication channel into multiple logical channels

microkernel OS architecture where the functionality and capabilities are added to a minimal core OS as plug-ins

monolithic design OS architecture where the entire OS is working in kernel space

multitasking approach toward achieving concurrency that makes it possible for the OS to run multiple processes at the same time using time slicing

mutual exclusion program that prevents simultaneous access to a shared resource

non-privileged system program program that can run only in the user mode

non-uniform memory access (NUMA) computer memory architecture where memory access time varies depending on the memory's location relative to a processor

operating system (OS) core piece of software that typically manages the interconnection of hardware and software on a given computer

page fault when the CPU demands a page, and this page is not present in the main memory

page fetching process of bringing pages into memory

page manager layer in the layered OS architecture that implements virtual memories for each process

page replacement when one page in the DRAM is swapped to disk while the requested page is brought into DRAM

paging storage mechanism that uses a page form in retrieving process from secondary or virtual memory to main memory

pipe data communication method between two processes that uses a specific name and standard I/O operations, and thus allows for data transfer within a file system; sometimes called named pipe

policy controls how to use a mechanism in specific situations, that is, choose what activities need to be done

primary memory type of computer memory that is the initial point of access for a processor and serves as direct storage for the CPU

primary storage holds data that can be directly accessed by the CPU with minimum or no delay and does not survive a power failure

privileged instruction instruction provided by the CPU that can be executed only by the OS

privileged system program program that can run only in the system mode

process any program that is running on top of the OS

process control block (PCB) data structure used by the operating system to store information about a process, including its state, process ID, registers, scheduling information, memory management details, and I/O status

process ID (PID) unique identifier assigned by the operating system to each process running on a computer, used to track and manage process activities

process synchronization when an OS manages the sharing of a system's resources to avoid interference and errors

properties characteristics that are considered when designing an OS

protection general mechanism that is used throughout the OS for all resources that need to be protected, such as memory, processes, files, devices, CPU time, and network bandwidth

quota amount of space to store files based on the available memory space
ready state when a process is waiting for the CPU
recovery process of resolving or receiving treatment to solve OS faults or errors
reliability system's ability to reduce faults and ensure that the information in the system survives OS crashes and hardware failures
replication procedure that allows multiple copies of a file to exist in the network; this improves performance and availability
round-robin scheduling (RR) scheduling algorithm that is widely used in time-sharing systems and is designed to ensure fairness among processes by giving each process an equal share of the CPU
running state when a process is being executed by the CPU
secondary memory type of computer memory that is nonvolatile and thus used for long-term storage, housing the operating system, applications, and data that need to persist even when the power is off
secondary storage persistent memory that survives power failures most of the time such as spinning disks, SSDs, and USB drives
semaphore data type that an OS uses to control access to a resource
sharing multiple processes can use the same piece of data concurrently
shortest remaining processing time (SRPT) scheduling algorithm that prioritizes processes based on the shortest amount of time left to complete their execution
shortest time to completion first (STCF) scheduling algorithm that takes the best approach to minimize the waiting time, but it requires that the processor knows the processing time in advance; also called shortest job first (SJF)
stack allocation dynamic storage management approach that uses linear data structure that follows last in, first out (LIFO)
stack pointer (SP) register that indicates the location of the last item that was added to the stack
static data data that does not change within the program
synchronization way of coordinating multiple concurrent activities that are using shared state
system call appears when the program requests a service from the kernel
system interrupt manages the communication between the computer hardware and the system
thrashing when a computer's operating system becomes overwhelmed by the number of processes requesting memory
thread smallest unit of execution within a process, allowing parallel tasks to run in the same memory space; it enables efficient and independent execution of sequences of instructions
time slice short time frame that gets assigned to a process for CPU execution and facilitates multitasking
translation lookaside buffer (TLB) small memory cache that speeds up the computer's memory access by storing recent virtual-to-physical address translations; if the TLB has the address translation, it quickly retrieves data; if not, the computer must search more slowly through its memory
two-factor authentication form of authentication that involves two factors: the system calls or texts a user's phone for the traditional password during login, employing the cell phone as a key
uniform memory access (UMA) computer memory architecture where access time to any memory location is the same across all processors
virtual machine (VM) software that is created to run like a physical computer and that operates its own operating system and applications like a separate physical server
virtualization allows a system to run different types of applications used by multiple users at a time on the same computer
working set size (WSS) total amount of memory a process requires during a specific period of activity; it is measured as the set of pages or data blocks the process accesses

Summary

6.1 What Is an Operating System?

- An operating system (OS) is at the core of all of the connected hardware and software.

- Improving efficiency results in speeding up the implementation of applications from coding time and runtime standpoints. OSs have a large influence because of the abstractions/interfaces they implement.
- Operating systems provide both mechanism and policy. Mechanism refers to a set of activities that you can do. Policy is how to use the mechanism in specific situations.
- Virtualization in an operating system allows the system to run different applications that are handled by multiple users at a time on the same computer.
- Server virtualization places a software layer called a hypervisor (e.g., virtual machine monitor or VMM) between a machine (e.g., server) hardware and the operating systems that run on it.
- Using OS-level or server virtualization allows a server to run different types of operating systems at the same time on the same computer.
- The OS translates from the hardware interface to the application interface and provides each running program with its own process.
- A process consists of address space, one or more threads of control executing in that address space, and additional system state associated with it. The thread is a path of execution within a process and a process may contain multiple threads.
- The instruction set architecture (ISA) defines a set of instructions that can be used to write assembly language programs that use the CPU while abstracting the hardware details from the program.
- OS functions guarantee protection, isolation, and sharing of resources efficiently via resource allocation and communication.

6.2 Fundamental OS Concepts

- An OS manages computer resources (hardware) and provides services for computer programs (software).
- An OS is a complex system and executes many kinds of activities ranging from executing users' programs, to running background jobs or scripts, to completing system programs.
- Processing involves a program, a process, and a processor. An OS is responsible for managing processes, and different OSs approach process management in different ways.
- The address space is the set of addresses generated by programs as they reference instructions and data. The memory space holds the actual main memory locations that are directly addressable for processing.
- Computer memory consists of two main types: primary and secondary memory. An OS manages memory space through memory allocation and memory deallocation as well as by maintaining mappings from virtual addresses to physical and switching CPU context among addresses spaces.
- Device drivers are the routines that interact directly with specific device types and related hardware to indicate how to initialize the device, request I/O, and handle interrupts or errors.
- A device register is the interface a device presents to a programmer, whereas each I/O device appears in the physical address space of the machine as a few words.
- In an OS, it is important to have dual mode operations to ensure a high level of security and authority. The dual mode is responsible for separating the user from the kernel mode.
- Successful OS designs have had a variety of architectures, such as monolithic, layered, microkernels, and virtual machine monitors. As the design of an OS—and even its role—are still evolving, it is simply impossible today to pick one "correct" way to structure an OS.
- A monolithic OS design is an OS architecture where the entire OS is working in kernel space.
- A layered OS architecture consists of implementing the OS as a set of layers where each layer exposes an enhanced virtual machine to the layer above.
- Hardware abstraction layer (HAL) is an example of layering in modern OSs. It allows an OS to interact with a hardware device at a general or abstract level rather than going deep into a detailed hardware level, which improves readability.
- In a microkernel OS architecture, the functionality and capabilities are added to a minimal core OS.

6.3 Processes and Concurrency

- Concurrent processing is a computing model that improves performance when multiple processors are executing instructions simultaneously.

- A process consists of at least an address space, a CPU state, and a set of OS resources.
- The OS's process namespace particulars depend on the specific OS, but in general, the name of a process is called a PID (process ID), which is a set of unique numbers that identify processes.
- The OS maintains a data structure to keep track of a process state, which is called the process control block (PCB) or process descriptor.
- Concurrency refers to multiple activities and processes happening at the same time. An OS can achieve concurrent processing via the use of threads or one of three different processing environments: multiprogramming, multiprocessing, or distributed processing.
- Scheduling is the act of determining which process is in the ready state and should be moved to the running state when more resources are requested than can be granted immediately, and in which order the requests should be serviced.
- A good scheduling algorithm minimizes response time, efficiently utilizes resources, and implements fairness by distributing CPU cycles equitably. Four simple scheduling algorithms are FCFS, RR, STCF, and SRPT.
- Synchronization is a way of coordinating multiple concurrent activities that use a shared state.
- Allocation is a method that defines how data is stored in the memory by providing a set of requests for resources and identifying which processes should be given which resources to make the most efficient use of the resources. There are three main forms of allocation: contiguous allocation, linked allocation, and indexed allocation.

6.4 Memory Management

- The OS loads executable files into memory, allows several different processes to share memory, and provides facilities for processes to exceed the memory size after they have started running.
- Memory multiplexing is dividing the capacity of the communication channel into multiple logical channels.
- There are several concepts that are critical to memory multiplexing, namely, isolation, sharing, virtualization, and utilization.
- Time slicing is a time frame for each process to run in a preemptive multitasking CPU such that each process will be run every single time slice.
- Sharing means that multiple processes can share the same piece of data concurrently.
- Memory sharing improves the performance of the system because the data is not copied from one address space to another, so memory allocation is done only once.
- Virtualization is a technique that gives an application the impression that it has its own logical memory and that it is independent from the available physical memory.
- Fragmentation is a problem where the memory blocks cannot be allocated to the processes due to their small individual size and the distribution of sizes in the pool; there might be enough total free memory to satisfy the demand, but the available chunks cannot be allocated contiguously.
- Linkers combine many separate pieces of a program, reorganize storage allocation so that all the pieces can fit together, and touch up addresses so that the program can run under the new memory organization.
- There are two basic operations used in dynamic storage management to manage a memory or storage to satisfy various needs: allocate a block with a given number of bytes or free a previously allocated block.
- Virtual memory is a key component of the operating system for ensuring process isolation by guaranteeing that each process gets its own view of the memory.

6.5 File Systems

- A file system is responsible for defining file names, storing files on a storage device, and retrieving files from a storage device.
- File systems define operations on objects such as create, read, and write, and they may also provide higher-level services, such as accounting and quotas, incremental backup indexing or search, file versioning, and encryption.
- File systems are concerned with lower-level characteristics such as performance and failure resilience.

- The file system interface defines standard operations such the creation and deletion of files (or directories), manipulation of files and directories, copy, and lock.
- File systems are responsible for managing parts of the disk that are used (inodes) and parts of the disk that are not used (free blocks).
- A distributed file system (DFS) is a file system that is distributed on multiple file servers or multiple locations that support network-wide sharing of files and devices.
- A DFS provides an abstraction over physical disks that is akin to the abstraction that virtual memory provides over physical memory.

6.6 Reliability and Security

- We consider an OS to be reliable if it delivers service without errors or interruptions.
- Protection is a general mechanism used throughout the OS and for all resources needed to be protected such as memory, processes, files, devices, CPU time, and network bandwidth.
- There are three aspects to a protection mechanism: authentication, authorization, and access enforcement.
- The traditional way of authentication involves a password, which is a secret piece of information used to establish the identity of a user and should be relatively long and hard to guess. Another form of authentication is two-factor authentication, which involves two factors: the system calls or texts a user's phone for the traditional password during login, employing the cell phone as a key.
- The authorization determines the relationship between principals, operations, and objects by defining which principals can perform which operations on which objects.
- An access control list (ACL) is a list of rules that specifies which users are granted access to a specific object or resource.
- A capability list is a list of objects and operations for each user that defines the user rights and capabilities.
- To support access enforcement, one part of the OS must be responsible for enforcing access controls and protecting authentication and authorization information.
- There are many advantages to using logging: recovery is much faster; it eliminates inconsistencies; a log can be localized in one area of disk, which makes log writes faster; and it results in better performance. One of the disadvantages of logging is that synchronous disk write happens before every metadata operation.
- Virtual machines have become a fundamental component of cloud computing, as they allow cloud providers to offer scalable and flexible computing resources to users on a pay-as-you-go basis.



Review Questions

1. What is a privileged instruction that can only be executed by the kernel in Windows 10 or macOS operating systems?
   a. opening a text file
   b. modifying system clock settings
   c. printing a document
   d. creating a new user directory

2. You are building your own computer and have finished installing all hardware components. What should you install first?
   a. Microsoft Office
   b. Microsoft Windows OS
   c. external I/O device drivers
   d. antivirus software

3. What process or component allows a system to run different types of applications used by multiple users at a time on the same computer?
   a. virtualization

b. kernel
c. operating system
d. thread

4. How is efficiency defined with regard to operating systems?

5. What is virtualization as it relates to OSs?

6. Who sets policies in OSs?

7. What is the difference between user mode and kernel mode?

8. What component handles devices and provides buffering?
a. device driver
b. device register
c. device manager
d. I/O devices

9. How can a monolithic OS design be described?
a. an OS architecture where the entire OS is working in kernel space
b. OS architecture where the functionality and capabilities are added to a minimal core OS as plug-ins
c. an example of layering in modern operating systems
d. a computer memory design where memory access time varies depending on the memory's location relative to a processor

10. What type of memory access is described as computer memory architecture where access time to any memory location is the same across all processors?
a. cache-only memory architecture (COMA)
b. non-uniform memory access (NUMA)
c. uniform memory access (UMA)
d. random access memory (RAM)

11. What are the main components of any operating system?

12. What are the differences between thread and process?

13. What does the hardware abstraction layer (HAL) refer to?

14. What scheduling algorithm prioritizes processes based on the shortest amount of remaining execution time?
a. first come, first served (FCFS)
b. round-robin (RR)
c. shortest remaining processing time (SRPT)
d. priority scheduling

15. What is synchronization?
a. the way of coordinating multiple concurrent activities that are using a shared state
b. computing model that improves the performance when multiple processors execute instructions simultaneously
c. the memory that can be accessed by multiple processes and the processes that can communicate with each other without the middleman
d. the data communication method between two processes, using a specific name and standard I/O operations, allowing for data transfer within a file system

16. What is an example of static data?
a. a variable to keep track of the number of iterations in a loop in a program
b. the date and time in the operating system

c. the name of a file in a directory
d. a hardcoded country code in a program that is created with the final keyword

17. How are processes managed by the OS conceptually?

18. How are I/O devices managed by the OS conceptually?

19. Why is scheduling counted as an important operation in OSs?

20. What is the term for a technique where a process's memory is divided into various segments or sections, each representing different types of data or code?
a. time slicing
b. paging
c. isolation
d. segmentation

21. Stack allocation uses what data processing technique?
a. last in/last out
b. first in/first out
c. first in/last out
d. last in/first out

22. How does a linker work?

23. What is the difference between static and dynamic linking?

24. How does caching relate to virtual memory?

25. What component is responsible for defining file names, storing files to a storage device, and retrieving files from a storage device?
a. file system
b. file versioning
c. file
d. file path

26. What is a directory?
a. persistent memory that survives power failures most of the time, such as spinning disks, SSDs, and USB drives
b. a collection of related information that is stored on secondary/virtual storage and is the smallest storage unit from the user’s perspective
c. a system that allows a file to exist in several versions at the same time, which gives the user complete control over file creation
d. a set of files that contains all the required information about the files, such as attributes, location, and ownership, which is managed by the OS

27. What is a distributed file system?

28. What is an inode?

29. Define the file system interface.

30. What is the term for the operating system that is virtualized?
a. guest operating system
b. host operating system
c. default operating system
d. dual boot operating system

31. What is the term for checking to see if a user's credentials match the credentials in a database of

authorized users or in a data authentication server?

a. authorization
b. access enforcement
c. authentication
d. badge

Conceptual Questions

1. What is the difference between a policy and a mechanism? Please give examples to illustrate your explanation.
2. How are the compiler, OS, and CPU ISA coordinated? What are all the code modules, where do they exist, and how do they cooperate?
3. Give an example of an OS that uses a layered design.
4. What alternatives to monolithic OS design have been tried?
5. Explain in detail how caching relates to the use of virtual memory.
6. Explain how virtual memory became a key component of the operating system.
7. Explain in detail the file system's higher-level services.
8. Explain how the file versioning will help the user and the system.
9. Explain the difference between authentication, authorization, and access control. Are there any other types of security protections you would want an OS to provide when using software applications?

Practice Exercises

1. Search on the Web for "Windows system structure" and compare it with "UNIX/Linux system structure."
2. Draw a high-level diagram that illustrates the flow of control for an application of your choice that leverages an OS. Make sure that you identify the various components and layers as well as the users involved, if any.
3. Based on the operating system you are using, search on the Web for your operating system architecture.
4. Draw the architecture of your operating system.
5. Give an example of a scenario that requires synchronization.
6. Search on the Web for the most used allocation mechanism.
7. Explain how segments and pages are used to support virtual memory.
8. Search the Web for how to find the total number of inodes using your operating system.
9. Explain the relative merits of various recovery approaches. Start with the ones mentioned in the book and explore more on the Internet as needed.
10. Research various encryption algorithms and provide a summary of the results found.

Problem Set A

1. You have a Windows computer and need to test software that you developed in a Linux environment. How can you test your software with one machine?
2. Explain why we need to study OS architecture.
3. Explain how an OS decides how much physical memory to allocate to each process and decides when to remove a process from memory.

4. Explain why we need to study OS allocation methods.
5. Explain how an OS decides how much physical memory to allocate to each process and decides when to remove a process from memory.
6. Explain how the memory is divided.
7. Explain fragmentation.
8. Imagine you're tasked with creating a new file system that will only be utilized to store videos on YouTube. Describe the kind of access patterns you anticipate occurring most frequently in that specific file system.
9. Suppose you are asked by a company to select a new authentication method. If the company is not using multi-factor authentication, how could you argue the need for this method?
10. How can you use badging in the authentication process?

Problem Set B

1. Write a simple piece of code on an OS of your choice that calls a function in a programming language of your choice and explain how your program uses the stack.
2. Write a simple piece of code on an OS of your choice using a programming language of your choice that makes use of the heap.
3. Outline the fundamental differences between two of the most popular mainstream operating systems (e.g., Max OS X, Windows 10, Linux) from an OS architecture and OS components standpoint. Do some research on the Internet to obtain architectural diagram and component descriptions from a trustworthy source and show all your work.
4. Most of the OSs now are moving to multiprocessing. Explain how multiprocessing reduces the latency and increases the overall performance.
5. Discuss the benefits of memory sharing for the user and OS perspective.
6. How and why might a file system created specifically for storing movies on YouTube's website differ from the businesses outlined in this book?
7. What are the advantages and disadvantages of using logging in your OS?

Thought Provokers

1. Consider our start-up company that is 100% committed to leveraging innovative technologies as a business growth facilitator. Describe how it can best use an operating system to create products or services that can generate business (e.g., mobile health application that detects elevated levels of stress and suggests playing games, listening to songs, or watching videos to reduce stress). Give precise examples and explain how the start-up would be able to scale the resulting business (i.e., keep sustaining the cost of doing business while increasing its number of customers).
2. Consider our start-up company that is 100% committed to leveraging innovative technologies as a business growth facilitator. Describe how it could leverage a new operating system design to support an innovative application that leverages the use of various sensors located at the edge of the network. Give some precise examples and explain how the start-up would be able to scale this approach.
3. Consider our start-up company that is 100% committed to leveraging innovative technologies as a business growth facilitator. Describe how it could leverage IPC and concurrency control to support an application that makes it possible to collect data at the edge of the network from a large variety of sensors and enable processing of that data in real time. Give some precise examples and explain how the start-up would be able to scale this approach in the context of an epidemic such as COVID-19.

4. Consider our start-up company that is 100% committed to leveraging innovative technologies as a business growth facilitator. Describe how it could leverage memory management to support very memory-demanding applications, making it possible to perform all computations on data in memory. Are there some examples of similar technologies that already exist today? Give some precise examples and explain how the start-up would be able to scale this approach.
5. Consider our start-up company that is 100% committed to leveraging innovative technologies as a business growth facilitator. Describe how it could leverage a distributed file system to make it possible to gather mission-critical data in real time from various users located at the edge of the network. Give some precise examples and explain how the start-up would be able to scale this approach.
6. Consider our startup company that is 100% committed to leveraging innovative technologies as a business growth facilitator. Describe how it could leverage OS technology obsolescence. Give some precise examples and explain how the startup would be able to scale this approach.

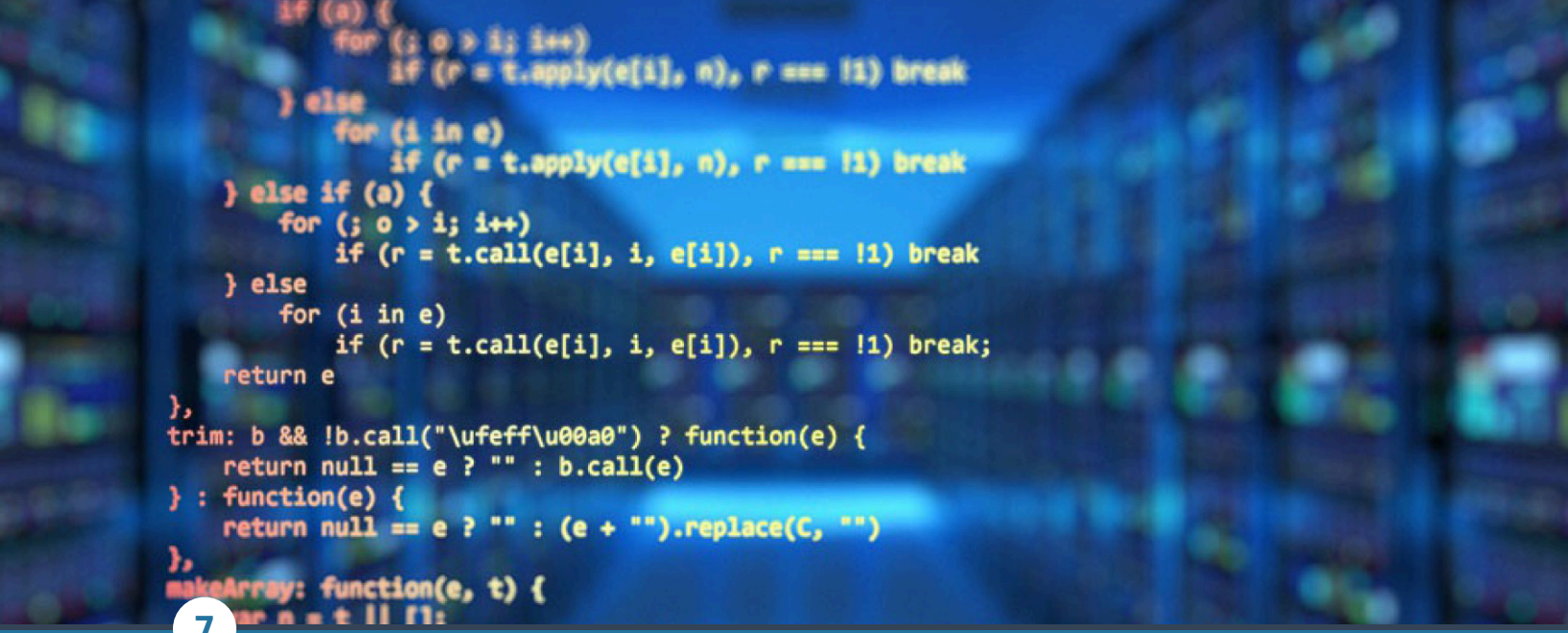
Labs

1. Install Oracle Virtual Box on your laptop and deploy an image of an operating system in Virtual Box that makes it possible to use a different operating system on your laptop.
2. Create a Linux virtual machine on a cloud of your choice and install X2Go to access the virtual machine from your laptop. Explain how the OS and windowing system make all of this possible.
3. Write a program in a programming language of your choice and deploy it using the OS of your choice. Use a GNU compiler tool to compile and link your code and demonstrate how your program makes used of memory management (e.g., dynamic memory allocation).
4. Create a file system on a cloud of your choice and mount it as a drive on your computer. Perform some experiments with various applications of your choice to determine if the performance is acceptable. Experiment with the SaaS functionality provided on various big clouds as you work on this lab.

   Examples with SaaS:

   - Office suites: Use SaaS offerings like Google Workspace or Microsoft 365 to create and edit documents. Observe the responsiveness of these services.
   - Development tools: Experiment with cloud-based IDEs like AWS Cloud9 or GitHub Codespaces to develop and run code. Pay attention to the execution speed and any latency in the development process.
   - Database management: Work with a cloud-based database service like Amazon RDS. Perform queries and updates to test performance.
   - Analytics: Utilize services like AWS QuickSight or Looker Studio to perform data analysis tasks. Evaluate the speed of data processing and visualization rendering.

5. Research the recovery features that are available on your computer's OS and document what you would need to do in case of a system crash. Create a recovery disk as needed so that you are prepared for the worst.



7

High-Level Programming Languages

Figure 7.1 High-level languages make it easier for programmers to solve problems and design software at a level above the computer's architecture. (credit: modification of "Computer science and engineering" by "BVECJordan"/Wikimedia Commons, CC0)

Chapter Outline

7.1 Programming Language Foundations
7.2 Programming Language Constructs
7.3 Alternative Programming Models
7.4 Programming Language Implementation

Introduction

Programming is the science behind writing programs, which makes it possible to implement algorithms that leverage mathematical and/or scientific knowledge. Programming is also an art that requires creativity and employs imagination. High-level languages (HLLs) give programmers the ability to produce linguistic realizations of algorithms using a notational system that facilitates human-computer interaction.

TechWorks is an example of a company focused on new technology; for it to leverage technology and fulfill its stated mission, it regularly makes decisions on which HLLs to use, what exactly to use them for, and many other HLL suitability factors such as the following:

- Types of application
- Target platforms
- Maintainability
- Scalability
- Performance
- Security

TechWorks will need to choose from a pool of programming languages that excel in different areas. For example, JavaScript is a versatile language that applies to the interactive elements that users will see and interact with when using TechWorks's web interfaces. JavaScript is a natural choice for this task due to its ability to create dynamic and engaging user experiences. For server-side operations, the choice of programming languages must strike a balance between latest technology needs and experienced

programmers' preferences. JavaScript with frameworks like Node offers a cutting-edge approach, while established languages like PHP or ASP.NET boast a larger pool of seasoned programmers. TechWorks will need to use the Structured Query Language (SQL) to communicate with database systems used to support its applications. Python is a powerful tool language for data analysis and manipulation. Its extensive libraries and clear syntax make it well-suited to extract insights from TechWorks's collected data. In a nutshell, TechWorks will need to strategically combine the use of various programming languages to create robust and user-friendly applications.

7.1 Programming Language Foundations

Learning Objectives

By the end of this section, you will be able to:

- Describe what HLLs are
- Summarize choosing appropriate HLLs
- Outline the history of HLLs
- Describe the implementation of HLLs

A high-level programming language is designed to be easy for humans to read, write, and understand. It abstracts away most of the complexities of the underlying hardware and machine code, allowing programmers to focus on solving problems and designing software without needing to manage the low-level details of the computer's architecture.

What Are HLLs?

High-level programming languages give humans the ability to direct computers to perform tasks and applications. There are many HLLs to choose from. Java is a popular choice for its ability to run on various operating systems (i.e., Windows, macOS, Linux) and mobile platforms (Android). This is called cross-platform compatibility. For development specifically targeting Windows systems, C# is another strong option. Additionally, to create the visual elements of a website, programmers can utilize HTML and CSS. HTML provides the structure and content of the web page, while CSS controls how web pages are styled and presented. Over time, many HLLs have evolved into a mature set of tools that are used to create modern applications (Figure 7.2).

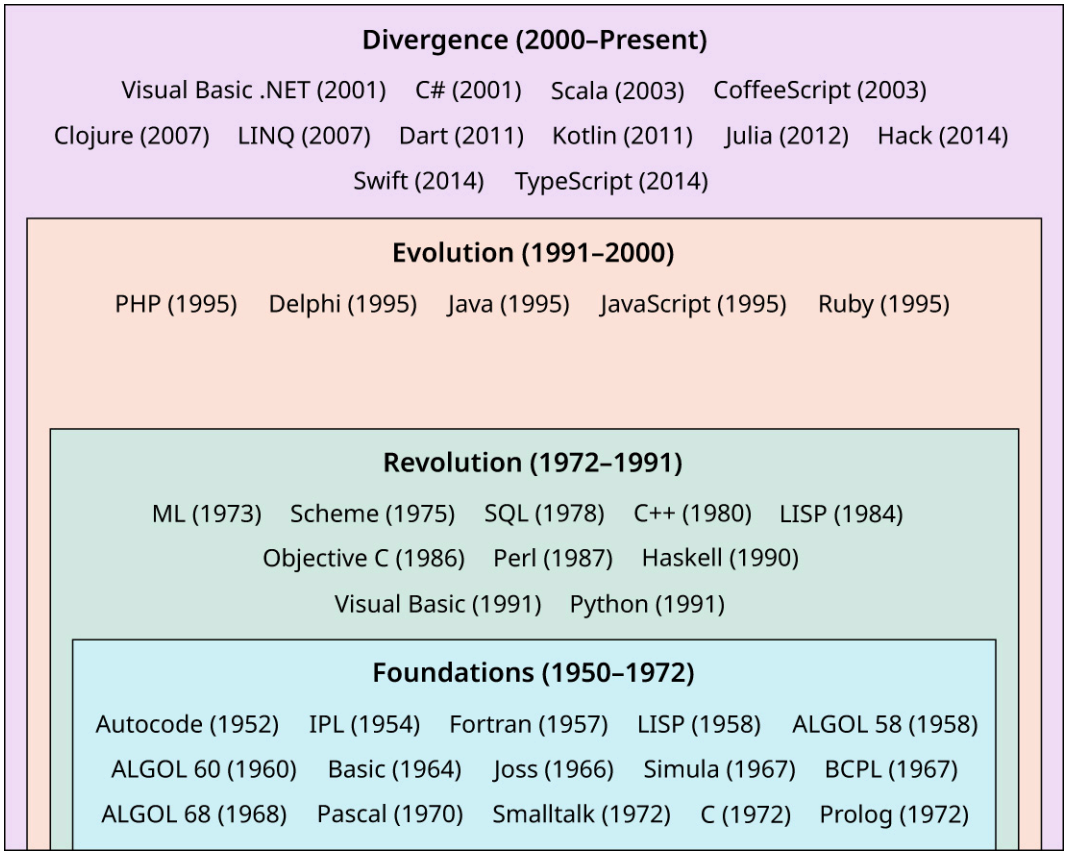


Figure 7.2 High-level programming languages have advanced from the foundational languages in the middle of the 20th century to more than 2500 HLLs that exist today. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

LINK TO LEARNING

There is a possibility that the widespread use of a variety of HLLs to develop networked and mobile applications at a global scale creates potential cybersecurity issues. Some HLLs are considered more secure than others. Read this article on the most secure programming languages (https://openstax.org/r/76ProgLanguages) for further information about the security—or lack thereof—of these languages.

Learning Motivations

Studying the fundamental concepts provided by various HLLs is necessary to choose them correctly, employ them effectively, and program efficiently. From a user point of view, examining HLL concepts helps the user get better at thinking and expressing algorithms. From an implementor's point of view, understanding HLL concepts helps programmers abstract away from (virtual) machines and become better at specifying what they want the hardware to do without getting down into the bits. In the end, studying HLL concepts helps programmers make better use of whatever HLL they use.

Implementing Abstraction

One way to relate to **abstraction** is as a way of thinking and expressing algorithms to indicate what the programmer wants the hardware to do. For example, the following statement represents one form of abstraction in the Java programming language:

```
System.out.println("Hello world!");
```

It tells the computer's operating system at a high-level of abstraction to output a string of characters, which

practically consists of moving the pixels that form characters one by one to a hardware device.

Implementing a high level of tasks would be impossible without abstraction. For example, you would not want to program an invoicing application in 1s and 0s (machine language); abstraction allows a programmer to build it in an English-like syntax.

Abstraction may be taken to much higher levels. It is one of three central principles (along with encapsulation and inheritance) in such object-oriented HLLs as C++, Java, C#, and Python. Various programming paradigms were introduced in Chapter 4 Linguistic Realization of Algorithms: Low-Level Programming Languages, particularly the mechanisms of object-oriented programming (OOP) and its standards, perspectives, or sets of ideas that may be used to describe the structure and methodologies of an HLL. Object-oriented HLLs help organize software design around data, or objects, rather than functions and logic, as we will discuss in Alternative Programming Models.

Choosing Appropriate HLLs

Studying and understanding HLL concepts allows us to make most efficient use of them by becoming familiar with various criteria that may be used to evaluate them, which helps us choose the most appropriate language for a project. Some of these criteria are listed in Table 7.1, which also shows how they are related to the characteristics of an HLL. These criteria are as follows:

- **readability**: measures how easily an HLL can be read and understood
- **writability**: measures how easily an HLL can be used to create and modify programs
- reliability: measures conformance to specifications

There are many other criteria including scalability, cost, flexibility, efficiency, portability, and maintainability. These can be used to identify which HLL is best suited for a given task.

Characteristic	Readability	Writability	Reliability
Simplicity: a manageable set of features and constructs	•	•	•
Orthogonality: a relatively small set of primitive constructs can be combined in a relatively small number of ways	•	•	•
Data types: adequate predefined constructs to hold data	•	•	•
Syntax design: form and meaning via self-descriptive constructs and meaningful keywords	•	•	•
Supports abstraction: hides all but the relevant data about an object in order to reduce complexity and increase efficiency		•	•
Expressivity: relatively convenient ways of specifying operations		•	•
Type checking: built-in testing for type mismatches			•

Table 7.1 Criteria for Measuring Characteristics of HLLs

Characteristic	Readability	Writability	Reliability
Exception handling: support for catching run-time errors and specifying corrective measures			•
Restricted aliasing: presence of two or more distinct referencing methods for the same memory location			•

Table 7.1 Criteria for Measuring Characteristics of HLLs

Learning New HLLs

Studying the concepts of HLLs makes it easier to learn new HLLs since most have similarities in syntax, structure, and semantics. There are also several best practices that apply to different HLLs. A **best practice** is the most accepted style and structure of code that can be used to ensure proper software development, which makes it possible to learn new languages easily once a programmer has mastered a given one. The HLLs that are most used as teaching languages today are Java, C++, and Python. Java and C++ are languages that take a significant amount of study to master, while Python is considered a much simpler language to learn.

LINK TO LEARNING

HTML and CSS are markup languages and not exactly programming languages like Java or Python. The official HTML and CSS standards (https://openstax.org/r/76HTMLCSSStds) are available at World Wide Web Consortium (W3C).

Best Use of HLLs

Programmers have to figure out how HLLs support certain features. For example, a **variable** gives a name to a memory location that is used in any HLL to hold a value. However, different languages use variables differently. Java is a **strongly typed** language, meaning that a variable may only contain a value of one of the language's defined data types for its entire existence. Therefore, a variable that is a number cannot become a string of text. JavaScript is **weakly typed** so a variable may at different times hold values of any of the language data types. It may be storing a number, then later, the same variable may store a string of characters.

Another example is the use of pointers in C and C++. As visible in Figure 7.3, the pointer is the variable that holds actual computer memory addresses, but they do not exist in Java. However, understanding how C handles memory makes it easier to understand how data is passed from one place to another in Java or C#.

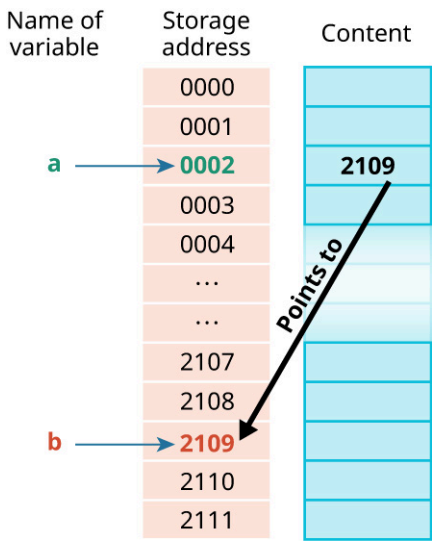


Figure 7.3 A C pointer variable "a" holds the memory address of the "b" variable. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Languages Are Purpose Driven

Let's again contrast C/C++ and JavaScript. Much of the syntax and grammar are the same, as are the flow of control structures of the language.

LINK TO LEARNING

You can review the current C/C++ standard (https://openstax.org/r/76C++Standard) to dig deeper into a syntax comparison.

So why pick one over the other? The answer is that languages are designed to fulfill certain purposes. C/C++ is a general-purpose programming language. As such, it is powerful for applications that include both systems programming and object-oriented graphical user interface (GUI) programming, JavaScript is intended for web programming.

Although programming languages differ in syntax, they all have libraries or packages that are installed as part of the language development environment. These libraries expose various functions via an application programming interface (API). These functions support the tasks for which the language is purposed while not requiring additional coding. The following illustrates the use of a C++ library function that prints a string to the screen:

```
cout << "Hello world!";
```

JavaScript's API contains a comprehensive set of features that enable the manipulation and dynamic behavior of web pages. A JavaScript API function that prints to the web page:

```
document.write("Hello world!");
```

TECHNOLOGY IN EVERYDAY LIFE

Using Map APIs to Navigate Your World

APIs are toolkits for programmers. They provide building blocks that make it easier to create software

applications. API functionality can also help people with everyday life situations. For example, Google provides a JavaScript Maps API for customizing map content to display on web pages. Imagine you are planning a road trip. You can then use the Maps API to create a customer map with your planned route, stops, and estimated travel times. Now imagine you own a coffee shop. The Maps API can then help you display your location and operating hours on a map, making it easier for customers to find you. Want to learn more about building apps with APIs? Check out the Maps JavaScript API (https://openstax.org/r/76MapJavaAPI) resource.

Google the APIs for an HLL that we have mentioned and find some functionality that applies to everyday life. Think of an app you use every day. How do you think it might use APIs? Provide a couple of scenarios to explain your choice.

History of HLLs

The evolution of HLLs began so that programmers could write programs in a familiar notation rather than using numbers (machine languages) or mnemonics (assembly languages). While there may be similarities in syntax among them, there are distinct purposes for which their development occurred. There is a much larger variety of HLLs than the ones mentioned in this section, but we will be looking at a few up close.

Fortran

In the early 1950s, IBM created one of the first HLL compilers for the Fortran language, which is one of the single biggest advances in computing. While Fortran was mostly used in mathematics and science, it could be easily read. Fortran makes it possible for programmers to comment their code by starting program lines with "!". It uses conditional statements with goto statements to branch out to different parts of the code. It also uses "do...end do" iterative statements. It was also the first HLL to use a compiler, computer software that converts source code from one language to another.

```
! Compute the average
Average = Sum / List_Len
! Count the values that are greater than the average
Do Counter = 1, List_Len
   If (Int_List(Counter) > Average) Then
      Result = Result + 1
   End If
End Do
```

COBOL

Common Business-Oriented Language (COBOL) was developed to be a high-level language for business that was standardized by the American National Standards Institute (ANSI) group in 1968. COBOL represents a distinct milestone in the evolution of computer science because of the ways in which it differed from Fortran.

The following code snippet illustrates reading an inventory record and computing the available stock:

```
100-PRODUCE-REORDER-LINE.
   PERFORM 110-READ-INVENTORY-RECORD.
   IF CARD-EOF-SWITCH IS NOT EQUAL TO "Y"
      PERFORM 120-CALCULATE AVAILABLE STOCK
      IF AVAILABLE STOCK IS LESS THAN BAL-REORDER-POINT
         PERFORM 130-PRINT-REORDER-LINE
```

```
110- READ-INVENTORY-RECORD.
   READ BAL-FWD-FILE RECORD
      AT END
         MOVE "Y" TO CARD-EOF-SWITCH.

. . . .
```

We can see that COBOL has a very different type of syntax than Fortran. It is purposed differently as it is very aligned to business applications and the programming of specific business activities that make up business processes.

BASIC

Beginner's All-Purpose Symbolic Instruction Code (BASIC) was developed in 1971. It is a programming language that has enjoyed widespread use. A variation of BASIC referred to as Visual Basic (VB) was the language responsible for much of the development work performed on the new generations of personal computers as it was easy to learn and read. Today it has evolved into Visual Basic .NET.

The following code snippet illustrates the same computation and comparison of an average we did in Fortran but this time in BASIC:

```
REM Compute the average
average = sum / listlen
REM Count the values that are greater than the average
FOR counter = 1 to listlen
   IF intlist(Counter) > average
      THEN result = result + 1
   End If
NEXT
```

Note that the BASIC syntax has its roots in Fortran but is more efficient.

Pascal and C

The programming languages introduced so far follow the **imperative language** paradigm that emphasize a "tell the computer what to do" approach. Pascal and C distinguish themselves by being both procedural and imperative languages, and they were invented at approximately the same time. A **procedural language** allows programmers to group statements into blocks of code within the scope of which variables may be defined and manipulated independently from the rest of a program. These blocks can be named, in which case it allows programmers to create functions or procedures that can be called from other parts of a program. Similar to other imperative languages, both Pascal and C also focus on evaluating expressions and storing results in variables (e.g., $a = 10$; $b = 5$; $c = a + b$).

The following Pascal code snippet illustrates the same computation and comparison of the average computed previously in BASIC:

```
{ Compute the average }
average := sum / listlen;
{ Count the values that are greater than the average }
for counter := 1 to listlen do
   if (intlist[counter] > average) then
      result := result + 1;
```

The introduction of procedures in Pascal improved programs' readability by allowing programmers to write

more modular code. Pascal became the preferred teaching language during the 1970s and early 1980s.

Pascal was overshadowed in commercial applications by C, which came into existence after the initial work on the UNIX operating system was completed in the late 1960s. That first OS version was written in assembly language, yet in the early 1970s, C became a better alternative. At the time, it was the perfect language for creating operating systems and was a huge commercial success.

The following C code snippet illustrates the same computation and comparison of the same computed average:

```
/* Compute the average */
average = sum / listlen;
/* Count the values that are greater than the average */
for (counter = 0; counter < listlen; counter++)
    if (intlist[counter] > average) result++;
```

The introduction of functions in the C language improved programs' readability and writability by allowing programmers to write more modular code. The C language runtime was also more efficient. C became the preferred language for commercial applications during the 1970s and early 1980s.

C++ and Objective C

By the mid-1980s, businesses started focusing on user experience (UX), the overall experience of a person using a computer application, especially in terms of how easy or pleasing it is to use, and the **user interface (UI)**, the point at which human users interact with a computer, website, or application. Windows-based UIs were adopted as new paradigms, which drove the creation of standards, perspectives, and sets of ideas that should be used to describe the structure and methodologies of an HLL. This resulted in the adoption of the OOP paradigm and the creation of OOP languages. The shift to OOP allowed software to focus on data and objects.

The C++ programming language extended the middle-level language features of C with OOP features that facilitated the expression of real-world requirements in programs, including in particular the support of graphical user interfaces (GUIs). Microsoft adopted C++ as the programming language for its Windows systems.

The syntax of the code in C++ for basic computation is exactly the same as the C code shown previously. There are major syntactical additions in C++ to support OOP. It became the preferred language during the mid-1980s and early 1990s and for the programming of GUIs. The code in Figure 7.4 illustrates the basics of Windows GUI programming using C++ and the Win32 API to create a simple "Hello World!" application with a graphical user interface (GUI).

```cpp
#include <windows.h>

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE, PWSTR pCmdLine, int nCmdShow
{
    const wchar_t CLASS_NAME[] = L"HelloWorldWindowClass";

    WNDCLASS wc = {};
    wc.lpfnWndProc = WindowProc;
    wc.hInstance = hInstance;
    wc.lpszClassName = CLASS_NAME;

    RegisterClass(&wc);

    HWND hwnd = CreateWindowEx(
        0,
        CLASS_NAME,
        L"Hello, World!",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
        NULL,
        NULL,
        hInstance,
        NULL
    );

    if (hwnd == NULL)
    {
        return 0;
    }
    ShowWindow(hwnd, nCmdShow);

    MSG msg = {};
    while (GetMessage(&msg, NULL, 0, 0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return 0;
}

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    switch (uMsg)
    {
        case WM_DESTROY:
            PostQuitMessage(0);
            return 0;
        case WM_PAINT:
        {
            PAINTSTRUCT ps;
            HDC hdc = BeginPaint(hwnd, &ps);
            FillRect(hdc, &ps.rcPaint, (HBRUSH)(COLOR_WINDOW + 1));
            EndPaint(hwnd, &ps);
        }
        return 0;
    }

    return DefWindowProc(hwnd, uMsg, wParam, lParam);
}
```

Figure 7.4 This GUI shows the “Hello World!” application using C++ and the Win32 API. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Objective-C is another hybrid language from the 1980s that makes use of imperative procedural, and object-oriented features. Apple adopted it to power their operating systems development efforts and became popular for a while as Apple’s only accepted language for programming apps for iPhones, later replaced by Swift.

Java

Consumer electrical devices, such as microwaves and interactive TVs, called for the invention of another OOP language: Java. Java evolved into a platform-independent, general-purpose language for computational devices, usable for everything from PCs and MACs to Androids and Samsung refrigerators. To this day, it remains one of the most widely used languages for teaching OOP and is one of the most popular HLLs in the world.

As shown in Figure 7.5, the syntax of the code in Java for typical computations is similar to that of the C code shown earlier. This is the case for most modern HLLs introduced since 1990.

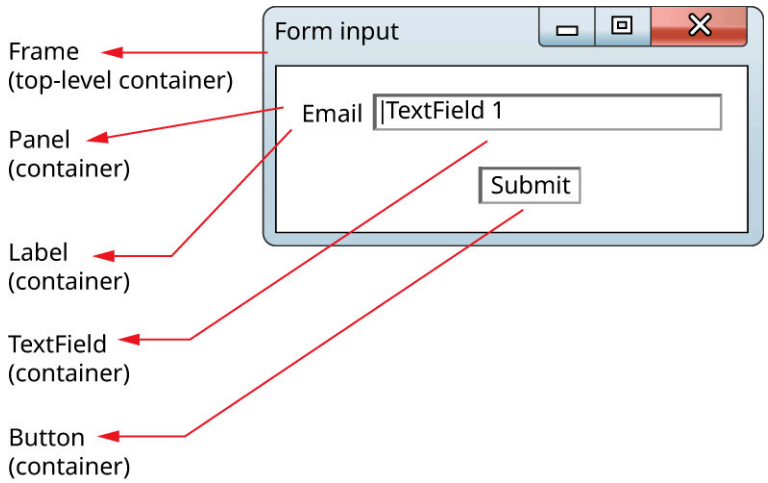


Figure 7.5 These are some of the elements available from the Java GUI API. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

LINK TO LEARNING

Refer to the ECMAScript standard (https://openstax.org/r/76JavaScript) if you would like to dig deeper into a syntax comparison that includes JavaScript.

Scripting Languages

A **scripting language** is characterized by placing a list of code statements into a file, referred to as a script. Script statements are typically interpreted line by line rather than being compiled as complete units to produce executable programs. There are advantages and disadvantages to interpreting and compiling methods, which we will cover in Implementation Approaches.

The most popular scripting languages employ C-like syntax, but they are purposed for different applications. For example, the JavaScript and PHP scripting languages are purposed for programming web applications.

CONCEPTS IN PRACTICE

HLLs and Web Applications

Most HLLs that are used to develop web applications are scripting languages. These include JavaScript, PHP, ASP.NET, and Python. JavaScript is nearly universal for front-end (browser, client-side) applications. Web servers such as the Apache web server and Microsoft's Internet Information Services (IIS) server support a Common Gateway Interface (CGI) that allows the invocation of server-side programs including scripts.

Some of these scripting languages are now bolstered by web frameworks that are designed to support the development of applications in the particular languages. For example, the most popular framework for web applications today in JavaScript is React. The most popular one for Python is Django. A guide to web frameworks (https://openstax.org/r/76WebFrameworks) may be found at the Statista website.

C#

In 2000, Microsoft announced C# would be its flagship language. It also has the same fundamental syntax as C++ and Java. However, it is purposed to support Windows applications by closely tying in with Microsoft's net framework, but it can also be used on Linux and macOS. Net C# is a multi-language, component-based software development tool designed to play nicely with all of the .Net languages including C#, Visual Basic.Net, and Managed C++.

INDUSTRY SPOTLIGHT

HLLs in Industry

HLLs are important in every industry. One example is Python. Part of its purpose is to support data analytics to process complex data, a major focus of many industries today. It does this with built-in analytics tools in its API which can process raw data and produce information and graphics that can be used to make business decisions. For example, a company interested in generating a graphical representation of its products' sales across various regions during the past year may use Python data analytics and plotting libraries.

Can you elaborate on how useful it will be to know about HLLs in an industry of your choice (e.g., finance, gaming, travel)? *Hint*: Think about industries which tend to specialize in specific areas.

Logic-Based Languages

Logic-based programming languages are those which incorporate a syntax to represent facts and rules about approaches to problems. They have been used to support rule-based approaches as part of the development of artificial intelligence (AI), the simulation of human intelligence by machines such as computers.

The most common language for logic-based programming is Prolog. It is used for both AI and linguistics programming. It is actually an older HLL first developed in 1972 and has stayed with us, receiving extensive updating as AI developed. Figure 7.6 illustrates on the left side how facts and rules can be specified in Prolog. The query window on the right side illustrates how the Prolog fact database can be queried to leverage available rules.

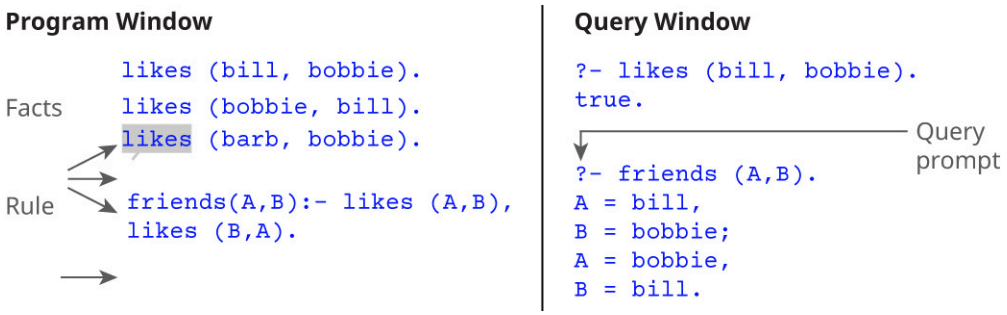


Figure 7.6 This sample program and query show the details of how to use Prolog. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The Implementation of HLLs

All computer languages can be grouped into particular categories. These are based upon support for certain programming paradigms (standards, perspectives, or sets of ideas that may be used to describe the structure and methodologies of an HLL). These paradigms include imperative/procedural, logical, functional, object-oriented, **event-driven** (the behavior of programs is controlled by actions (events) which are listened for and then acted upon or handled), and parallel programming (dividing a program into concurrent processes). Our modern HLLs are almost always hybrid combinations of these. We will learn about this in more detail in the following subsections of this chapter.

Imperative/Procedural Programming

As we have discussed, imperative HLLs take the "tell the computer what to do" approach. This approach is different from that of declarative HLLs that tell a program to obtain information without prescribing how the program should go about doing it. Declarative languages are used to interact with systems that are programmed to figure out these details. An example of a declarative language is **Structured Query Language (SQL)**, which is used to specify a query that a database system can process to store or retrieve data. Imperative languages typically focus on evaluating expressions and storing results in variables. There are other shared features of these languages, such as iteration (looping or repetition).

Procedural languages extend the imperative paradigm. They make use of procedure calls to change the flow of control. A procedure (function) can be called from anywhere in a program to have it perform a particular job. Some of the languages that support this paradigm are Fortran, COBOL, Pascal, Visual Basic, Ada, C, C++, and C#. Scripting languages, including Python, JavaScript, and PHP, may also be of this type.

Event-Driven Programming

Most imperative languages also embrace event-driven programming. This paradigm allows the generation of events (for example, as a user clicks on a button). In general, computer operating systems constantly process events of various types that result from interaction with users or are generated by application programs or computer hardware. In event-driven programming, a program is told to listen for selected events, such as the single-click on a particular object (e.g., a button on a UI). The programmer establishes an event handler to deal

with the event whenever it is triggered.

Parallel Programming

This paradigm refers to the computer's ability to process multiple tasks at the same time, which is especially useful in modern multicore systems. However, a program may not be allowed to execute across multiple cores without proper synchronization. For example, the program in one core may need a result or data item that is being produced by the program in another core. Therefore, effective parallel programming must have the tools by which to synchronize processes. The Ada programming language introduced built-in support for concurrent programming using tasks and protected objects. Ada tasks are defined with the task keyword and have their own declarations and executable parts. While tasks help structure programs in concurrent flows, protected objects safeguard shared data, ensuring that only one task can access them at a time to avoid race conditions and deadlocks. Other examples of HLLs that support parallel programming include C++, Java, and C.

Implementation Approaches

As we have learned, language implementations are commonly differentiated into those based on compilation and those based on interpretation. Many modern languages make use of a hybrid execution style.

Pure Compilation

In pure compilation, programs are translated directly into machine language. The compiler takes the entire high-level source code program and produces an equivalent object code. The compiler is not used in the actual execution of the program; the object program is launched by the operating system and executes on the underlying machine from start to finish as shown in Figure 7.7.

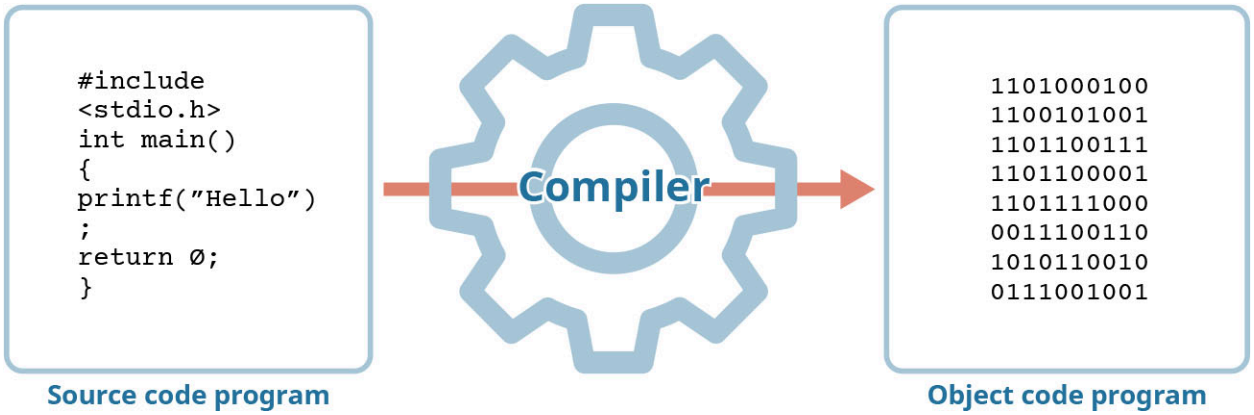


Figure 7.7 A sample C++ source program is compiled into machine language to run on a particular platform. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The advantages of pure compilation are better performance and better code analysis to detect source code typing errors. The disadvantage is that the compiled program is platform dependent and must be recompiled for other target machines. Examples of purely compiled languages include C and C++.

Pure Interpretation

In pure interpretation, programs are translated by another program known as an interpreter. The interpreter executes the program line by line. Because the interpreter executes the program, it is not platform dependent and is designed to execute code for the platform on which it resides. The disadvantages of interpretation are the slower execution speed due to having to both translate and run each line. Examples of purely interpreted languages are JavaScript, PHP, and Python.

Hybrid Implementation

Some language implementation systems are a mix of compilers and interpreters which is known as **hybrid implementation**, a method of language translation which involves the use of both a compiler and an

interpreter. The compiler first translates the HLL programs to an **intermediate language**, a language that is generated from programming source code, but that the CPU cannot typically execute directly. Some hybrid implementations allow easy interpretation during execution using **just-in-time (JIT) translation**, in which intermediate language is translated and executed exactly when needed. This method is much faster than pure interpretation.

Java is a good example of a hybrid implementation system, purposed to give the language platform independence. As illustrated in Figure 7.8, the compiler translates the HLL source code to intermediate **bytecode**, object code produced by Java compilation which is then interpreted by the **Java virtual machine (JVM)**, the Java interpreter that translates bytecode into executable code. This enables the same source to be used on all platforms for which a JVM has been constructed.

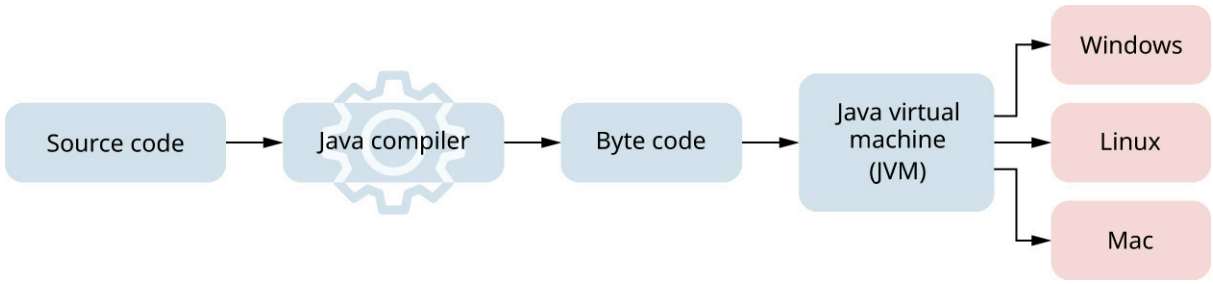


Figure 7.8 A Java source program is compiled into an intermediate language then interpreted to produce and execute object code to run on a particular platform. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

C#, and the other .NET languages, are implemented with a different JIT system. The compiler produces managed code into which the .NET languages are translated. The runtime environment is known as the **Common Language Runtime (CLR)**, which takes managed code and provides a JIT execution that allows all the languages to play nicely with each other. This hybrid implementation is purposed for cross-language in addition to cross-platform compatibility.

THINK IT THROUGH

HLLs and TechWorks

Come up with one application for TechWorks that will implement a particular part of the business (e.g., finance, sales, advertising). Briefly state the application and its objective. For this application, what will you choose as an implementation HLL—compiled, interpreted, or hybrid? Explain why.

Programming Environments

Programming environments include a collection of tools used in software development. Depending on which operating system and HLL you are using, there are many options for a software development environment. This bundle of tools targeted for a specific HLL that helps with source code editing, compilation and/or interpretation, debugging, styling, and other useful programming tasks.

LINK TO LEARNING

This article provides an interesting guide to current HLLs (https://openstax.org/r/76HLLs) and includes pros, cons, usages, average salaries, and other useful data.

7.2 Programming Language Constructs

Learning Objectives

By the end of this section, you will be able to:

- Discuss and compare HLL data types
- Demonstrate the use of variables
- Examine HLL expressions and statements
- Describe the implementation of flow of control in HLLs
- Introduce the concept of functions
- Classify well-structured programs
- Explain the concept of exception handling
- Summarize files and input/output

HLLs exist to communicate to a computer the logical steps for approaching a given task or application, and many HLLs act the same. Because of this, once you have mastered a modern HLL, it becomes easier to learn additional languages since you now know the correct questions to ask. For example, a starting point might be to find out how to obtain a simple program output which allows you to see how to run a program and test the concepts we are about to learn.

In this section we will describe the structural concepts of HLLs to give us the tools with which to compare them and learn them in a consistent way. A good starting point to examine programming language constructs is to demonstrate the fundamental building blocks of HLLs. These include the data types that languages can legally manipulate, how they store such data, how they structure the expressions and statements by which they communicate, and the control of the programming flow.

HLL Data Types

The data types of a language form the legal set of the kinds of data which an HLL may manipulate. These data types may be very simple, or they may be more complex. The simplest data type of a language is a **primitive data type** (also, basic data type), for example, integers and char in the C programming language. Data corresponding to variables of these types can usually be represented and manipulated directly using the machine hardware both in memory and via registers.

However, languages usually contain complex data types as well. A **complex data type** consists of multiple primitive types that are used as their building blocks. An example of this is the **string** data type which represents a sequence of characters. In the C programming language, character strings are complex data types represented using arrays of characters. In JavaScript, string is a primitive data type. Figure 7.9 relays the various data types.

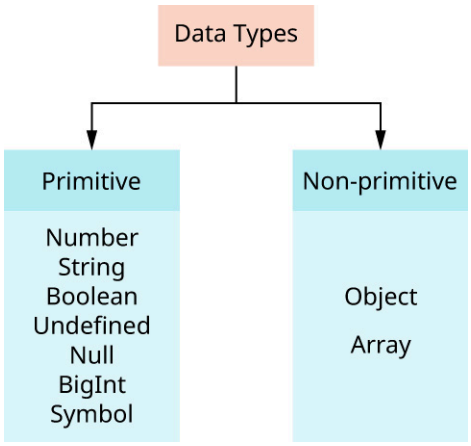


Figure 7.9 JavaScript data types are divided into primitive and complex types. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

In general, data types are collections of values from a given domain: the JavaScript number data type covers the domain of floating-point number values that can be represented in 64 bits. It also consists of a legal set of well-defined operations that may be performed on the values of the domain covered by the number data type. The operations on these numbers in JavaScript are defined by the arithmetic operators of the language.

Some languages separate numbers into **integer** types (whole numbers) and **floating point** types (decimal numbers). Even among similar languages such as C++ and Java, there may be different numbers of primitive data types.

Primitive Data Types

These data types are considered primitive because they relate very closely to the machine hardware. This means that the format, or bit pattern, of the actual values can be recognized by the registers and arithmetic-logic unit (ALU) of the computer. Some examples of primitive data types are number, character, and **Boolean** (hold the values *true* or *false*), as visible in Table 7.2.

Java	C++	Size	Value Range
short int	short int	2 bytes	–32,768 to +32,767
	unsigned short int	2 bytes	0 to +65,535
int	int	4 bytes	–2,147,483,648 to +2,147,483,647
	unsigned int	4 bytes	0 to +4,294,967,295
long int	long int	4 bytes	–2,147,483,648 to +2,147,483,647
	unsigned long int	4 bytes	0 to +4,294,967,295
long long int	unsigned long long int	8 bytes	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,808
	unsigned long long int	8 bytes	0 to 18,446,744,073,709,551,615

Table 7.2 Contrast Between Integer Data Types in Java and C+

We learned that strong typing refers to the characteristic of an HLL in which a variable is restricted to holding values of the type with which it is defined. The concept of **coercion** refers to the ability of a variable of a data type to be forced to hold a value of a different data type. In other words, coercion rules are a relaxation of type checking. For example, a Java int data type holds a whole number of size four bytes, while a short int holds a whole number of two bytes. We can legally assign the value of the short int to the int: it is coerced by the assignment, which makes absolute sense because the short value can fit into the longer value.

On the other hand, we cannot assign a Java 8-byte long data type to a Java 4-byte int; it is too big to fit, and if

we try, we will get a compile time error that will not allow the program to run. However, we can coerce the assignment by using a mechanism called a **type cast**. This is a mechanism in many HLLs which allows us to force the larger value into the smaller space given to us by the smaller variable. This can have side effects, which must be known by the programmer to use the mechanism effectively. The side effect of the Java long to int example is truncation: four of the bytes are dropped.

Complex Data Types

We have learned that some of the data types of a language are primitive types, meaning that data of that type can be directly represented in the registers and memory locations of the machine. However, languages usually contain complex data types as well.

A complex data type is one consisting of multiple primitive types used as its building blocks which is why we also call them composite types. These multiple types may be of the same type, as in a complex data type known as an array, or they may consist of collections of different data types in one construct, such as a C# class.

Arrays

An array is a typical composite type that is used as a data container. A great way to visualize an array is as a shelf unit, a connected structure where we can place items on each element. An instance of an array in this case could be a bookshelf that is meant to contain books (a book would be another composite type).

An **array** is a named variable that references a block of contiguous memory locations, and each "shelf" of the array is an element, which occupies exactly as many of the contiguous bytes as it takes to accommodate a value of the data type being stored. In the simplest type of array structure, an **indexed array**, the shelves are numbered with an index, starting at zero, or the lowest memory location. In a strongly typed language, all elements of an array must be of the same data type which means that every element will be of a uniform length in bytes.

Figure 7.10 illustrates an array in any number of HLLs including Java, C, and C#. We start off with the array declaration, which gives the data type of each of its elements, names the array variable numbers, indicates it is an array with the opening and closing square brackets ([]), and assigns five values to it with what is known as an **array initializer** (values separated by commas placed between curly braces). We can see that the length of the array is 5, the indexes run from 0 to 4, and each of the elements are contiguous in memory and are 4 bytes in length. The following statement assigns an element of the array to a variable:

```
int myNumber = numbers[5];
```

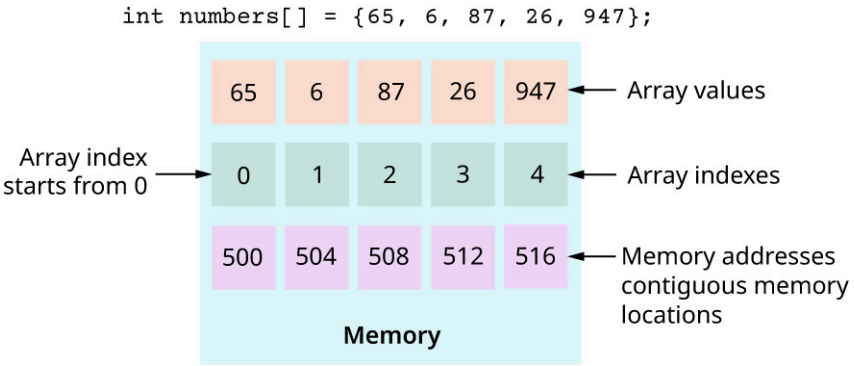


Figure 7.10 Each value in an array is assigned an index and a memory address. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Strings

Strings, which are another composite data type, are arrays of characters in most HLLs. In object-oriented

languages, they are quite a bit more complex. They are implemented in some HLLs as an array which holds individual characters as its elements. The OOP languages usually implement strings as objects with built-in functions (methods). Here is an example of a string in Java:

```
String myUniversity = "Union Technical College";
```

Reference Types and Pointers

In our study of variables and data types so far, we have examined the concept of a variable being a name-value pair. With primitive types, the memory location referenced by the name stores the actual value of the variable directly at that spot.

In the case of complex data types, things are not quite so simple. Let us take the example of a string. We can store a name in the string such as "Jimmy" to start. Now let us say that we change the value to "Johnathan" at some later time. The memory required to store the value has now changed and perhaps it will no longer fit in the original location. We have learned that the C language has a primitive data type known as a pointer. It is used for that reason.

Pointers are variables that hold actual computer memory addresses (references). They match the word size of the machine, which is typically 64 bits. We call a variable that holds memory addresses a **reference variable**. There is no such thing in Java or C. Therefore, when we create an array or a string in these languages, the value that is actually stored in the variable is the memory address of the place where the complex object exists. So, in the case of our string example, if we change the name, we can just change the value in its variable to be a different memory address—refer to Figure 7.3.

Variables

We learned in 7.1 Programming Language Foundations that a variable is a container that is used in an HLL to hold a value. In computer science we have many instances of this type of construct, which we call a **name-value pair**, a construct-like variable that is named and can hold values. The types of values that they may hold consist of the legal data types of the language.

Identifiers

A variable name is called an **identifier**. Different HLLs have different rules about legal identifier syntax. For example, in C# rules are as follows:

- An identifier cannot be a **keyword**, which is a word reserved by the language and that has a special meaning.
- A letter, @symbol, or an underscore must start an identifier while the remaining portion may be digits, the underscore symbol, and/or letters (different from this, an identifier in PHP starts with a dollar sign ($)).
- Identifiers are **case-sensitive**, where uppercase and lowercase letters are treated as distinct. Therefore, the C# identifier *myAge* is a different entity than *myAGE* (Fortran, BASIC, and Pascal are not case-sensitive).

GLOBAL ISSUES IN TECHNOLOGY

Learning About Programming: A Language Barrier for Non-English Speakers Learning HLLs?

Have you ever struggled to understand something because it was explained in a language that you do not speak? That is the challenge that many non-English speakers face when learning HLLs. Most HLLs use English keywords that make sense to the compiler but not necessarily to someone unfamiliar with the language. Since programming has become a worldwide endeavor, English keywords can be a stumbling block for non-English speakers learning HLLs. Fortunately, there is a bright side! While keywords are in

English, they comprise a relatively small set of words in a program. The real power of programming lies in its ability to work with data and instructions in any language, which is made possible via Unicode. Unicode can represent most international character sets, allowing programmers to use characters from almost any language alongside the English keywords.

But what about the future? As technology evolves, will programming languages find ways to become even more natural language-independent? Perhaps future HLLs will offer interchangeable keywords or entirely new approaches that do not rely on any given language.

Variable Declarations

A variable must be made known to a compiler or an interpreter before it may be used by a computer program. This process is variable declaration and/or definition. In strongly typed languages, a **variable declaration** consists of a statement which specifies the variable name and data type. Weakly-typed languages omit the data type when values are assigned to the variable, which may be different types at different times.

Variable definitions in various languages are as follows:

Java: `int myAge;`

JavaScript: `var myAge;`

PHP: `$myAge = 21;`

Assigning its first value to a variable is known as **initialization**, which may be done at any time after declaration, such as in the following Java snippet:

```
int myAge;
myAge = 21;
```

It is a best practice to always initialize variables when they are declared. This is known as declaration and initialization. This keeps the value that is stored from being undefined at any time, which can have grave consequences in code in various situations. For example, in the C programming language, failing to initialize a pointer to an array of characters in a program and copying a string of characters to the (uninitialized) memory location referred to by that pointer later in the program will crash the program. Here is another example in Java:

```
int myAge = 21;
```

Assignment

A **literal** is a value of one of the legal data types of an HLL that can be written directly into the code. For example, in JavaScript, one of the data types is numeric, which may be represented by either the literal whole number *2* or by the floating-point number *2.0*. In C++, a literal of the type *char* may be written as the single quoted sequence *a*.

Storing a value in a variable is carried out by creating an **assignment statement**: The value assigned may be a literal, or it may be the value that has been placed in another variable or the result of an expression. The value in a variable may also be replaced by using assignment. Therefore, variables may hold different values at different times.

In a PHP expression that makes up an assignment as shown, the variable is located at the left. Notice that the identifier starts with the dollar sign ($), complying with the identifier rules of PHP. The equals sign (=) is known as the assignment operator, as in most languages with C-like syntax (C/C++, Java, C#, Python, JavaScript, PHP).

```
$myAge = 21;
```

In programming languages, we refer to the left hand of a variable assignment statement (the variable) as the **lvalue**. The right-hand value (the literal) is referred to as the **rvalue**. The assignment operator is a **binary operator**, meaning it is surrounded by two operands. The operand is the lvalue or the rvalue on either side of the operator. The rvalue of a variable assignment statement may be the value of another variable as shown here or the result of an expression. An example of this in Java is as follows:

```
myAge = yourAge;
```

Let us examine the concept a little more deeply. Variables may be named memory locations. We give a variable a name so that it is easy for humans to deal with it. It is a best practice to use names that are indicative of both the purpose of the variable (what it will be used for) and the data type that it will hold. So the variable myAge in the previous example meets both characteristics. One HLL best practice is to use an agreed upon convention for variable names. An example is camelCase, a naming convention that eliminates spaces and punctuation in favor of capitalization of specific words; in this case, the first letter of the first word is lowercase and if the name has multiple words, the later words start with a capital letter (e.g., firstName and lastName). Other conventions exist such as snake case (e.g., first_name, last_name), kebab case (e.g., first-name, last-name), and Pascal case (e.g., FirstName, LastName).

When a program is compiled, the compiler allocates a memory location to hold variables' values and reserves the amount of memory necessary to hold such value based on the data type of the variable. The addresses of the memory locations that the compiler assigns to variables are relocatable, meaning that the linker may change these addresses when creating the executable version of the program, and the program loader will also change them when running the program to match actual memory addresses in the machine memory. Figure 7.11 illustrates what this looks like.

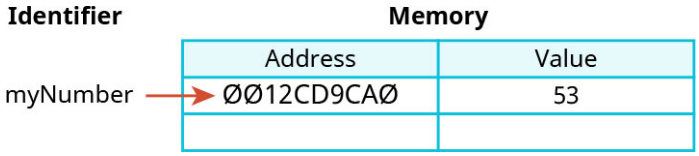


Figure 7.11 In JavaScript, the variable, in this case, myNumber, has a value that is assigned to a memory address. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Named Constants

A variable may hold different values within the data type restrictions of the language at various times during program execution. There are times when we would like a value to be assigned to a specific memory location and not allow it to be changed thereafter. Most modern HLLs and associated DLLs (dynamic link libraries) provide a construct for this purpose called a **named constant**. It is also a best practice in most language cultures to use capitals with underscores separating multiple words. The following statement in C++ creates a named constant:

```
const int PAY_RATE = 15;
```

The word *const* is a keyword, *int* is a C++ integer data type followed by the name of the constant, equals (=) is the assignment operator followed by the rvalue, and the statement ends with a terminator. It is a best practice in programming to use named constants whenever indicated and possible. In this example, if an employee pay rate was to change, it would only have to be changed in one spot in the program rather than having to locate its usages in many lines of code.

Operators

Much like in algebra, an **operator** in HLLs performs various types of operations (processes) on values.

Different HLLs may support operators differently. Some examples of the types of operations by which values may be manipulated are arithmetic operators (mathematical), relational operators (comparison), logical operators, and string operators (sequences of characters). They perform these operations within expressions of the language. To review, an **expression** is a construct in a programming language that evaluates two values. Operators may act upon different numbers of operands, usually one (a **unary operator**), two (a binary operator), or three (a **ternary operator**). Operators perform under certain rules that dictate the order of operations, or precedence. Although many HLLs use the same operators, when learning a new language, it is necessary to research its operators to identify the very few exceptions.

Arithmetic Operators

The arithmetic operators perform the four familiar mathematical operations on their operands: addition (+), subtraction (-), multiplication (*), and division (/). There is one more unfamiliar operation known as **modulo operator (%)**, which evaluates to the remainder left after division. Some languages also employ an **exponentiation operator (**)**, the operator that raises the value of one operand to the power of the second operand. The **increment operator (++)** and **decrement operator (--)** raise or lower the value in a variable by one, respectively, and are used in the vast majority of HLLs. These Java arithmetic operators are outlined in Table 7.3.

Operator	Name	Example Expression	Meaning
*	Multiplication	a * b	a times b
/	Division	a / b	a divided by b
%	Remainder (modulus)	a % b	remainder after dividing a by b
+	Addition	a + b	a plus b
-	Subtraction	a - b	a minus b

Table 7.3 The Java Arithmetic Operators

Relational Operators

The relational operators compare their operands, and expressions using them evaluate to the Boolean values *true* or *false*. Table 7.4 lists the relational operator symbols that are used in the vast majority of HLLs.

Operator	Name	Example Expression	Result
<	Less than	1 < 2	True
>	Greater than	1 > 2	False
<=	Less than or equal to	1 <= 2	True
>=	Greater than or equal to	1 >= 2	False

Table 7.4 The C# Relational Operators

Operator	Name	Example Expression	Result
==	Equal to	1 == 2	False
!=	Not equal to	1 != 2	True

Table 7.4 The C# Relational Operators

Logical Operators

The logical operators are used to connect two or more expressions. They evaluate the entire expression to the Boolean values *true* or *false*. Table 7.5 lists the logical operator symbols that are used in the vast majority of HLLs.

Operator	Name	Example Expression	Result
&&	Logical AND	(1 < 2) && (2 > 1)	True
||	Logical OR	(1 < 2) || (2 > 1)	True
!	Logical NOT	!(1 < 2)	False

Table 7.5 The JavaScript Logical Operators

Expressions using logical operators are evaluated based on a **truth table**, a chart that shows what the resulting value would be given each combination of operands. Table 7.6 shows the truth tables for the three logical operations we have studied.

A	B	A && B	A || B	!A
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

Table 7.6 Truth Table for Logical AND, OR, and NOT Operators

Combined Assignment

The assignment operator may be combined with other operators, usually mathematical, as a shortcut notation. This is called **combined assignment**. In this construct, the operation is carried out first, followed by the assignment. Table 7.7 shows the most common combined assignment expressions.

Operator	Example	Equivalent to
+=	x += 7	x = x + 7
-=	y -= 4	y = y - 4
*=	z *= 2	z = z * 2
False	a /= b	a = a / b
False	c %= 9	c = c % 9

Table 7.7 Combined Assignment Operators

Documentation and Comments

All programming languages allow a programmer to document their code to improve code readability. This is a best practice in programming and a prospective job applicant may not get a job without demonstrating documentation skills.

Documentation is carried out by using a structure called a **comment**, a container used to hold documentation in code. A comment begins with a defined symbol or set of symbols of the language followed by the comment text itself. Comments may be single line or multiline depending upon the symbol used. Single line comment symbols indicate that whatever follows on the line is ignored by the compiler or interpreter. Multiline comments have an opening and a closing symbol or set of symbols. Examples are as follows:

```
// single line comment in C#
int myAge = 21;
int myAge = 21; // another single line comment in C#
/* this is a multiline comment in JavaScript myAge will be used to hold the age of a
student */
int myAge = 21;
```

Other languages may use different syntax for comments; in Python the symbol is the pound sign (#). In BASIC it is the REM keyword, which is short for “remark.”

HLL Expressions and Statements

Commands in HLLs are structured as statements, and statements are made up of expressions. We have learned that an expression in a programming language evaluates to a value. Examine the following statements in a C-like language:

```
int a = b + c * d;
```

If it is given that b = 4, c = 6, and d = 2, a simple scan of the statement would have the first addition statement evaluate to 10, with that being used to evaluate the resulting statement of 10 * 2, giving a value of 20. However, this calculation is incorrect because expression evaluation within statements follows order of **precedence**, the rules that determine the order in which operators in statements are evaluated. Therefore 6 * 2 will be evaluated first, giving the value 12, and then 4 + 12 will be evaluated, giving a final result of 16.

Parentheses may be used to modify the order of precedence in statements. Expressions within parentheses are always evaluated first. If we were to rewrite the example:

```
int a = (b + c) * d;
```

we would come up with the result of 20. Table 7.8 shows the results of various expressions when following the rules of precedence and applying parentheses to them.

Expression	Value	Parenthesized Expression	Value
5 + 2 * 4	13	(5 + 2) * 4	28
10 / 5 - 3	-1	10 / (5 - 3)	5
8 + 12 * 2 - 4	28	8 + 12 * (2 - 4)	-16
4 + 17 % 2 - 1	4	(4 + 17) % 2 - 1	0
6 - 3 * 2 + 7 - 1	6	(6 - 3) * (2 + 7) - 1	26

Table 7.8 Expressions and Their Values

Some HLL expressions that use the logical operators (Java, C++, C#, JavaScript) do not have to be completely evaluated for their results to be known, a concept called **short circuiting**. Referring to the truth tables of the logical *&&* and *||* operators, we can see that in a && operation, the only way a result of true can be obtained is if both operands are true. Therefore, if the first operand is false, the expression does not have to be evaluated further—it is false. This is short circuiting. Here is an example in JavaScript:

```
if (x == y || today == "Tuesday") {
// do_something
}
```

If the operand on the left side of the logical operator || evaluates to true, then the expression is true, and the expression on the right side of the || operator does not need to be evaluated. Short circuiting increases efficiency and performance.

Flow of Control

As we learned in Introduction to Data Structures and Algorithms, we need to define the steps to be taken to solve a problem or complete a task. The order in which, or if, the statements of a program are executed is called **flow of control**. By default, program statements execute in sequential order from an established starting point; however, the flow of control can be modified. This is necessary in order to have the ability to model the situations of the real world that our algorithms represent.

Sequential Execution

Executing statements in the order in which they appear, **sequential execution**, is a linear ordering of statements in which one statement directly follows another. This is the default flow of control; it is automatic. An example of sequential execution would be the following:

```
int myAge = 35;
String myName = "Johnathan";
boolean isStudent = true;
```

These statements will execute one at a time in the order in which they are written.

A **code block** is a statement that consists of one or more statements that are structured in a sequential group

and delineated as such. This is necessary so that an entire group of commands may be executed as a single sequential structure. In the C-like languages, opening and closing curly braces ({ }) are usually used to denote the beginning and end of a code block. We can modify the preceding example into a code block as follows in Java:

```
{
   int myAge = 18;
   String myName = "Johnathan";
   boolean isStudent = true;
}
```

Note that variables defined within code blocks only exist in the context of that code block, which makes it possible to manage the scope of variables within code more precisely. It is a best practice in programming to indent code contained within structures so that the code is both readable and maintainable.

Selection

Decision making in programming is called **selection**. A decision-making construct allows us to choose from one or many paths of execution in a program. To see how they work, one must understand the concept of conditional expressions.

Conditional Expressions

Sometimes called a Boolean expression, a **conditional expression** evaluates the Boolean values of *true* or *false*. Based on a Boolean result, a computer may decide which path of execution to take. Some examples of conditional expressions follow:

- `value1 == value2`
- `value1 != value2`
- `value1 < value2`
- `value1 <= value2`

Selection Statements

The simplest form of selection statement is a decision structure known as a one-way branch, as displayed in Figure 7.12. Most languages represent a selection statement that is a one-way branch with a construct known as an if statement. Other languages may use a slightly different syntax. In Java, a relatively simple if statement using a code block looks like this (it is recommended to use code blocks for single-line selection statements to avoid errors subsequently when/if more statements are added):

```
short temp = 90;
(…)
if (temp >= 95) {
   System.out.println("It is hot");
}
```

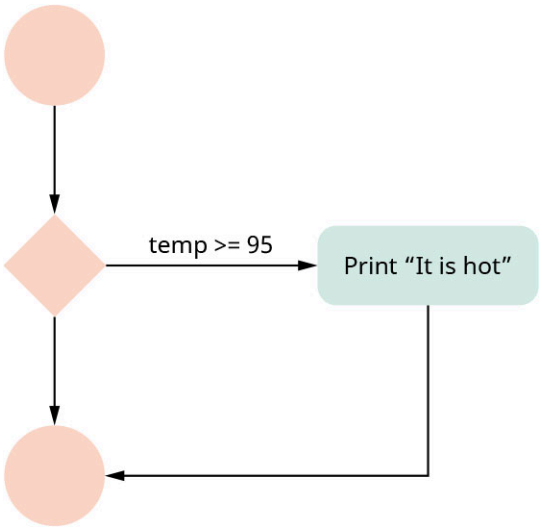


Figure 7.12 This unified modeling language (UML) activity diagram represents a selection statement that is a one-way branch. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The only type of conditional statement that a language needs is the simple one-way branch and any number of them may be placed in sequential order to carry out any decision-making task. However, a variety of decision structures have evolved into different flavors that make programs easier to code, more readable, and more maintainable. Its logic is shown in Figure 7.13.

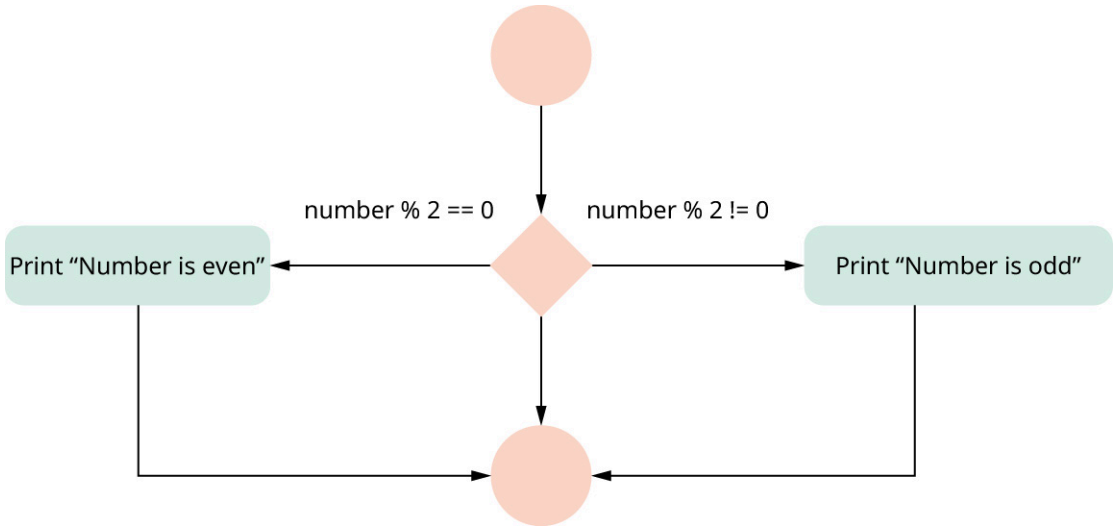


Figure 7.13 A unified modeling language (UML) activity diagram representing a selection statement that is a two-way branch. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

In Java, this might be coded with a two-way branch using an *if...else* statement:

```
int myNumber = 4;
if (myNumber % 2 == 0)
{
   System.out.println("The number is even.");
}
else
{
   System.out.println("The number is odd.");
}
```

Sometimes decisions might result in three or more possible pathways. In the C-like languages, this situation might be coded with a multi-way branch using an *if...else...if* statement:

```
if (myGrade >= 90)
{
   System.out.println("You got an A.");
}
else if (myGrade >= 80)
{
   System.out.println("You got a B.");
}
else if (myGrade >= 70)
{
   System.out.println("You got a C.");
}
else
{
   System.out.println("You need to work harder.");
}
```

Many programming languages have employed an additional selection statement for times when decisions result in three or more possible pathways. This is known as a switch or case statement, and it can be much more readable and maintainable than many if statements strung together. This looks like the following:

```
switch (myGrade)
{
   case 90: System.out.println("You got an A.");
   break;
   case 80: System.out.println("You got a B.");
   break;
   default: System.out.println("You need to work harder.");
   break;
}
```

Iteration

Also known as looping, **iteration** is going around and re-executing sequences of statements. One of the great strengths of computers is their ability to repeat actions over and over. Iteration structures allow us to write statements to be repeated just one time with the repetition monitored by flow of control structures.

The number of iterations is controlled by conditional expressions much like selection. The type of conditional statements used to control loops are categorized as either condition-controlled or count-controlled. In the **condition-controlled** scenario, the loop will continue to iterate until a condition is met. In a **count-controlled** situation, the loop repeats a specific number of times.

Iteration structures may also be categorized as top-tested or bottom-tested. In a **top-tested loop**, a condition is set at the entrance to the code block. If the condition is met, the loop executes and will continue to repeat until the condition is false. Note that if the condition in a top-tested loop is never true, the loop will not execute at all (for example “while True:” as the top-test of a loop in Python will result in the loop being executed forever).

In the **bottom-tested loop**, the **sentinel**, the expression that sets the condition at the entry or exit of a loop for continued iteration, is at the exit after the code block. A loop such as this is guaranteed to execute at least

one time. The sentinel decides if the loop runs again. It is useful to guarantee at least one repetition of a loop.

As with selection statements, most languages have a variety of structures to choose from and just like in selection, there is only one that is necessary to do any kind of iteration: the top-tested condition-controlled loop. Its logic is represented by Figure 7.14. Note that it is assumed in this example that the variable "raining" can change as a result of an external event in reaction, say, to the output of a sensor that detects rain and issues a callback to an event handler (not shown here) that changes the value of the "raining" variable to being true.

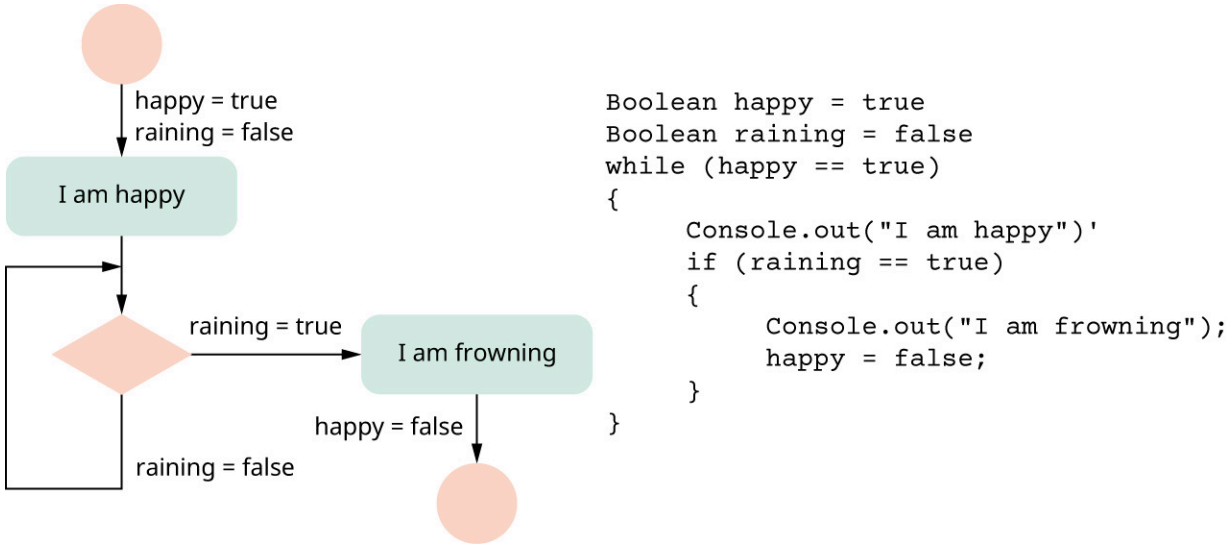


Figure 7.14 This unified modeling language (UML) activity diagram represents a while loop. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

In most modern languages, a top-tested loop is known as a while loop. Its syntax in C# is as follows:

```
Boolean happy = true;
Boolean rainin = false;
while (happy == true)
{
   Console.out("I am happy");
   if (raining == true) // raining value controlled by event handler
   {
      Console.out("I am frowning");
      happy = false;
   }
}
```

The sentinel in this loop is the variable happy, and if this condition remains true the loop will iterate.

This type of loop is **non determinative**, a loop for which we cannot predict the number of iterations (condition controlled). A **determinative** loop that is predictable in that its number of iterations will execute exactly five times and could be constructed in C# as follows:

```
int count = 1;
while (count <= 5)
{
   Console.out("I am smiling");
   count = count + 1;
}
```

In most modern languages a bottom tested loop is known as a do...while loop as highlighted in Figure 7.15. It's syntax in C# is as follows:

```
Boolean happy = true;
Boolean raining = false;
do
{
   Console.out("I am smiling");
   if (raining == true) // raining value controlled by event handler
   {
      Console.out("I am frowning");
      happy = false;
   }
} while (happy == true)
```

The sentinel in this loop is still the variable happy; the loop will iterate at least once, and if this condition remains true the loop will continue to iterate.

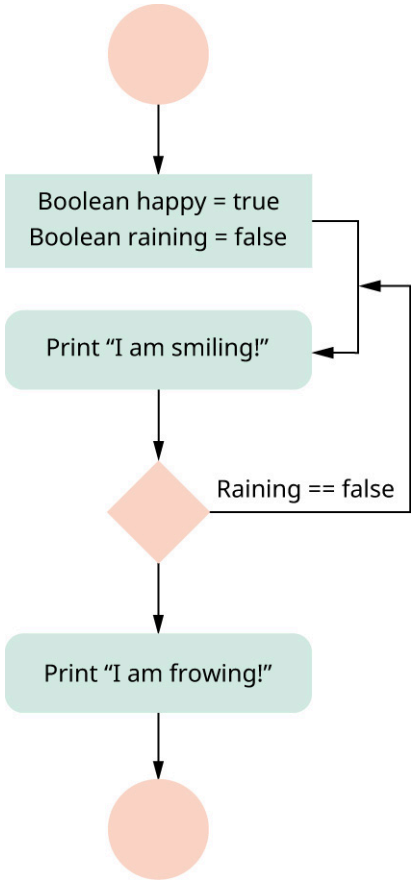


Figure 7.15 This unified modeling language (UML) activity diagram represents a do...while loop. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The sentinel in this loop is again the variable happy; it will iterate at least once and will continue if this condition remains true. This type of loop is non determinative because we cannot predict when it will rain and, therefore, how many times it will execute. We could construct it in such a way as to make it count-controlled. For count-controlled loops, another style is a **for loop**. This structure is top-tested and count-controlled. It is concise, elegant, easy to maintain, and efficient. An example of a for loop in PHP is as follows:

```
for ($count = 1; $count <= 5; $count = $count +1) {
   print("the count is: "", $count); }
```

The rules are as follows: inside the parentheses, the first expression is executed only once at the first entrance to the loop. The second expression is the sentinel and it is tested at all iterations of the loop. The last expression is executed at the bottom of each iteration and sets the stage for the next iteration.

Functions

Functions are sometimes referred to as, or replaced by, subroutines, subprograms, procedures, or methods. Some computer scientists consider them to be flow of control constructs because calling upon a function to do a job pulls the execution out of the regular sequence. In general, **function call** is code which invokes the function as well as passes to it values that may be needed for it to do its job while being designed to **return** values back to the place at which the function was called. However, in some programming languages such as C, functions can only return a value of a given type or no value. In that case, function parameters may be passed as references (i.e., by passing a pointer as a parameter) and the values associated with these parameters may be changed by the function.

A JavaScript alert is an API function that is designed to send a message to a web page and wait for the user to acknowledge it via an OK button, for example. The following illustrates code that invokes such an alert function:

```
alert("This message is for you, click OK to continue");
```

Modularity

This refers to the need for a program to be broken into various tasks. An employee program would probably be carrying out many tasks, one of which might be to compute the weekly salary. Modular design allows us to specify these tasks, name them, and call upon them without the finished code. We can define the tasks as functions and build the stub of them without all the details. Then we can call them from our main flow of control and have them respond to the call.

Maintainability

If we had to change the way we pay our employees, perhaps due to a change in deductions, we have to change that code in a multitude of places. By using a function to carry out the job for all employees, changes can be limited to just one place in the code.

Reusability

Building functions to carry out tasks means that we can either reuse the code in other software and/or call upon the same code from other locations. An API that is shipped with most languages is a perfect example. Our code calling upon the PHP print routine shows the efficacy of this—it is not a part of the language itself; rather, it is part of the API that is shipped with it.

Function Signature

To build a function in an HLL, one must define its **function signature**. The signature defines a function and informs the compiler or interpreter of details that it needs in order to call upon that function to execute. It also defines what it may return to the code which called the function. The following illustrates a function signature in Java (in this case, the function may be a method) for the task of paying an employee:

```
public double payEmp(String empName, double empBasePay, double empHours);
```

The function can be called upon to do its job with the following code:

```
myPay = payEmp("John Doe", 15.0,40.0);
```

The pay amount would be computed by the function and the resulting value would be returned to the point of the call and placed in the variable.

The word *public* is a keyword in Java that is called an **access modifier**; it denotes where in code this method may be accessed from. In this case, it is available from anywhere in the program code.

The keyword double indicates that this method will return a value of that data type to the place where the method was called. In this case, we want it to return the total pay for the employee. In Java, if we use the keyword **void** for the return, it tells the compiler that we will not be returning any value.

The identifier payEmp is the name we have given to the method. Its syntax follows the same rules and best practices that are used for a variable identifier in this language.

The parentheses indicate that this is a function (method), thereby delineating it from a variable identifier.

A **formal parameter** represents values that the function needs to receive to do its job. Parameters are not required, but if they are specified, the call of the function must pass actual parameters values for them into the function. Not doing so would result in a compile error which will not allow the program to run. In fact, not complying with any of the signature items results in a compile error.

Function Call

A function call in an HLL must comply with the function signature. A Java call to the function looks like this:

```
double thisSalary = payEmp(thisName, thisPayRate, thisHrs);
```

Starting from left to right, we declare a variable thisSalary and assign to it the value that will be the return of the function. We then call the function, passing the values that are required for its defined parameters. The term **argument** is used for the values of a function call which must match the parameters in both number and data type.

Parameter Passing

Parameters are passed using one of two methods. In **pass by value**, a copy is made of the value and the copy is passed as an argument to the parameter. When it encounters the parameter, the value is assigned and the values in the original variable cannot be affected by any changes which may occur inside the function.

On the other hand, **pass by reference** is employed in Java to pass a complex data type, meaning that the value stored inside the variable is a pointer to the memory address of the actual complex object. The value copied into the parameter in this case gives the code access to the object itself within the scope of the function so any changes made to its value will be reflected in the actual object that was passed to the function as shown in Figure 7.16.

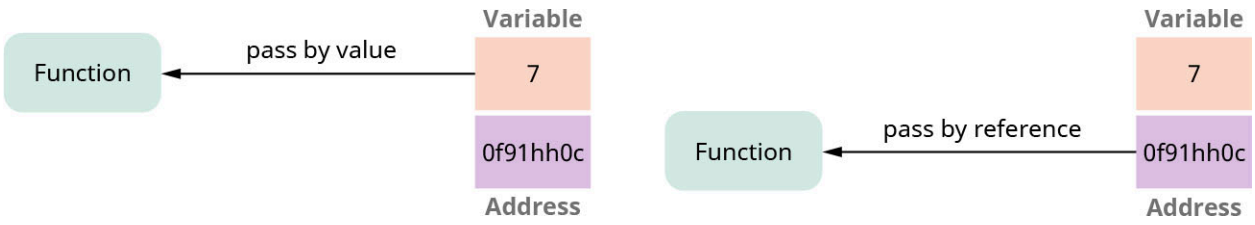


Figure 7.16 Compare passing a parameter by value versus passing it by reference in C++. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Let's code the entire function in Java to get a complete view of its components:

```
public double payEmp(String empName, double empBasePay, double empHours) {
    double empPay = empBasePay * empHours;
```

```
    System.out.println(empName + " is being paid " + empPay);
    return empPay;
}
```

In the example, empBasePay and empHours are passed by value of primitive data types. On the other hand, empName is passed by reference because string is a complex data type.

Call Stack

The **call stack** (execution stack) is a data structure that controls the calling and return of functions. Other duties include storing local variables, which dictates **scope**, defined as the visibility and lifetime of variables. It also has control of parameter passing.

Describing the call stack is usually done by comparing it to a cafeteria tray unit. Clean trays are pushed onto the unit as they come in and are popped off the unit as required. This concept is known as last in, first out (LIFO). Thus, when a subroutine is called, it is pushed onto the call stack. Its presence and all of its parts on the stack are called a **stack frame**. When it finishes executing, it is popped off. The flow of control of the program follows the current state of the call stack. A call stack is outlined in Figure 7.17.

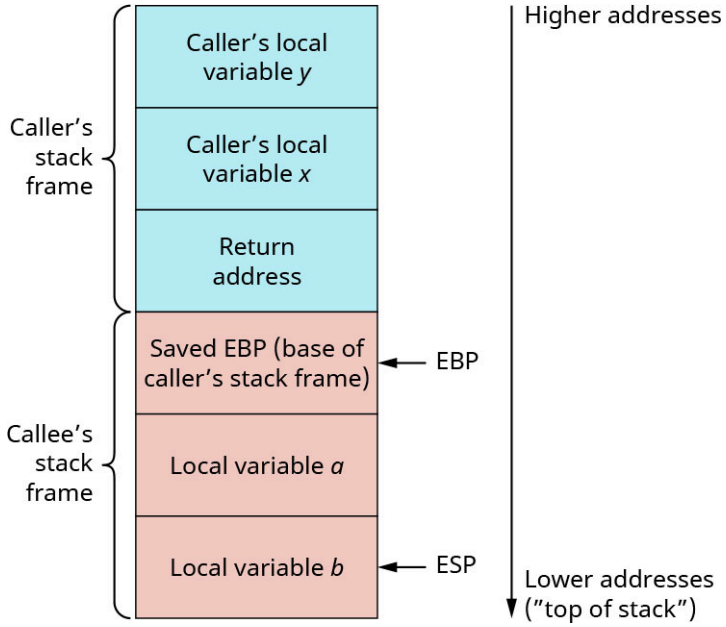


Figure 7.17 This is representative of a call stack. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

In this implementation, function1 is called first. It is pushed onto the stack in stack frame format and the parameters with their values are pushed first followed by the return address so that the flow of control can revert to the location of the function call after the stack frame has been popped off the stack. Lastly, the local variables are pushed onto the stack frame as they are defined. In this example, function1 has called function2 and has been pushed onto the stack. When it is popped off the stack, control will revert to the function1 stack frame.

Looking at function1, we see that the parameters come into existence when the function is pushed onto the stack and disappear when the function is popped from the stack. The same is true of any local variables declared in the function. This is an illustration of the scope of a variable which controls both the visibility and lifetime of a variable (i.e., a variable that is local to a function is only visible within the scope of that function for the duration of execution of that function). This scope is local because the visibility and lifetime is local to the function; they do not exist and cannot be seen from anywhere else in the program.

Recursion

Recursion is the sequence in which a function calls itself. If a function calls itself, then the next instance of the function will also call itself. This proceeds onward infinitely until the call stack runs out of memory, unless there is a means within the function to shut down the process. This condition is known as **stack overflow**.

Certain problems or tasks that we want a program to engage with lend themselves to recursion, such as with a rocket launch. If we want to program the display of a countdown sequence to launch, we can do it recursively. The signature, known as a prototype in C++, might look like this:

```
void countDown(long);
```

As indicated by the void, the function will return nothing to its calling point. It has a parameter that represents the starting point of the countdown. Therefore, if we want to start the countdown at 100, the call will look like this:

```
countDown(100);
```

Building out the recursive function could be done as follows:

```
void countDown(long count) {
   if (count < 0)
   {
      return;
   }
   cout << count << endl;
   count = count - 1;
   countDown(count);
}
```

Well-Structured Programs

A major challenge with writing well-structured programs is the complexity of today's systems, especially when they are influenced by so many programmers. A common dilemma is how to divide and conquer the task of writing programs when the effort relies on the notion of **modularization**, or splitting the large job into independent units which may be then called upon. We can now expand that concept to include entities that exist in completely separate pieces of source code.

Programs are very efficiently built out of modules, as shown in Figure 7.18. A **module** is a component of a program and has a **public interface**. For example, in Java the interface only includes documentation of what a user needs to access an object derived from a class, along with its attributes and methods. The interface documents exactly how components can be used by other programmers: what they are named, what, if anything, they require to do their jobs, and what they may or may not return. Documentation is necessary for another programmer to know how to employ them.

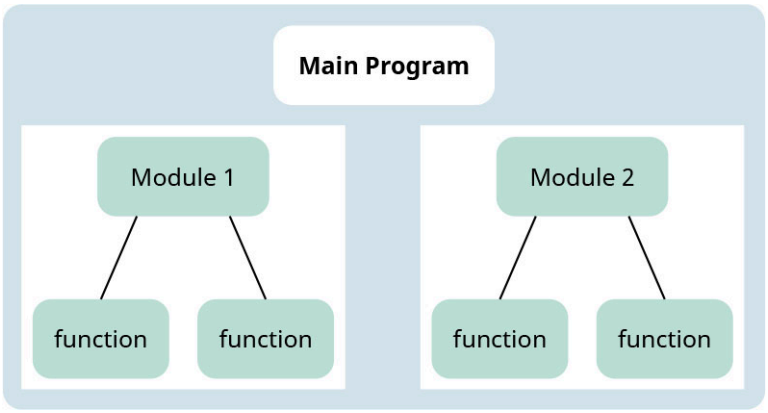


Figure 7.18 This diagram shows a program made up of multiple modules, each of which contains multiple functions. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

This allows us to engage in **information hiding**: the hiding of bits and bytes of implementation that the user does not need to know to make use of the module. A good example is the PHP print function; its implementation is not located in any place that we can see. Rather it comes from a module that somebody has programmed and is shipped with the language API, such as in Figure 7.19.

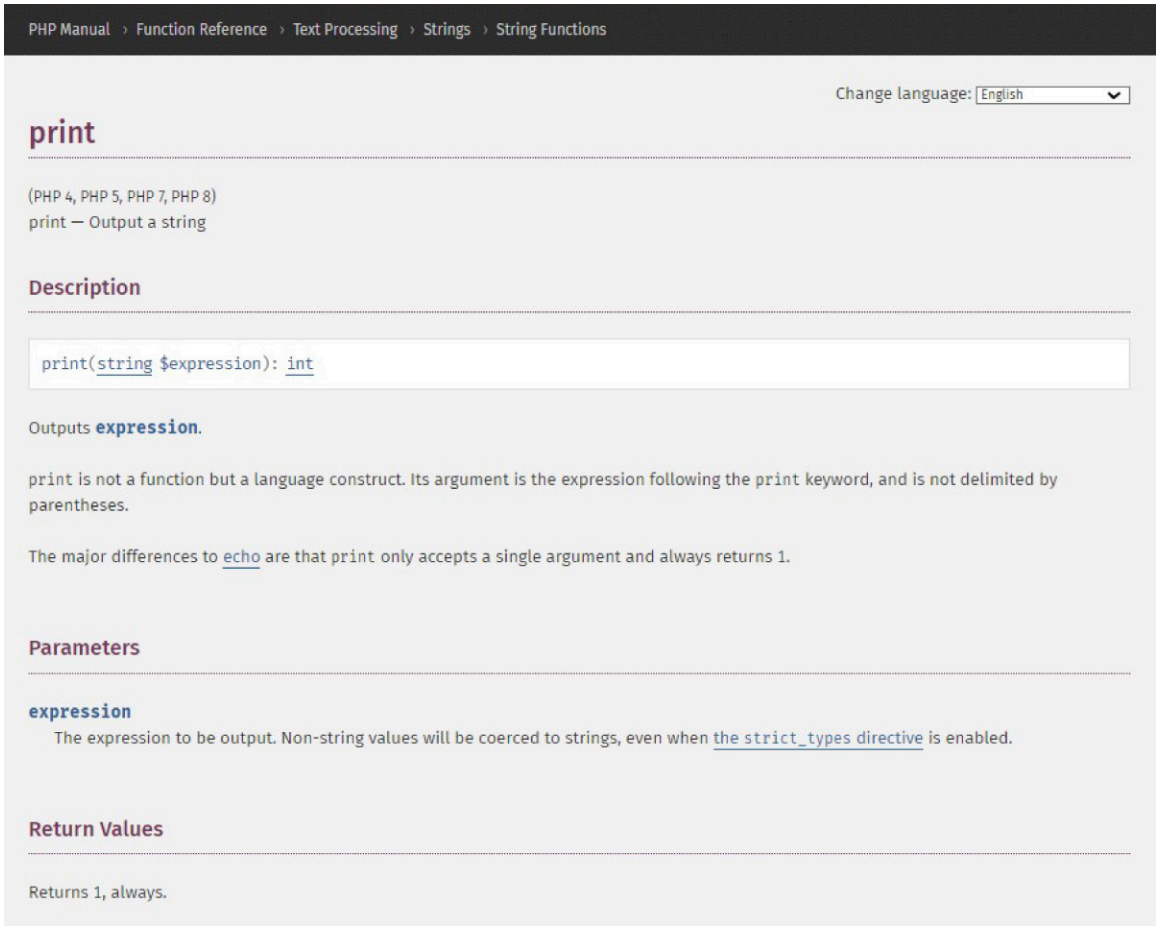


Figure 7.19 This portion of the PHP manual gives the documentation for the PHP print construct. (credit: The PHP Documentation Group, CC BY 3.0)

Exception Handling

No code is perfect; therefore, any computer program can generate runtime errors. This type of error usually indicates that an exceptional condition has occurred. These errors are generated by a hardware-detected run-time error or an unusual condition detected by software. Some of these errors include arithmetic overflow,

division by zero, end-of-file on input, and user-defined conditions (not necessarily hardware or software errors).

Most modern languages divide errors by category: a **runtime error** is an exception that is serious enough that it cannot or should not be handled by the software, and an exception is an unusual behavior that can be recovered using exception handling support the programming language provides (e.g., try/catch clause in Java). The proper terminology is that an error or exception has been thrown.

Files and Input/Output

File handling features and the input/output (I/O) capabilities (facilities that allow a program to communicate with the outside world) are highly dependent on both the operating system which is hosting the program and the HLL. All HLLs differ in the way that these are handled. The following are some examples of different output statements in various HLLs:

- Java: `System.out.println("Hello");`
- C#: `Console.Writeln("Hello");`
- C: `printf("Hello");`
- C++: `cout << "Hello";`
- JavaScript: `alert("Hello");`
- PHP: `print("Hello");`

There are as many ways of handling file I/O as there are HLLs. This is a subject that requires research with every new language.

LINK TO LEARNING

Aside from being one of the most used programming languages in the world, Java is a very teachable language. As such, the Basic Language Constructs of Java (https://openstax.org/r/76JavaLang) serves as a great reference for learning this new language.

7.3 Alternative Programming Models

Learning Objectives

By the end of this section, you will be able to:

- Discuss characteristics of functional programming
- Explain characteristics of declarative programming
- Distinguish the characteristics of object-oriented programming
- Explain HLL constructs used to support concurrency and parallelism
- Summarize when to use scripting languages

As we have learned, HLLs can be classified into various paradigms, two of which are imperative and declarative. There are other paradigms that fall within these major categories. Some worth investigating are functional programming, declarative programming, object-oriented programming, and parallel programming utilizing concurrency.

Functional Programming

In 1930, Alonzo Church developed the *lambda calculus* model of computing, which got its name from the Greek letter lambda (λ). In this model, each parameter was introduced by the lambda symbol and computation was performed by placing parameters into expressions in the same way that high-level programs transfer arguments to functions. Figure 7.20 highlights various HLLs.

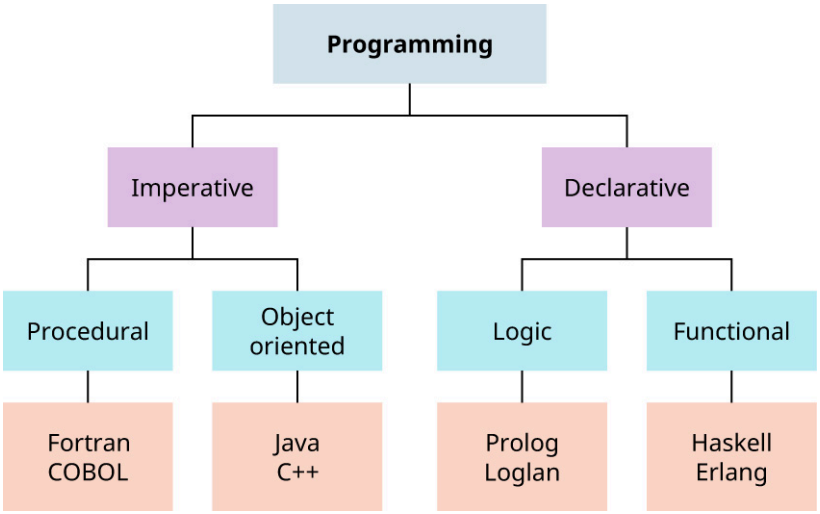


Figure 7.20 Functional programming languages can be differentiated by their paradigms. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Functional Programming Concepts

The functional programming model's key concept is that programs are constructed by composing functions and applying them. A **pure function** is one that will return the same result every time given the same arguments, and there is no reading of shared mutable state. In this context, **shared data** means that the data is available to multiple program locations or scopes, **state** represents data that are remembered over time, and **mutable** means it is changeable. If the data used by the function is shared mutable, it is possible to return different results for the same arguments. A pure function may not have any side effects, meaning it cannot modify any state that is not local to it. For this reason, functional programs do not have assignment statements, which is also known as referential transparency.

Some other necessary features and characteristics that may be missing in some imperative languages include first-class functions and high-order functions. A **first-class function** is one that can be assigned as a value to a variable. A **high-order function** is one that can take one or more first-class functions as arguments. It can also return functions as results, which gives a function the ability to apply functions to its arguments one at a time. This is very useful in making code both more flexible and simpler. It encourages the creation of smaller functions that take care of just one piece of a larger job, which makes for great maintainability as well.

Functional Programming Assessment

In the same way that you perform tasks over and over each day, functional programming treats functions used to execute these tasks as prime citizens. Some benefits of repetition include making programs easier to understand for the programmer as well as shortening programs. Some downsides of functional programming are having to move data through the functions, effective implementation is difficult (but not impossible), and new arrays must be created when data in an element are changed.

Object-Oriented Programming

A number of imperative languages are considered object oriented, and most modern HLLs support this paradigm to some degree. Its basic characteristic is to design software around the concept of a **class**, which is the blueprint from which objects are constructed. Constructing an object is known as **instantiation** or building an instance of a class. Objects are made up of attributes (characteristics) and behaviors (methods or functions).

The three main principles of OOP are encapsulation, inheritance, and polymorphism, which provide a very high capability of abstraction. With **encapsulation**, the attributes and behaviors of classes and objects are self-contained and go along with them throughout their lifetime. It also means that internal implementation is

hidden from outside the class or object. In **inheritance**, objects take on specified attributes from their ancestors. An example is that a Dog object inherits from an Animal parent object. Polymorphism means that behaviors that are inherited may perform in different ways that depend upon their context. A Dog is an Animal object and makes noise. A Snake is also an Animal object that makes noise—a different noise.

The degree to which an HLL conforms to these principles measures how object oriented it is. Java and C# are fully object oriented. C/C++ is a hybrid of procedural characteristics and OOP characteristics. Languages such as Python, JavaScript, and PHP partially ascribe to these characteristics. Therefore, they are sometimes referred to as object based.

Encapsulation

OOP languages contain features to implement encapsulation; they are self-contained and enable data hiding. These features include scope rules, for example, defining variables and methods within classes and using access modifiers.

Object-oriented languages often have constructors, and some have destructors which support encapsulation. A **constructor** is a specialized method that is called upon to instantiate the object. Very often, constructors have parameters which can be used to initialize the values of the attributes with arguments that are passed to them so that the variables are not directly accessed or assigned to. A **destructor** is used to destroy the instantiated object and recover its memory as in C++. Java and modern HLLs have a background running program for **garbage collection** that automatically destroys objects and recovers memory when there are no longer any references to them in the code.

They also employ the use of methods to both set and return the values in variables so that they are not directly accessed. Data hiding is further enabled in some modern HLLs such as Java. This engages a background running process for garbage collection that automatically destroys objects and recovers memory when there are no longer any references to them in the code.

Data hiding is one of the main objectives of OOP that falls under encapsulation. Its purpose is to address the details of implementation in a class or object that are irrelevant to users of the class or object. They need only know how to use them; therefore, the details are hidden and provide only what is necessary in its public interface.

Most OOP languages have the keywords public and private. They are used to mark the visibility and access to attributes and methods by areas of code that are outside the class code itself. We call them access modifiers. A **member** is an attribute or method that is encapsulated within a class or object. A public member is part of the public interface, and a private member is hidden. The following Java code is an abridged class definition that demonstrates the concept:

```
public class Rectangle {
   // attributes
   private double length;
   private double width;
   // constructors
   public Rectangle(double length, double width) {
      . . . . .
   }
   // methods
   public double getLength() {
      return length;
   }
   . . . . .
   public double getArea() {
```

```
        . . . . .
    }
}
```

The class itself is public, granting access to it and its contents. The attributes are all private, not part of the public interface. In this context, the designer has decided not to allow direct access to these attributes from outside. Instead, they are accessed by public methods, some of which are not shown. The methods that we employ, called getters and setters, can present our data to the outside world in any way we choose. The method can be described as a utility method that can provide the public useful data so that they do not have to compute it themselves.

Inheritance

Figure 7.21 illustrates the implementation of **single inheritance**, the methodology used by most modern OOP HLLs. Classes and objects can inherit from only one parent.

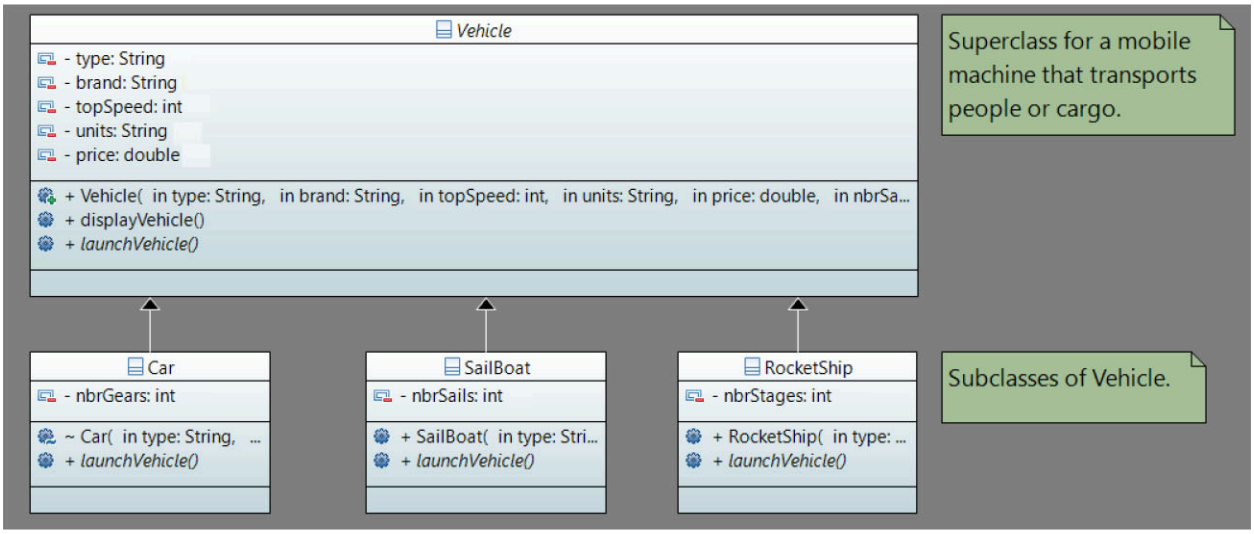


Figure 7.21 This UML diagram shows single inheritance in Java. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

We refer to the parent class as the **superclass** in an inheritance relationship and the child classes as a **subclass** which contains the attributes and methods of its parent class. In this illustration, all the Vehicle attributes are inherited by the Car and the other subclasses with the same exact access modifiers. The same is true of all the methods.

Only C++ implements **multiple inheritance**, inheriting attributes and methods of more than one parent. This concept is exceedingly messy and can cause major problems in implementation. An example is where both parents have a method of the same name and it cannot be specified which one is inherited or overridden. To implement behavior that is shared from multiple parent classes, Java introduced the interface, which can be implemented in C++ to solve multiple inheritance issues.

An **interface** resembles a class definition, but it is a collection of methods only. These methods are constructed as abstract methods. An **abstract method** is declared without a code block for implementation. A class can use one or more interfaces. The following is a Java class definition that makes use of an interface:

```
public class Rectangle implements geometryMath{
}
```

The interface acts as a contract with the programmer. If the programmer declares that a class implements an interface, it is required that the class provide implementation code for the functions that make up all of the

interface. An **override** defines another method with the same name and modifies its signature. The override forms a concrete instance of what was an abstract method in the interface. Interfaces appear in Java, C#, Go, or Ruby, and are the dominant approach, superseding true multiple inheritance.

Polymorphism

The feature of OOP in which methods that are inherited may perform in different ways in different subclasses depending on their context is called **polymorphism**. This is implemented overriding inherited methods in a subclass or the ones in an implemented interface. In the UML diagram, the launch behavior of a car is very different from that of a sailboat or a rocket ship. The resulting behavior is dependent on the context because it is not set at compile time. Rather, the runtime decides what to do when it encounters an object based on its class. The behavior morphs and produces different results for the same method calls depending on the object in hand, which is **dynamic method binding** or **late binding**, meaning that all methods are resolved dynamically at runtime, not by the compiler.

OOP Assessment

Remember that some languages like Java and C# are completely object-oriented; some like C++ are hybrids of procedural and object-oriented, and some are on the path toward complete object-oriented like JavaScript, PHP, and Python. Any assessment of the object-oriented paradigm must be taken in the context of where a particular language is on the scale of object orientation.

OOP advantages:

- Strong abstraction with the ability to reuse code efficiently
- Improved software development productivity and maintainability
- Faster software development, thus lower cost of development as it is very efficient for parallel development
- Adapts well to **parallelism**, where programs can have more than one part of the code running simultaneously
- More consistent software—dynamic method binding producing polymorphism are a mandate that subclasses implement the same behaviors with differences

OOP disadvantages:

- Steep learning curve to master OOP
- Program creation can be complex
- Slower execution due to more generated instructions by the compiler

Concurrency and Parallel Programming

A process, or thread, is an active execution context, meaning that it is an executable code block. The ability of an application to multitask is called concurrency. When an application can **multitask**, it can process more than one task at the same time. It is the case of their executions overlapping in time. In concurrency we have the illusion of simultaneous execution. A single core CPU can only run one task at a time while the processor rapidly switches between concurrent processes, thus creating the illusion. We refer to the executable unit as a **thread of control** because the processor is controlling which thread is executing at a designated time.

In parallelism, programs can have more than one thread of control running at the same time. This is possible only if we have multiple or multicore processors which allow us to produce simultaneous execution. We must program for both concurrency and parallelism in the same way because the execution logic is the same, only the physical hardware is different. The process of coding for these environments is concurrent programming or parallel programming in which a thread can be thought of as an abstraction of a physical processor.

Race Conditions

When the condition of a program and its behaviors are not synchronized, a **race condition** occurs. Race

conditions are all about timing because anomalies can occur when we do not know which process will finish first or which part of the program is currently executing. We can exert control over when a process pauses or ends in the run-until-blocked scenario where some mechanism pauses a thread. This situation is not always a negative one; race conditions can sometimes be positive by allowing multithreading to compete unchecked for processor attention.

In many multitasking situations we want to avoid race conditions and execute a degree of control of execution; in other words, we want to synchronize threads in either the interleaved or parallel situations.

Synchronization

The building of cooperation between threads of execution, or synchronization, is often handled by first ascertaining which segments of code form a critical section. A critical section is a code block that influences the results of concurrency or parallelism. A simple example might be an employee program in which one concurrent function calculates employee pay and another one prints the paycheck. The print function cannot proceed to put the amount on the check until the calculation function returns the amount. Other situations that need the use of critical sections are as follows:

- When multiple threads need access to shared memory or resources and the timing of the access can be critical to the eventual result
- Times when processes need to communicate with each other to proceed, often implemented as message passing between processes
- Cases where all concurrent processes must finish before execution of the program proceeds
- Cases where one or more of the processes need some result from another process

As visible in Figure 7.22, when processes run in a requested order, synchronization is occurring.

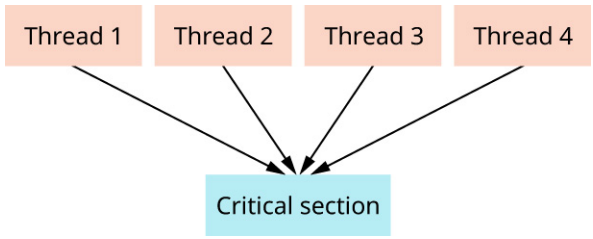


Figure 7.22 When multiple threads seek access to a critical section, they need synchronization. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Since the goal is to avoid negative race conditions, we do not want to over-synchronize because it lessens the degree of parallelism that we need for performance.

Implementing Synchronization

Synchronization must be carefully implemented to avoid situations that synchronization itself can cause:

- **starvation**: when a process must wait to enter a critical section, but the other processes monopolize the section, and the waiting process doesn't get processor time.
- **busy waiting**: when a process continually looks to see if it has access to a critical section, taking processor time from all processes.
- deadlock: when multiple threads need a critical section that is being monopolized by one of the processes.

Java is known for its strong implementation of synchronization. When marked as synchronized, it is guaranteed that it can have only one thread executing in it at the same time. Any other thread trying to enter the method or code block is blocked until the running occurrence exits the method or block. The following Java code shows the syntax of using the keyword on a method:

```
public synchronized void calcAverage() {
   // all of the method code
```

```
}
```

A synchronized method is coordinated on the object that owns the method. In OOP each instance of the object may have its own synchronization. For example, we might have a Car class that owns a method called start(). Each car instance we build has its own copy of the method; therefore, if we are starting a car, no other process can start that particular car. But if we have more than one car, another one may be started concurrently.

The following Java code shows the syntax of using the keyword on a code block within a method:

```
synchronized (this) {
   // critical section
}
```

This code will execute exactly as if it was a synchronized method. This keyword refers to the current object, so this code is also synchronizing on the object which owns the code block. This usage provides much more granularity and efficient execution times.

Another powerful tool for synchronization in Java is to declare a whole class as a thread, meaning the whole class is within a single thread of execution. There are two ways to do this. The first is having the class inherit from the Thread class from the Java API:

```
public class SomeThread extends Thread {
}
```

The other way is to have the class implement an interface from the API:

```
public class SomeThread implements Runnable {
}
```

Either of these methodologies provides access to the useful methods of the Thread class which are inherited and can be overridden to support the particular situation. The interface forms a contract with the programmer to implement these methods to support the particular task.

A Thread object is controlled by a priority-based scheduler. Some of the most important methods at the programmer's disposal are as follows:

- `start()`: causes the thread to begin execution
- `yield()`: tells the thread scheduler that the current thread is willing to yield its control
- `sleep(int time)`: causes the currently running thread to temporarily hibernate for a specified number of milliseconds
- `setPriority(int priority)`: changes the priority of the current thread
- `join(int time)`: waits for the indicated milliseconds before commanding the thread to die

CONCEPTS IN PRACTICE

Which HLL Works Best?

Modern HLLs combine a variety of programming models to make it easier to tackle problems in various domains and industries. Two of the models we have studied are object-oriented programming and concurrency.

The need for an object-oriented model may be dictated by the need for a high degree of abstraction capability. Other reasons include improved productivity in software development, cost, scalability, and maintainability.

OOP also adapts well to parallelism, which is another paradigm to consider. A need for speed can require the selection of an HLL that supports concurrency and/or parallelism well. It may be the case that the application we build requires both.

We have choices, the most popular of which are C++, Java, and C#. Narrowing down our choices is helpful and puts us in a great position to examine each to make a well-informed decision.

Sometimes OOP is desirable, but concurrency may not be. This would occur when programming operating systems such as Windows with C++. It has an object-oriented UI, but it must also manipulate the machine elements directly. In this case concurrency may not be desirable, so some modules should be programmed without concurrency by coding them in just the C aspects of the language.

Programming with Scripting Languages

Scripting languages have their roots in **shell scripting**, originally referred to as stringing together a group of commands to perform tasks on the user interfaces of various operating systems such as pre-GUI, Unix DOS, and CPM. Modern OSs still provide this facility in such programs as Windows PowerShell. You could string together a group of commands to perform tasks, either directly on the terminal or more effectively in batch files that the shells could execute. Batch files are created in a text editor to contain the script. They usually have a file extension that the shell can recognize, such as backup.bat, which might contain commands to backup an MS-DOS computer. These languages are interpreted at runtime and are not compiled.

Early scripting languages include the following:

- MS-DOS command interpreter
- Unix: the standard command line interpreter when running the Unix Bourne shell
- Microsoft PowerShell: used for automation and configuration management on Windows systems

Most of our HLL scripting languages used today evolved from these, particularly for programming the Web. Figure 7.23 also illustrates some of these languages.

- Perl is an older scripting language primarily used for web scripting that uses the Common Gateway Interface (CGI), an industry specification for web server communication with web browsers. The two different entities need rules by which to “talk” to each other, known as protocols such as HTTP. Perl has mostly been replaced by PHP, ASP.NET, and other web server scripting languages.
- Python is a general purpose HLL for both core and web programming.
- JavaScript is a general purpose web scripting language that is almost universal. It originally focused on front-end web browsers and now takes a great amount of market share on backend servers with the newer ES6 version.
- PHP is a very high-level scripting language for web servers that is platform independent.

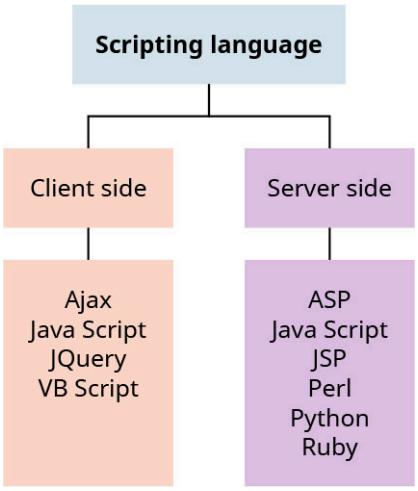


Figure 7.23 Scripting languages can be further categorized by whether they are on the client side or the server side. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Common Gateway Interface scripts are the original mechanism for server-side web scripting. A **Common Gateway Interface (CGI)** script is an executable program residing in a special directory known to the web server management program. When a client requests a web address known as a Uniform Resource Indicator (URI), the server executes the program and outputs HTML to a screen that is readable by the browser, which renders it to a user. For example, when you make an online purchase, the web server launches multiple scripts behind the scenes to gather product information, images, and pricing. Most websites use loading screens to allow for one or more scripting languages. Though widely used, CGI scripts have several disadvantages, with the main one being slow loading times by web pages due to having to launch each script as a separate program.

A **server-side script** runs on a server in a web designer's domain and generally is faster than a CGI script due to its ability to compile in real time; therefore, in PHP, Python, Ruby, and backend JavaScript ES6, the client only sees the standard HTML.

A **client-side script** runs on a client computer, usually under the control of a web browser and needs an interpreter on the client's machine. Client-side scripts often use an **embedded script** in which one language is embedded inside another. A frequent example is JavaScript embedded within HTML. JavaScript for interactive features on a web page is almost the universal standard. Microsoft produces a language called TypeScript which is a superset of JavaScript. It is a compiled language which generates JavaScript. It has the advantages of being strongly typed and has stronger OOP features. It can be used on the front end and back end, although it is not as popular as JavaScript ES6 on the server side.

Both client-side and server-side languages like JavaScript and PHP can be embedded into HTML code with embedded elements. An embedded element is a program designed to run inside some other program like HTML. On the client side, we embed JavaScript into HTML <script> elements and a browser calls the JavaScript interpreter when it encounters one. On the server side, we can embed PHP the same way with some rules. The script must be installed on the server and must have a file extension of .php.

LINK TO LEARNING

Object-oriented programming has become an enormous force in the modern construction of complex applications. There are many languages to choose from, and this introduction to object-oriented programming (https://openstax.org/r/76ObjOrProgram) is a great reference to have when selecting an HLL.

7.4 Programming Language Implementation

Learning Objectives

By the end of this section, you will be able to:

- Discuss how to build and run programs written in various HLLs
- Describe the work of an HLL runtime management implementation
- List and explain various HLL optimization methods applicable to programs

To implement programs that you create, you must use a process to generate machine code from source code. As previously discussed, the major methods of implementing programming languages are compilation, pure interpretation, and hybrid implementation. These are complex processes best learned in stages. There are differences between a compiler and an interpreter, as shown in Table 7.9.

Compiler	Interpreter
Scans and translates the entire source code at once	Scans and translates the source code one line at a time
Takes a relatively large amount of time to scan and translate	Takes a relatively small amount of time to scan and translate but has an overall longer execution time
Always generates an intermediary object code and will need further linking, thus will need more memory	Does not generate an intermediary code so highly efficient in terms of its memory
Generates any error message only after it scans the complete program	Translates and executes the program until an error is encountered, then it stops working

Table 7.9 Differences Between Compilation and Interpretation

Preprocessing

The single process that takes the source code the programmer creates and transforms it into another language is called **compilation**. In short, it is the rendering from one language into another and includes a deconstruction of the input, and its product may be machine language. Modern compilation usually starts with **preprocessing**, a step in which the source code is manipulated to ready it for the compilation stage. Some of the actions the preprocessor can take are as follows:

- Removing any comments and white space
- Executing a **preprocessor directive**, which describes resources the program needs to compile, usually files to insert into the source code. The C/C++ preprocessor provides a fine example.
    1. The #include directive of C/C++ deals with a variety of files such as header or standard files containing the definitions of objects and functions from the API that are going to be used in the program. It can also take care of user-defined files or modules.
    2. The #define directive of C/C++ names a piece of code for the preprocessor to replace every time it sees the name.
    3. The #ifdef directive of C/C++ tells the compiler to include a specified piece of code in compilation depending upon some condition.
- Other tasks include pre-identifying high-level constructs such as loops and functions.

Compilation

The phases of compilation are grouped into three stages that are usually executed in three passes through the preprocessed source code:

- **front end** that is responsible for the analysis of source code
- **middle end** that is responsible for performing optimizations of the code
- **back end** (or code generator) that is responsible for the production of the target code

Front-End Compiler Structure

The front end is responsible for analyzing the source code for syntax, semantics, and other tasks. The first part of this process is **lexical analysis**, which converts the source code strings of characters into **tokens**. These are sequences of compiler recognizable atomic symbols which make it much easier for the next steps to recognize them.

The process proceeds to **syntax analysis**, usually referred to as **parsing** or reading the code to make sure it conforms with the syntax rules of the language. This is done to test conformability logically by breaking code into separate parts.

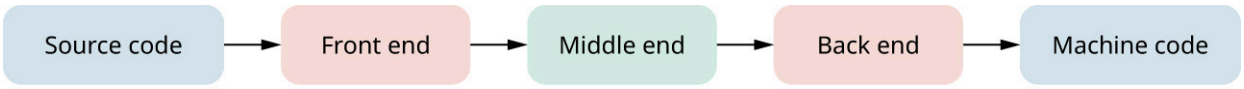
The next step is **semantic analysis**, which is the discovery of meaning in the code. It takes care of type checking in strongly typed languages. It also performs binding, which associates variable and function references with their declarations and definitions. The compiler runs a static semantic analysis which concerns itself only with the source code without testing inputs.

Semantic analysis cannot figure out all meaning in the code. Many languages also use dynamic binding (late binding) that defers the binding of certain elements such as objects and methods until runtime. This is particularly true of OOP languages which support inheritance and polymorphism. Some processes, such as when an array goes out of bounds, will not be known until run time, and are part of the program's dynamic semantics.

Semantic analysis concludes with the creation and management of the symbol table, a data structure that is used to connect every symbol with necessary information such as data type, scope, and memory location.

Middle-End Compiler Structure

The main job of the middle-end stage of compilation is **optimization**, producing what will be needed to generate the fastest and most efficiently running code as possible. The attributes which are usually optimized are execution time and space (memory usage). Figure 7.24 outlines this process.



Middle End	Task	Goal
	Optimization	Improve code
	Transforms	Analyze and rewrite code to reduce running time, space, and power consumption of the compiled code.

Figure 7.24 Code optimization is the phase between the front end and back end of the compilation process. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Intermediate Form

The result created after semantic analysis when the program passes all checks is called **intermediate form (IF)**. The nature of the IF is often chosen for machine independence and ease of optimization. IFs usually resemble the machine code for some imaginary idealized machine. An example is **managed code** which is targeted to run under a particular runtime environment. This is true of C# and other Microsoft.net languages. They are designed to run under its CLR, which manages the execution of Microsoft's .NET languages. In this

case, the purpose is to allow code from the different languages to work well together at execution time.

TECHNOLOGY IN EVERYDAY LIFE

HLLs and Big Data

There is a growing need to process massive amounts of "big" data to gather insights in real time about the best way a business can serve their customers. This is called data analytics, the systematic computational analysis of data or statistics. Netflix uses this technique as a means with which to plan its targeted advertising. Basically, Netflix monitors which types of movies its customers typically watch and collects related data that it uses to bring in new movies and advertise them to its customers. This has led to an increased use of concurrent/parallel processing frameworks that allow concurrent programming. The popularity has caused languages like Python to get a lot of attention for big data analysis. Do some research on Python and provide a well-documented opinion on the features of Python that support this.

Back-End Compiler Structure

The back-end process is known as **code generation**, the transformation of the code to the target or object language. It uses the intermediate form of the code in combination with the symbol table to carry this out. As previously stated, the final produced code is not necessarily machine language. In hybrid implementations, such as in Java and others, the code will still be a type of intermediate form and it is left for another process to turn it into machine code, usually at runtime.

Pure compilation code generation generally produces this phase (and all compile phases) while also relying on the symbol table, which is the structure responsible for following identifiers within a program and tracking the work of the compiler. Once compilation is complete, the debugger may retain the symbol table.

The back-end phase of the compiler may also perform machine-independent code generation. The purpose of this is to allow a piece of source code to generate instructions that will run on different platforms without change such as Windows and Android. Java is a great example of this, as referenced in Figure 7.25.

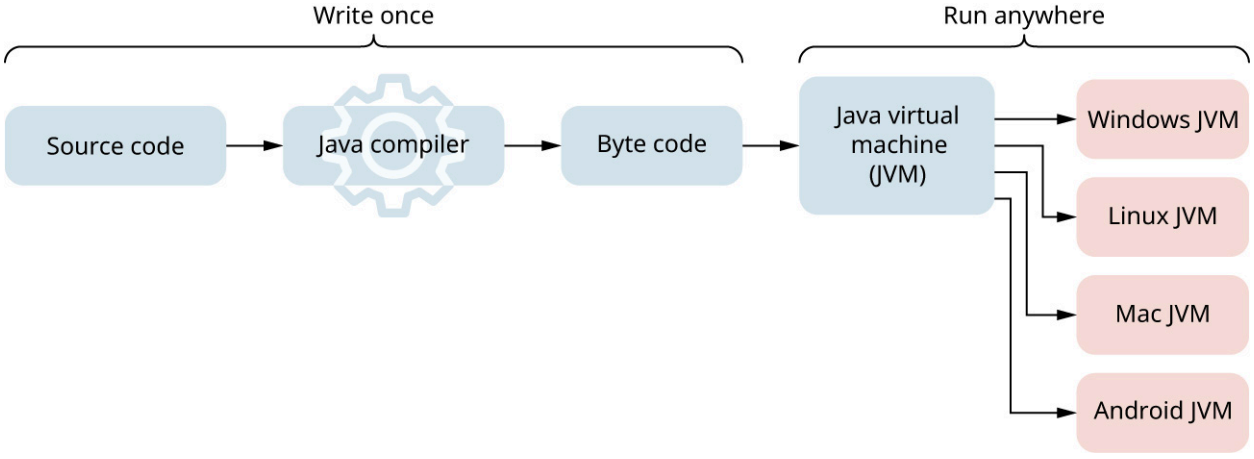


Figure 7.25 The Java compiler generates byte code from source code. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

This can then be run in any environment for which a JVM has been built.

Compilation of Interpreted Languages

In some languages, compilers are present, but they aren't pure and only perform a selective compilation of pieces, or pre-processing, of the remaining source. Occasionally the compiler generates code that builds expectations around decisions that won't be completed until runtime. If these assumptions are sound, the

code runs very fast. If not, a dynamic check reverts to the interpreter.

Dynamic and Just-in-Time Compilation

Sometimes a program delays compilation until the last minute. In Java, a bytecode is a set of instructions for a virtual machine. The process of **just-in-time (JIT) compilation** is when **intermediate code**, which is the language translation phase that produces a code level between the source code and the object code, gets its final compilation (or usually interpretation) phase right at the start of runtime. Bytecode is the standard format for distribution of Java programs to any runtime platform such as Windows, macOS, and Linux. The bytecode is interpreted by a Java virtual machine (JVM) that has been implemented by the producer specifically for their platform. Since there is a JVM for macOS but not for Mac IOS, Java is incompatible with iPhones.

Assembly

Most compilers generate assembly language that must subsequently be processed by an assembler to create an object file. The process of converting a low-level assembly language of the compiler into machine language that the computer can execute in binary form is called **assembly**. Assembly language is a very low-level language that has a strong correspondence with the machine language but is still humanly readable, as visible in Figure 4.7. Post-compilation assembly has some distinct advantages, such as expediting debugging so that the language is easier to read while making sharing possible among compilers. It also isolates the compiler from changes needed in the format of machine language files; for example, when computer chips change, only the assembler must be changed.

A computer's hardware does not implement the assembly-level instruction set; it is run by the interpreter. The interpreter is written in low-level instructions (microcode or firmware). These items are housed in read-only memory and executed by the hardware.

Linking

Language implementations that are intended for the construction of large programs support **separate compilation** where pieces of the program can be compiled and built independently. Once compilation is complete, these pieces called compilation units are rejoined by a linker.

An important function of the linker is to join modules of precompiled libraries of subroutines, such as those in the language API or precompiled user defined functions, into the final program so that they can be separately compiled. If the program draws from a library, it will still need to be linked. Additionally, a **static linker** produces an executable object file by completing its tasks prior to the program running, while a **dynamic linker** allows the program to be carried over to memory for execution after it runs.

Runtime Program Management

The runtime system refers to the set of libraries that the language implementation depends upon for correct operation. The compiler generates program-specific **metadata**, data about data, that the runtime must inspect to do its job. It is recommended that the compiler and runtime system be developed together to avoid some of the following issues:

- Garbage collection: the ability to dynamically destroy program objects when no longer needed and recover their memory
- Variable numbers of arguments: for languages that allow differing numbers and types for function arguments only encountered at runtime; languages support function overriding, such as Java, C++, and C#
- Exception handling: recovery implemented when a program encounters a runtime error or exception
- Event handling: the ability to respond to runtime occurrences such as a button click or other unpredictable event
- Coroutine and thread implementation: languages where implement can handle concurrency and parallelism

Virtual Machines

A virtual machine (VM) provides a complete program execution environment that is the reproduction of a computer architecture. Many modern languages employ virtual machines as their runtime environment. In general computer science, VMs can refer to a system VM as supplied by such vendors as VMware, which provide all the hardware facilities needed to run a standard OS so that programs built for one OS can run on another.

However, with HLLs we make another distinct type of VM called a process VM which provides the environment needed by a single user-level process. Perhaps the first and best example is the Java virtual machine (JVM) because its purpose was to elegantly and efficiently meet a main design goal of the language, which was hardware and OS independence.

The JVM is a complete runtime manager and interpreter. As we have learned, the Java compiler generates as output an intermediate form of code known as bytecode which is placed in object files with a .class extension. This bytecode is identical for every platform and OS on which Java may be run. The JVM is designed to take the last step of interpreting the bytecode into the machine language and the OS instruction set of the target architecture. It is also the runtime manager for the process. A more recent invention is the Java JIT compiler, which may call for more speed than a JVM interpreter can produce. Figure 7.26 illustrates these principles.

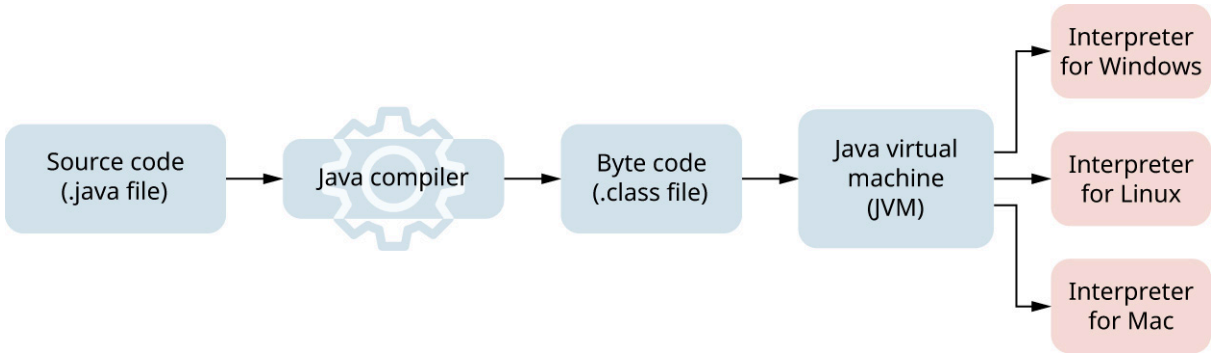


Figure 7.26 This shows the translation of a code snippet of assembly language into binary machine language. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

A JVM manages the runtime as well as provides storage management functionality that embraces features, including class structures (metadata), heaps, register sets, thread stacks, and code storage.

The Microsoft CLR is a similar animal but directed for the Microsoft .NET platform and all its component languages. CLR and JVM share features including multithreading and stack-based VMs, and both have garbage collection. Additionally, both use platform-independence, are self-descriptive, and contain bytecode notation.

INDUSTRY SPOTLIGHT

Using Proven HLLs

While runtime program management using virtual machines provides large flexibility gains, many industries that rely on large enterprise-level software to support demanding mission-critical business applications typically rely on more traditional runtime environments and older, proven HLLs. This is particularly true in the financial industry today. A good example is the use of Python as a core language (https://openstax.org/r/76PythonCorLang) for J.P. Morgan's Athena program and Bank of America's Quartz program. Why do you think this is the case?

Symbolic Debugging

While debuggers are present in most virtual machines and integrated program development environments

(IDEs) and provided by programming language interpreters, they also come as standalone tools. Symbolic debuggers understand high-level syntax. In short, a debugger finds errors in programs. It can also set parameters for stopping an execution should it reach a certain point in the source code or if a constant is read.

Code Optimization and Improvement

The goal of code optimization is to rework parts of your code for efficiency. Some of the problem areas of optimization are redundant computations, as well as inefficient use of the hardware registers, multiple functional units (input, output), memory, and cache.

We can interpret the results of optimization to mean fast and/or decreasing memory requirements. Optimization is best carried out at multiple levels, one of which is at the **basic block** level. A basic block is a minimal length sequence of instructions that will always execute in its entirety if it executes at all, equivalent to a code block. Code improvement at the level of basic blocks is known as **local optimization**, consisting of eliminating redundant operations (unnecessary loads, common subexpression calculations), providing effective instruction scheduling, and providing register allocation.

At higher levels, compilers review and analyze all subroutines for further improvements. The code improvements at multiple layers that include loop performance improvement, register allocation, and instruction scheduling is called **global optimization**.

THINK IT THROUGH

Understanding Optimization

The medical industry views code optimization as a must-have. Most medical operations make use of patient portals for communication between patients and the medical practice. This requires both speed and platform portability such as Windows and Android. Because of this, Java is the language of choice (https://openstax.org/r/76LangofChoice) for these applications. Why is it important for programmers to understand code optimization techniques? Should they only rely on the optimizing compiler provided for the HLL they use? Why or why not? Will you become a better programmer by understanding the way that lower-level code works in order to have a high-level view of optimization?

As with code optimization, code improvement also aims to increase execution speed. Code improvement focuses on eliminating basic blocks redundancy, loop improvements, and instruction scheduling. The biggest area of improvement focuses on the review and enhancement of the behavior of loops. Reordering loops can be difficult but rewarding as all data dependencies must be respected (loop-carried dependencies).

The conclusion that one can form based on code optimization and code improvement is that the “need for speed” is paramount in today’s global software and internet environments.

CONCEPTS IN PRACTICE

Optimizing Your Code

Did you know that compilers can often improve the performance of your code? This process is called optimization. Optimizers can make your code run faster by using techniques such as removing unnecessary calculations, improving memory usage, and reorganizing code for better efficiency.

By understanding optimization techniques, you can write smoother code and use fewer resources. As a side note, mainstream programming language compilers provided by Microsoft and Oracle are designed to optimize the execution of programs. Microsoft’s .Net framework uses virtual machines that manage the CLR

of various programming languages. The intermediate byte code generated by compilers that target these virtual machines is optimized using some of the techniques described in this chapter.

Chapter Review

Key Terms

abstract method declared without a code block for implementation; the stubs of methods that will be implemented by a class which uses the interface

abstraction way of thinking and expressing algorithms to indicate what the programmer wants the hardware to do

access modifier keyword that denotes where in code to access a variable or method

argument value of a function call, which must match the parameters in both number and data type

array named variable that references a block of contiguous memory locations

array initializer values separated by commas placed between curly braces

assembly process of converting a low-level assembly language product of the compiler into machine language that the computer can execute; binary form

assignment statement operation by which a value is stored in a variable

back end compiler step that is responsible for code generation or production of the target code

basic block minimal-length sequence of instructions that will always execute in its entirety if it executes at all, equivalent to a code block

best practice most accepted style and structure of code that can be used to ensure proper software development, which makes it possible to learn new languages easily once a programmer has mastered a given one

binary operator operator surrounded by two operands

Boolean data types that can hold the values true or false

bottom-tested loop condition for further loop iteration set at the exit from the code block

busy waiting condition of a process that is continually looking to see if it has access to a critical section, taking processor time from all processes

bytecode object code produced by Java compilation, which is then interpreted by the Java virtual machine

call stack execution stack that is the data structure that controls the calling and return of functions

case-sensitive refers to whether uppercase and lowercase letters are treated as distinct from each other

class code blueprint from which objects are constructed

client-side script script that runs on a client computer, usually under the control of a web browser and needs an interpreter on the client's machine

code block statement that consists of one or more statements that are structured in a sequential group and delineated as such

code generation transformation of the code to the target or object language

coercion ability of a variable of a data type to be forced to hold a value of a different data type

combined assignment when the assignment operator is combined with another operator, usually mathematical, as a shortcut notation

comment container used to hold documentation in code

Common Gateway Interface (CGI) executable program residing in a special directory known to the web server management program

Common Language Runtime (CLR) takes managed code and provides a JIT execution that allows all the languages to play nicely with each other

compilation single process that takes the source code that the programmer creates and transforms it into another language

complex data type data type consisting of multiple primitive types used as its building blocks

condition-controlled loop that will continue to iterate until a specific condition is met

conditional expression (also: *Boolean expression*) evaluates to the Boolean values of true or false

constructor specialized method that is called upon to instantiate an object from a class

count-controlled loop that will continue to iterate a specific number of times

decrement operator (--) unary operator that lowers the value of its operand by one
destructor specialized methods used in some languages to destroy an instantiated object and recover its memory
determinative loop that is predictable in its number of iterations
dynamic linker compilation process which allows the program to be carried over to memory for execution after it runs
dynamic method binding all methods are resolved dynamically at runtime, not by the compiler, sometimes referred to as late binding
element each "shelf" of an array that can hold a value of a data type
embedded script script in which one language is embedded inside another
encapsulation attributes and behaviors of classes and objects are self-contained and go along with them throughout their lifetime
event-driven when behavior of programs is controlled by actions (events) which are listened for and then acted upon or handled
exception unusual behavior that can be recovered using exception handling support the programming language provides
exponentiation operator (**) operator that raises the value of one operand to the power of the second operand
expression construct in a programming language that evaluates two values
first-class function function that can be assigned as a value to a variable
floating point data type consisting of a decimal number
flow of control order in which, or if, the statements of a program are executed
for loop particular loop construct that is top-tested and count-controlled
formal parameter represent values that the function needs to receive to do its job
front end compilation step that is responsible for the analysis of the source code
function code sequence that is designed to perform a particular task
function call code which invokes the function and passes to it values that may be needed for it to do its job
function signature defines a function and informs the compiler or interpreter about the details that it needs to call upon that function and what it may return
functional programming programming model in which the key concept is that programs are constructed by composing functions and applying them
garbage collection process used to destroy unneeded objects and recover memory when there are no longer any references to them in the code
global optimization code improvements at multiple layers that include loop performance improvement, register allocation, and instruction scheduling
high-order function function that can take one or more first class functions as arguments
hybrid implementation method of language translation which involves the use of both a compiler and an interpreter
identifier name given to a variable or other construct
imperative language emphasizes a "tell the computer what to do" approach
increment operator (++) unary operator that raises the value of its operand by one
indexed array array in which the elements are numbered with an index, starting at zero
information hiding masking of the actual bits and bytes of implementation that the user of a module does not need to know
inheritance feature of OOP in which classes and objects take on specified attributes from their ancestors
initialization assigning its first value to a variable
instantiation process of building an instance (an object) of a class
integer data type consisting of whole numbers
interface construct which resembles a class definition but contains only methods which are constructed as abstract methods

intermediate code language translation phase that produces a code level between the source code and the object code
intermediate form (IF) middle end compilation product created after semantic analysis if the program passes all checks
intermediate language generated from programming source code, but the CPU typically cannot execute directly
iteration looping, going around and re-executing sequences of statements
Java virtual machine (JVM) Java interpreter that translates bytecode into executable code
just-in-time (JIT) compilation process when intermediate code gets its final compilation (or usually interpretation) phase right at the start of runtime
just-in-time (JIT) translation process in which intermediate language is translated and executed exactly when needed
keyword word reserved by the language that has a special meaning
late binding when all methods are resolved dynamically at runtime, not by the compiler; sometimes referred to as dynamic binding
lexical analysis compilation step that converts the source code strings of characters into tokens
literal value of one of the legal data types of an HLL that can be written directly into the code
local optimization elimination of redundant operations in a basic block
lvalue left-hand operand in an expression
managed code code targeted to run under a particular runtime environment
member attribute or method that is encapsulated within a class or object
metadata data about data
middle end compilation step responsible for performing optimizations of the code
modularization splitting a large job into independent units which may be then called upon to perform tasks
module component of a program
modulo operator (%) operator which evaluates to the remainder left after division
multiple inheritance ability to inherit attributes and methods from more than one parent
multitask ability of an application to process more than one task at the same time or to engage in concurrency
mutable something that is changeable, possibly referring to state
name-value pair construct-like variables that is named and can hold values
named constant container that can be assigned the value of a data type which may not be changed thereafter
non determinative loop for which we cannot predict the number of iterations (condition controlled)
object code machine code produced by a compiler or interpreter
operator performs various types of operations (processes) on values
optimization producing what will be needed to generate the fastest and most efficiently running code possible
override define another method with the same name and modify its signature
parallelism when programs can have more than one part of the code running simultaneously
parsing front-end compilation process, also referred to as syntax analysis, or reading the code to make sure it conforms with the syntax rules of the language
pass by reference actual memory address of a variable passed as an argument, meaning the function has access to modify the original variable
pass by value copy made of a value and passed as an argument, meaning the function has no access to modify the original variable
polymorphism feature of OOP in which methods inherited may perform in different ways in different subclasses and depend upon their context
precedence order of operations rules that determine the order in which operators in statements are evaluated

preprocessing pre-compilation step in which the source code is manipulated to prepare it for the compilation stage
preprocessor directive code that describes resources the program needs in order to compile, usually files to insert into the source code
primitive data type (also: *basic data type*) simplest data type of a language that can usually be represented directly by the hardware memory and registers
procedural language allows programmers to group statements into blocks of code within the scope of which variables may be defined and manipulated independently from the rest of a program
public interface documentation of only what a user needs to access an object, its attributes, and its methods
pure function function that will return the same result every time given the same arguments
race condition when the condition of a program and its behaviors are not synchronized
readability measures how easily an HLL can be read and understood
reference variable variable that holds memory addresses
return command that ends a function and/or sends values back to the place at which the function was called
runtime error exception that is serious enough that it cannot or should not be handled by the software
rvalue right-hand operand in an expression
scope defines both the visibility of variables to locations in the code and the physical lifetime of variables
scripting language characterized by placing a list of code statements into a file, referred to as a script
selection making a decision on which path of execution to follow
semantic analysis compilation step that discovers the meaning of the code
sentinel expression that sets the condition at the entry or exit of a loop for continued iteration
separate compilation situation where pieces of the program can be compiled and independently built
sequential execution executing statements in the order in which they appear
server-side script script that runs on a server in a web designer's domain
shared data data available to multiple program locations and/or scopes
shell scripting stringing together a group of commands to perform tasks on the user interfaces of various operating systems
short circuiting when HLL expressions that use the logical operators do not need the second operand evaluated for results to be known
single inheritance methodology used by most modern OOP HLLs in which classes and objects can only inherit from one parent
stack frame structure representing a function and its parts that is placed on the call stack
stack overflow condition in which the call stack has run out of memory in which to expand
starvation condition of a process when a process must wait to enter a critical section, but the other processes monopolize the section, and the waiting process does not get processor time
state data that are remembered over time
static linker produces an executable object file by completing its tasks prior to the program running
string data type representing a sequence of characters
strongly typed language in which a variable may only contain one of the language's defined data types for its entire existence
Structured Query Language (SQL) most widely used language to specify a query that a database system can process to store or retrieve data
subclass child class in an inheritance relationship
superclass parent class in an inheritance relationship
syntax analysis front-end compilation process usually referred to as parsing or reading the code to make sure it conforms with the syntax rules of the language
ternary operator acts on three operands
thread of control thread the processor is controlling, which is actually executing at the time
tokens front-end compilation product that are sequences of compiler recognizable atomic symbols which make it much easier for the next steps to recognize them

top-tested loop when a condition for further loop iteration is set at the entrance to the code block
truth table chart that shows what the resulting value would be given every combination of values of operands in an expression using a logical operator
type cast operation that coerces an assignment of a possibly incompatible data type to a variable
unary operator acts on one operand
user interface (UI) point at which human users interact with a computer, website or application
variable gives a name to a memory location that is used in any HLL to hold a value
variable declaration consists of a statement which specifies the variable name and data type
void a keyword in many HLLs that denotes a null value stored in a variable or that a function returns no value
weakly typed when a variable may at different times hold values of any of the language data types
writability measures how easily an HLL can be used to create and modify programs

Summary

7.1 Programming Language Foundations

- HLLs give us the ability to make use of abstraction to build algorithms in a close-to-English representation.
- There are criteria that are used for selecting the proper HLL for the required tasks. These can include scalability, cost, flexibility, efficiency, portability, and maintainability.
- HLLs are designed to fulfill a purpose, which could be general purpose, web oriented, object-oriented, or parallel-programming oriented, among others.
- HLLs are shipped with application programming interfaces (APIs) that contain useful tools and objects to help program them for their purpose.
- HLLs have different implementation approaches. These vary from compiled to interpreted to hybrid.
- IDEs are structured environments containing tools that are targeted to programming different HLLs.

7.2 Programming Language Constructs

- HLLs all have data types which are either primitive or complex that form the domain of legal data types they recognize.
- HLLs all have operators that represent the legal set of operations that may be performed on the data types such as addition or assignment.
- HLLs support variables named containers that may hold the legal values of the HLL data types and literals, which are a plain language representation of those values.
- HLLs have rules for identifiers that are user-defined names of language elements such as variables, constants, and functions.
- HLLs allow for documentation, usually as comments, so that programmers know about a program.
- HLLs provide data structures, such as arrays, which are containers to store multiple values.
- HLLs have constructs for flow of control which dictate the path of execution of the code, usually conditional statements and iteration constructs.
- HLLs provide the means to modularize, usually in the form of functions.
- A call stack or execution stack is the data structure that controls the execution of a program.
- Most HLLs provide the means for handling and recovering from runtime errors, a process called exception handling.
- HLLs can handle both user interactive and file input and output.

7.3 Alternative Programming Models

- Functional programming is a declarative language paradigm where programs are constructed by composing functions and applying them; some of whose features are implemented in imperative languages.
- Object-oriented programming (OOP) is a paradigm based on classes and objects and is widely implemented in many HLLs.
- The degree of support of encapsulation, inheritance, and polymorphism is indicative of the degree of object-orientation of an HLL.

- Concurrency is the behavior of an HLL when it can multitask sections of its own code, giving the illusion of simultaneous execution.
- Parallelism is the behavior of an HLL when it can simultaneously execute sections of its own code, requiring multiple cores or processors.
- Scripting languages are HLLs that are usually interpreted but retain many of the features of compiled languages and are often used for web programming.

7.4 Programming Language Implementation

- Compilers are the general tool used to implement the process of translating source code to another language that is either machine language or is closer to assembly language.
- Interpreters follow the same processes as compilers with differences in the timing of the translation phases.
- The compilation process is divided into distinct stages: a front end (code analysis), an optional middle end (code optimization), and a back end (code generation). These tasks may be done in just two stages: front end and back end.
- Many compilers generate assembly language either as their output (needs separate assembly) or as the output of one stage (assembly performed as part of compilation).
- A linker or link-loader is used to stitch together pieces of separately compiled code for final execution.
- Runtime management is handled by a runtime system which is highly aware of the functionality of the compiler as to enable the use of features like garbage collection, exception handling, and concurrency or parallelism.
- Runtime management may be handled by virtual machines which provide execution environments that are emulations of the computer architecture.
- Code optimization and code improvement are highly desirable features of program implementation.



Review Questions

1. Which HLL was developed first?
   a. COBOL
   b. C
   c. Java
   d. Fortran

2. What is the term used to describe the most accepted style and structure of code that can be used to ensure proper software development?
   a. paradigm
   b. best practice
   c. model
   d. abstraction

3. Which abbreviation refers to a library shipped with the language which contains objects and useful functions for the tasks for which the language is purposed?
   a. UX
   b. GUI
   c. API
   d. UI

4. Why are various HLLs related to each other and/or part of the same history?

5. What is event-driven programming?

6. What are the names of some different programming paradigms?

7. What is the difference between a compiled and an interpreted HLL?

8. What are the typical constituent parts of an HLL development environment or IDE?

9. What is weak typing? What is strong typing? What is the difference between them?

10. What are the equivalents of dynamic and static typing?

11. What is an example of a data type that is always primitive?
- **a.** string
- **b.** array
- **c.** number
- **d.** object

12. What best defines a function interface?
- **a.** parameter
- **b.** return
- **c.** code block
- **d.** signature

13. What type of operator has only a single operand?
- **a.** logical
- **b.** relational
- **c.** unary
- **d.** Boolean

14. What is "short circuiting" in the context of HLL expressions evaluation?

15. What is the difference between variable passing by value versus passing by references?

16. What is the difference between integer data types and floating-point data types?

17. What are composite types?

18. What is the difference between subroutines and modules?

19. What term represents a feature in OOP?
- **a.** synchronization
- **b.** polymorphism
- **c.** mutual exclusion
- **d.** parallelism

20. What is the term for a process used to return memory that is no longer needed?
- **a.** instantiation
- **b.** polymorphism
- **c.** starvation
- **d.** garbage collection

21. What is a high-order function in a functional HLL?

22. What is typically the last statement of a recursive function?

23. Give the definition of a thread.

24. Explain encapsulation, inheritance, and polymorphism in object-oriented HLLs.

25. What is the difference between concurrency and parallelism?

26. Explain mechanisms that may be used to enable synchronization in concurrent programming.

27. What types of problems are best solved by scripting HLLs?

28. What is the name for software that scans and translates the source code one line at a time?
- **a.** interpreter
- **b.** compiler
- **c.** assembler
- **d.** linker

29. The middle end stage of compilation is responsible for what task?
- **a.** lexical analysis
- **b.** optimization
- **c.** code generation
- **d.** assembly

30. Software that provides a complete program execution environment is the reproduction of which computer architecture?
- **a.** dynamic linker
- **b.** parser
- **c.** virtual machine
- **d.** static linker

31. What is the difference between the front end, middle end, and back end compiler phases?

32. Explain the different phases of compilation.

33. What is the difference between static and dynamic linking?

34. Explain late binding.

35. Explain the different phases of code improvement.

36. Why is there such a focus on loops improvement as part of code optimization?

Conceptual Questions

1. Should all languages be kept simple, small, and straightforward to avoid dangerous complexity? Support your opinion.

2. Are HLLs purpose-driven? Give some examples of purpose-driven HLLs, expanding on the purpose of each.

3. Do you think that all high-level programming languages support abstraction? Give a few examples of languages that do, and list abstraction mechanisms that are used in each.

4. While a similar set of programming language constructs are applicable to all modern HLLs, can you explain why modern HLLs such as C, Java, Python, C++, C#, JavaScript, and PHP are so different?

5. Explain how HLLs that support specific programming models can be classified as imperative or declarative.

6. Why is code optimization sometimes specific to a given HLL?

Practice Exercises

1. Research the data types of Java and C++. Write a summary outlining the differences and similarities between the two.

2. Explain the differences between pure compilation, pure interpretation, and hybrids of the two. Give the advantages and disadvantages of each methodology.

3. Programming for iPhone and programming for Android require different purpose-driven HLLs. List the HLLs used for each and summarize some of the differences between them.

4. What are some of the criteria for selecting among various HLLs? Describe these criteria.
5. Describe the purpose of an API and explain how the use of APIs can lead to faster development, increased readability, and less complexity.
6. The Python language controversially uses an indentation of a uniform number of spaces to construct code blocks in control statements and other structures. What are the arguments and pros and cons for using this technique instead of the usual curly braces? To illustrate your answer, please write a short program snippet that demonstrates the pros and cons of indentation (in Java) as compared to curly braces (in Python).
7. Write a conditional statement to determine if a variable is within a specific numerical boundary, and if so, multiply the number by 2, and if not, multiply the number by itself.
8. Research what Fibonacci numbers are. Write a recursive program that finds the *N*th number of the Fibonacci sequence.
9. Interpret the following program snippet. What will the function return?

```
int main() {
   int x = 1;
   int y = 7;

   if (x == 1 && y > 10)
   {
       return 0;
   }
   else if (x < 0 || y == x)
   {
       return 1;
   }
   else if (x != 0 && y == x)
   {
       return 2;
   }
   else if (x > 0 && (x != y || y > x))
   {
       return 3;
   }
   else
   {
       return 4;
   }
}
```

10. Write a do while loop that will execute five times.
11. Encapsulation is one of the three fundamental components of object-oriented programming. Write a summary which defines the term and explains the advantages of it. Please provide a code snippet that demonstrates poor usage of encapsulation and improve it to highlight the advantages of proper encapsulation.
12. Write a short program in JavaScript that asks the user to input the base and height of a triangle. Calculate

the area of the triangle and print it back out.

13. One of the key features of OOP is inheritance. Research multiple inheritance and how Java and C++ differ when it comes to multiple inheritance. Why would restricting multiple inheritance be a good idea?
14. Research the synchronized keyword in Java. Implement a critical section of code in Java by using the synchronized keyword.
15. The back end of a conventional compiler differs from that of a JIT compiler. List the ways that they are different. Does their design add to their differences?
16. Implement a code block in C++ that is only "activated" when we are debugging the application using a preprocessor definition.
17. Run the program you wrote above in GDB. Explore setting breakpoints and setting watches on variables as they change. You can either download and install GDB on your computer or use the free online website (https://openstax.org/r/76gdb) that runs GDB for you.

Problem Set A

1. Research how high-level languages have portability restrictions and how some languages handle portability. For example, compare C# and Java on Linux platforms.
2. What is the tradeoff between C# being fully compiled and Java running on the JVM?
3. Research for loops. Write a for loop that matches the functionality of the following while loop:

```
int main () {
    int iter = 0;
    while(iter < 10)
    {
        iter++;
        printf("%d", iter);
    }
}
```

4. In C/C++, data types can be returned from a function or used as an output parameter by passing the object by reference using a pointer. Write a piece of code that demonstrates two functions with the same performance, but one that returns a data type and one that uses an output parameter.
5. Research what a style guide and coding standard are. Review a sample coding standard for at least three different languages. Are the style guides and coding standards the same for each language? How could the style guides and coding standards lead to well-structured code?
6. Describe how inheritance can simplify the design of a software application. Can inheritance add to the complexity of the software?
7. You are designing a real-time software application, so timing of computations is critical. How could you develop a solution that takes less time to execute? What safety considerations do you need to design around?
8. Research preprocessor definition uses. How can an organization have a common baseline of code but deliver only certain modules of code in a delivery.

Problem Set B

1. Research how much support and how many APIs are available for cutting-edge technologies for the most

popular languages of C#, C++/C, Java, and Python.

2. You're the lead software engineer for a team that has two projects on going with the same set of user capability requirements. Since the requirements are the same, you want to find a common solution and develop at least a common module. The client for Project 1 has agreed to a REST API as the standard to interface to the application, while the client for Project 2 will integrate the software as a static library. Project 2 requires that the software needs to run on both Windows and Linux, and C must be the language of choice. Project 1 only needs to run on Windows and has no language constraints. Research C and the level of support for REST API development. Would C be a good choice for the top layer for Project 1? Would Java be a better solution?
3. We have demonstrated that one of C and C++'s features is that you can return a data type and can also use an output parameter to get the output of a function by sending in a pointer to the variable or data structure. Research and explain the pros and cons of using an output parameter by sending in a pointer to the data type.
4. Research the concept of garbage collection in programming languages. How do Java and C++ differ in garbage collection?
5. Research how arrays are different between C/C++ and Java in terms of memory allocation. How can this impact safety and reliability of the code?
6. If functional languages such as C cannot build objects using classes, how can a developer create complex data types? Research how structures are used in functional languages to create data types.
7. Write three Java classes using polymorphism where there is one super class and two classes that inherit from the super class.
8. Research how compilers optimize code at build time and how this is a configurable setting.



Thought Provokers

1. Consider TechWorks, the startup company committed to leveraging innovative technologies as a business growth facilitator. Describe how it can best use programming in various HLLs to create innovative software that can administer and generate business. Give some precise examples and explain how the startup would be able to benefit from HLL programming from a business standpoint (i.e., keep sustaining the cost of doing business while increasing its number of customers).
2. Consider our startup company committed to leveraging innovative technologies as a business growth facilitator. Describe how it can best use the Java or C# programming language constructs to create specific products or services that can generate business. Give precise examples and explain how the startup would be able to scale the resulting business (i.e., keep sustaining the cost of doing business while increasing its number of customers).
3. Consider our startup company committed to leveraging innovative technologies as a business growth facilitator. Describe how it can best use hybrid HLLs to implement products or services in a business context. Give precise examples and explain how the startup would be able to scale the approach and save costs while keeping its customers happy.
4. Consider our startup company committed to leveraging innovative technologies as a business growth facilitator. Describe how it can best use programming language implementation to create a repertoire of related services as well as a repository of code samples that may be used to generate business. Give precise examples and explain how the startup would be able to scale the resulting business (i.e., sustain the cost of doing business while increasing its number of customers).

Labs

1. Work through the "Hello, World!" Java tutorial (https://openstax.org/r/76JavaTutorial) to learn how to program Java to write a sentence to the screen. Start Exercise to cross reference the tutorial with the program. Find the online IDE Replit (https://openstax.org/r/76Replit) and establish an account. Research how to write "Hello World" to the screen in C++ and Python. Enter the code in Replit and write a report on your experiment pointing out some of the key concepts that you learned.
2. C++ has the concept of libraries of container classes, and both Java and C# have similar libraries of collections classes. These contain various types of data structures. We have examined the array; however, the array has limitations. Learn more about the container and collections classes, and look for examples of the patterns of data they are designed to represent and support. Contrast the various offerings with the plain array. Catalog the similarities and differences between the concepts in these three languages. Which of these libraries do you prefer? Why?
3. Study the most popular API library packages which enable multithreading for the languages of C, C++, and Java. In each of these languages, can you describe how multithreading works? Out of the routines you've learned so far, which of them are safe for multithreading? Which are not? Can all routines be made thread-safe?
4. WebAssembly (https://openstax.org/r/76WebAssembly) is a new assembly programming language intended to execute high-performance code in the browser. Investigate the language features, and write a summary of the reasons for its use and the possible benefits and disadvantages for employing it for a simple web application.



8

Data Management

Figure 8.1 Big data taxonomy includes data providers, data consumers, data owners, and data viewers while providing the data flow between them. (credit: modification of "DARPA Big Data" by Defense Advanced Research Projects Agency (DARPA)/Wikimedia Commons, Public Domain)

Chapter Outline

8.1 Data Management Focus
8.2 Data Management Systems
8.3 Relational Database Management Systems
8.4 Nonrelational Database Management Systems
8.5 Data Warehousing, Data Lakes, and Business Intelligence
8.6 Data Management for Shallow and Deep Learning Applications
8.7 Informatics and Data Management

Introduction

Managing data today requires an end-to-end perspective and the ability to combine the use of various types of data. For example, Amazon needs to run its day-to-day businesses and sell products to customers, handle returns, pay commissions to retailers and wholesalers, and develop and price new products. This is all part of day-to-day operations, and most retail companies strive to achieve operational excellence as it is a key driver of customer satisfaction. For that purpose, retailers typically use traditional relational databases and structured data to run their operations. At the same time, retailers need to remain competitive and develop with pricing strategies that attract and retain customers. Doing so requires being able to analyze market prices accordingly before advertising products to their customers. For that part, businesses generally rely on datasets and big data analytics to predict competitive prices in order to optimize sales. Overall, businesses and organizations that deliver products to customers need to leverage insights coming from metadata to transform as they perform and remain competitive while sustaining their operations. It is no longer possible to rely on operational excellence to ensure continued success.

In this chapter, we will cover the spectrum of data management activities, including the storage and retrieval of various types of data. In other words, the chapter is about how big data are managed today from an end-to-end standpoint.

As an example, TechWorks is a start-up company that is 100% committed to leveraging innovative technologies as part of its repeatable business model and as a business growth facilitator. TechWorks has multiple departments including human resources, finance, sales, marketing, operation management, and information technology. Each department has its own information system and database; they do not share any resources. TechWorks has many success stories in the market. However, they have a huge problem with integrating their reports to aid in decision-making. TechWorks has many challenges that impact their competitiveness and survival in the market, including their traditional database not working properly; their practices in collecting, managing, and analyzing data not promoting better business decision-making; and their servers being old and sometimes unable to sufficiently handle the company data. The chief executive officer (CEO) of TechWorks decides to develop a new database management system with the following objectives:

- Integrate all of the departments' practices into one system.
- Use the cloud to store, manage, analyze, and maintain the data.
- Hire a data scientist, computer scientist, and information architect to work as a team with the database designer and the database administrator.
- Apply the extraction, transformation, and loading process to the data.
- Use business intelligence and machine learning tools to make decisions based on the data.

8.1 Data Management Focus

Learning Objectives

By the end of this section, you will be able to:

- Discuss data management and its relation to computer science and data science
- Understand that data are the backbone of the industry
- Identify and explain key concepts in data management
- Distinguish various roles in the field of data management
- Explain the current state of data management

In the current digital world, collecting information and facts that are stored digitally by a computer, or **data**, is a straightforward process. There are many direct and indirect ways, such as social media, that support the data collection process. A large amount of data is collected every day, every hour, and every minute. But this amount of data is not useful unless benefits can be drawn from it, which requires knowing what to do with these data. In this context *data* are like the "crude oil" that needs to be stored digitally in order to allow extraction of information and knowledge via information and knowledge management systems. Related structured data is stored in a database management system. The knowledge supports the decision-makers in any organization in making important decisions, such as increasing the sales in some regions, changing the suppliers within the supply chain, decreasing the manufacture of a specific product, modifying the hiring process, studying the community culture, and identifying customer demand. It is clear that proper data management is required to support decision-makers. The study of managing data effectively, or **data management**, treats data as a corporate asset similar to physical assets such as plants and equipment. Data management requires having a strategy to analyze the data by collecting data, then storing the data, cleaning the data, preprocessing the data, and preparing the data for analytics, which will lead to decision-making.

CONCEPTS IN PRACTICE

Managing Your Social Media Data

Facebook, Instagram, LinkedIn, Snapchat, Tik Tok, X, and many other social media applications are controlling your data. You are most likely posting too much information about your life, work, and school, but it is not only you—everyone is sharing data on social media. A supermassive Mother of all Breaches

(MOAB) publicized discovered in early 2024 contains data from numerous previous breaches, comprising 12 TB of information from LinkedIn, X (formerly Twitter), Weibo, Tencent, and other platforms' user data.

Have you ever asked yourself how service providers are managing this data? How do they store the data? Where do they store it? Which database are they are using? And the big question: can we benefit from this data and manage our own data? Meta is offering the option to control your Facebook experience by managing your data; all you need to do is complete a form, which gives you multiple options such as downloading your data, managing ad preferences, and removing a tag from a post.[1]

Data Management in Computer and Data Science

Computer science is the study of computing algorithms that receive data as inputs, process data, and produce outputs. Computer science applies the principles of mathematics, physics, and engineering. Hardware, software, operating systems, and applications are working together to perform computer science computations.

Data science is the study of extracting knowledge from information using hypothesis analysis and algorithms. Information is the result of processing raw data. Figure 8.2 illustrates the skills and knowledge needed to be successful in data science.

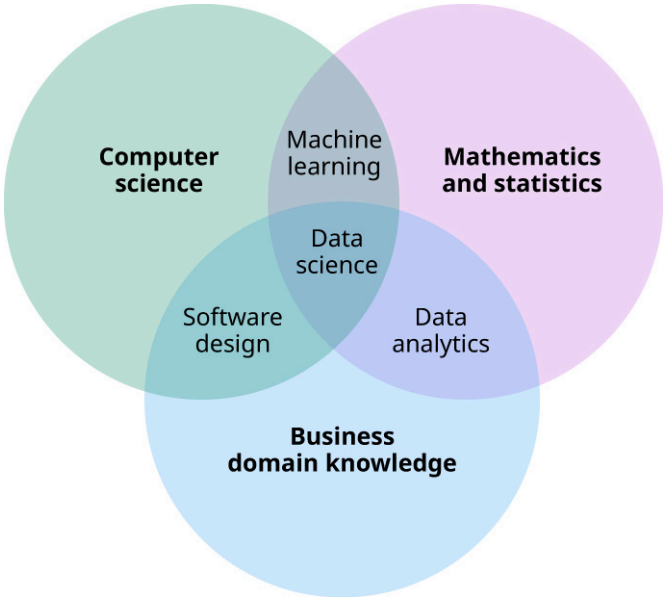


Figure 8.2 Data science combines programming skills, analytic systems, statistics, and data mining. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Data Management in the Industry

Data are the backbone of any enterprise because analyzing data improves performance and increases profit. Gathering raw data from various sources and processing it for modeling is about 80% of the data scientist's work.[2] Enterprise projects usually involve a massive amount of data, which makes it impossible to be stored locally, so many businesses move their data to the cloud. A business may also choose to run applications as well as databases and operating systems in the cloud. Data management is often handled by a separate data engineering team because most computer and data scientists do not have enough knowledge about data storage and infrastructure.

1 R. E. G. Beens, "The privacy mindset of the EU vs. the US," *Forbes*. Updated April 14, 2022. https://www.forbes.com/sites/forbestechcouncil/2020/07/29/the-privacy-mindset-of-the-eu-vs-the-us/?sh=215a8a127d01

2 ProjectPro, "Why data preparation is an important part of data science?" Updated April 11, 2024. https://www.projectpro.io/article/why-data-preparation-is-an-important-part-of-data-science/242.

INDUSTRY SPOTLIGHT

Industry Data Management

Data management is important in every industry today. The main benefit of data management is to minimize potential errors by controlling access to data using a set of policies.

Elaborate on how useful it is to know about data management in retail industries for product marketing purposes. (*Hint*: Data management can help with targeted advertising, for example.)

Data Management Aspects

There are various aspects of effective data management: metadata cataloging, metadata modeling, data quality (data accuracy, data completeness, and data consistency), and data governance, which we will discuss in more detail in the following subsections.

Metadata Cataloging

Metadata are used in a **database management system (DBMS)**, which is a system that creates, stores, and manages a set of databases. In a DBMS approach, metadata are stored in a catalog called a data dictionary. A **data dictionary** is a set of information describing the content, format, and structure of a database (e.g., in a relational DBMS, the catalog includes the names of all available tables and their associated fields). The metadata catalog constitutes the heart of the database system and can be part of a DBMS or a stand-alone component. The metadata cataloging process is collecting data about processes, people, products, and any enterprise-related data, which provides an important source of information for end users, application developers, as well as the DBMS itself. The catalog typically provides an extensible metadata model, import/export facilities, support for maintenance and reuse of metadata, monitoring of integrity rules, facilities for user access, and statistics about the data and its usage for the database administrator and query optimizer. Metadata cataloging improves the user experience, adds competitive advantages, and improves the efficiency of the business.

Metadata Modeling

A business presentation of metadata is **metadata modeling**. A database design process can be used to design a conceptual model of a database storing metadata as an enhanced entity–relationship (EER) model (Figure 8.3) or unified modeling language (UML) model. EER and UML are used to create models for designing large systems. Metadata modeling may cover various views of these models. For example, EER models and UML class diagrams help express views of a data model. While EER focuses purely on data modeling, the UML notation may be used to express comprehensive models of information systems using additional diagrams.

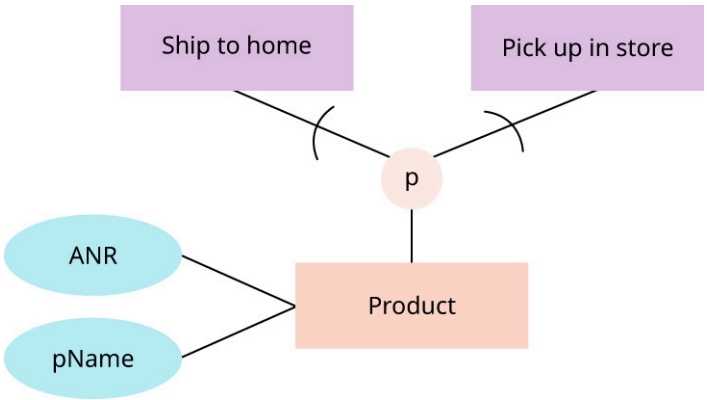


Figure 8.3 This EER diagram models a product entity and states that a product can be either shipped or picked up but not both. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Data Quality

The measure of how well the data represents its purpose or fitness for use is called **data quality (DQ)**. Data of acceptable quality in one decision context may be perceived to be of poor quality in another. Data quality determines the intrinsic value of the data to the business. Businesses may use the concept **garbage in, garbage out (GIGO)**, which means that the quality of output is determined by the quality of the input. Poor DQ impacts organizations in many ways and at different management levels. For example, it negatively impacts operations in day-to-day operations; however, it makes a big difference at the strategic level in making decisions. DQ is a multidimensional concept in which each dimension represents a single aspect such as views, criteria, or measurements. DQ comprises both objective and subjective perspectives and can categorize different dimensions of data quality. The DQ framework has four categories: intrinsic DQ, contextual DQ, representation DQ, and access DQ, as illustrated in Figure 8.4.

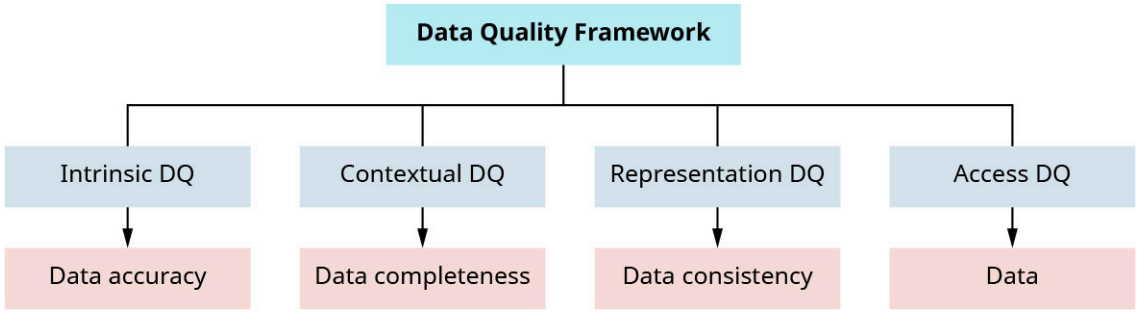


Figure 8.4 The four data quality framework categories are intrinsic DQ, contextual DQ, representation DQ, and access DQ. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

In the intrinsic DQ category, **data accuracy** refers to whether the data values stored for an object are the correct values, and they are often correlated with other DQ dimensions. We can count the data reliability as a part of data accuracy. The degree to which all data in a specific dataset are available with a minimum percentage of missing data is called **data completeness**. It can be viewed from at least three perspectives: schema completeness (the degree to which entity types and attribute types are missing from the schema), column completeness (the degree to which there exist missing values in a column of a table), and population completeness (the degree to which the necessary members of a population are present or not). The **data consistency** dimension is part of the representation category and can also be viewed from several perspectives: consistency of redundant or duplicated data in one table or multiple tables, consistency between two related data elements, and consistency of format for the same data element used in different tables. The accessibility dimension is part of the access category and reflects the ease of retrieving the data from the underlying data sources (this often involves a trade-off with security, which is also part of the access category).

There are many common causes of bad data quality, such as the following:

- computer processing
  - duplication: multiple data sources providing the same data, which may produce duplicates
  - consistency problem: different occurrences of data or incorrect data
  - objectivity problem: data giving different results in every process
  - limited computing resources: insufficient computing resources limiting the accessibility of relevant data
  - accessibility problem: large volumes of stored data making it difficult to access needed information in a reasonable time
- human intervention
  - biased information: using human judgment in data production
  - relevance problem: different processes updating the same data
  - data quality problems: decoupling of data producers and consumers

Data Governance and Compliance

The set of clear roles, policies, and responsibilities that enables a business to manage and safeguard data quality using internally set rules and policies is called **data governance**. The related concept that ensures that data practices align with external legal requirements and industry standards is called **data compliance**. For example, the UK-GDPR (General Data Protection Regulation) is the United Kingdom's data security regulation, modeled after the EU-GDPR that governs and regulates how UK organizations and businesses collect, store, use, and process personal data.

Using data governance, data are managed as an asset rather than a liability. Data governance has three dimensions: technology, people, and process. It should include standard roles for quality, security, and ownership.

In planning for data governance, we should answer four questions: what, how, why, and who. What policies should we include? How do we integrate the policies with the enterprise business process? Why do we need this policy? Who will be part of this policy?

Different frameworks have been introduced for data quality management and data quality improvement: Total Data Quality Management (TDQM), Total Quality Management (TQM), Capability Maturity Model Integration (CMMI), ISO 9000, Control Objectives for Information and Related Technology (COBIT), Data Management Body of Knowledge (DMBOK), Information Technology Infrastructure Library (ITIL), and Six Sigma. These frameworks provide guidelines for organizations that define how a product or process should be based on high-quality standards. The main issue in these frameworks is that they may cause failure when the system is not considering all of the processes in a correct flow.

It is possible to annotate the data with data quality metadata as a short-term solution. Unfortunately, many data governance efforts (if any) are mostly reactive and ad hoc.

GLOBAL ISSUES IN TECHNOLOGY

Hacked!

Chegg is an educational technology company based in California. It offers various online services such as textbook rentals, tutoring, homework assistance, and more to students around the country. In April 2018, an unauthorized party gained access to Chegg's database. This database hosted user information for both Chegg and its affiliated companies (e.g., EasyBib). The hacked information included names, emails, passwords, and addresses, but did not include any financial information or social security numbers. After discovering the breach, Chegg implemented plans to notify the 40 million affected users. The motivation for the attack is still unclear, but it is likely that a third party sought information to profit from identity theft.

What are some things that Chegg users can do to protect themselves from this hack and future breaches of personal information?

Data Management Roles

Within any organization, there are various data management roles including information architect, database designer, data owner, data steward, database administrator, computer scientist, and data scientist. Their roles are outlined in the following sections.

Information Architect

An **information architect**, also known as a *data architect* or *information analyst*, is responsible for designing the conceptual data model (blueprints) to bridge the gap between the business processes and the IT environment.

The information architect collaborates with the database designer who may assist in choosing the type of conceptual data model (e.g., EER or UML) and the database modeling tool. Figure 8.5 shows an example of different database systems that various personnel working in data management may encounter.

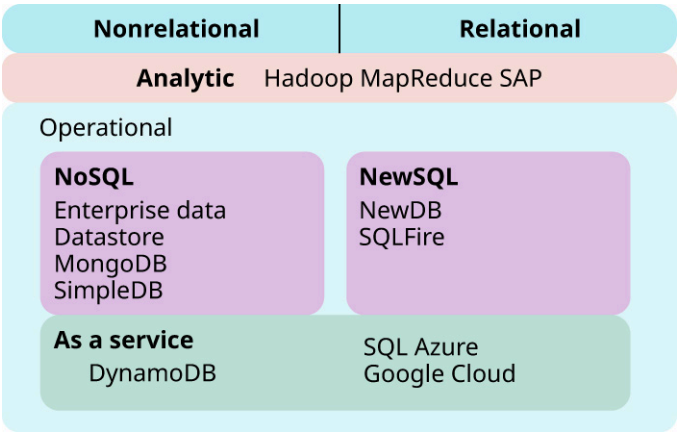


Figure 8.5 Existing database systems can be categorized as nonrelational and relational. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Database Designer

A **database designer** is responsible for creating, implementing, and maintaining the database management system. Other responsibilities include translating the conceptual data model into a logical and internal data model, assisting the application developers in defining the views of the external data model, and defining company-wide uniform naming conventions when creating the various data models.

Data Owner

The **data owner** has the authority to ultimately decide on the access to, and usage of, the data. The data owner could be the original producer of the data, or the data could originate from a third party. A person who assumes the role of a data owner should be able to insert, edit, delete, or update data as well as check or populate the value of a field. The data owner is the one responsible for checking the quality of one or more datasets.

Data Steward

A **data steward** is a DQ expert who ensures that the enterprise's actual business data and the metadata are accurate, accessible, secure, and safe. The data steward performs extensive and regular data quality checks, initiates corrective measures or deeper investigation into root causes of data quality issues, and helps design preventive measures (e.g., modifications to operational information systems, integrity rules).

Database Administrator

The **database administrator (DBA)** is responsible for the implementation and monitoring of the database as well as ensuring databases run efficiently by collaboration with network and system managers.

Computer Scientist

A **computer scientist** is a person who has theoretical and practical knowledge of computer science. The computer scientist will solve problems using technology by applying computer science practices. Computer scientists typically focus on building end-to-end solutions for companies. For example, computer scientists can create software applications that implement complex algorithms and store and retrieve related data into and from database systems. Computer scientists may also be involved in designing and fine tuning complex machine learning algorithms.

Data Scientist

Data scientists typically focus on creating classification and prediction models by training existing machine learning algorithms. The resulting trained models act as programs to help classify new data or predict an outcome based on input data. In general, a **data scientist** is a person who has theoretical and practical knowledge of managing data. A data scientist's background combines computer science, mathematics, and statistics. A person in this role is responsible for gathering a large amount of structured, semistructured, and unstructured data to preprocess and prepare data for advanced data analysis to develop a product or to make a business decision.

Data Management Road Map

The data management road map has multiple steps, starting from collecting and storing data to having a final product or decision. Available databases vary in type and vendor (e.g., SQL, Hadoop, Spark, and MongoDB). Storing the data in databases is not hard, but the entire data management process most often takes place on the cloud, which adds a new set of necessary skills for the computer scientist and the data scientist. Massive data growth transformed the way data are stored and analyzed, and many applications and databases are hosted on servers in data centers elsewhere. While preparing data, data scientists and computer scientists should clean and format the data/information to be used for the correct marketing use. We will discuss database types in more detail together with their data structure implementations (refer to 8.2 Data Management Systems).

TECHNOLOGY IN EVERYDAY LIFE

End-to-End Data Management

End-to-end data management covers the data life cycle within the system. A data life cycle is a process that helps the organization to manage the flow of data, and it includes creating the data, storing the data, and sharing the data.

A global positioning system (GPS) is an embedded system that mainly uses data to provide routes and destinations. Most of us use a GPS to check on road construction, traffic, or to find the shortest route.

How does knowledge of end-to-end data management help people in their everyday life? Look at these data collections from all aspects of data management including collecting, storing, and analyzing these data. Provide a few illustrative scenarios to explain your opinion.

8.2 Data Management Systems

Learning Objectives

By the end of this section, you will be able to:

- Define important terms and characteristics of data management systems
- Explain the various aspects that characterize database management systems

Remember that data are any raw facts you can collect such as the number 48502. When we process the data, we have information that has meaning. For example, the number 48502 is a zip code. When we have a collection of related data, we call it a database. To store, retrieve, edit, and maintain the related data in the database we need a system called a database management system (DBMS).

Definition and Characteristics

Data, data model, database, and DBMS concepts are connected to each other. Data relate to known facts that can be recorded and have an implicit meaning. A **data model** is an abstract model that contains a set of

concepts to describe the structure of a database, the operations for manipulating these structures, and certain constraints that the database should obey. A database is a collection of related data items within a specific business process or problem setting; an example of a large commercial database is the one maintained by amazon.com. A DBMS is the software package used to define, create, use, and maintain a database while considering appropriate security measures. A **miniworld**, or universe of discourse (UoD), represents some aspect of the real-world data that are stored in a database (e.g., student grades at a university). There are many characteristics of the database approach such as being superior to the file approach in terms of efficiency, consistency, and maintenance; providing loose coupling between applications and data as well as facilities for data querying and retrieval; removing redundancy and cleansing the data; supporting multiuser transactions that allow multiple users to access the same data and multiple views of the data; and allowing sharing data between users and applications.

THINK IT THROUGH

University Miniworld

Let us think about the university as a miniworld. In this miniworld, we need to define a data dictionary to describe the following: students, courses, sections, departments, and instructors.

Suggest a miniworld that would be part of a university environment. Define the main entries to be included such as STUDENTs = ID + FirstName + LastName.

Applications of Database Technology

A **database application** is a program or piece of software designed to collect, store, access, retrieve, and manage information efficiently and securely. The following are some examples of database applications: multimedia applications (e.g., YouTube, Spotify), biometric applications (e.g., fingerprints, retina scans), wearable applications (e.g., Fitbit, Apple Watch), Geographic Information Systems (GIS) applications (e.g., Google Maps), sensor applications (e.g., nuclear power reactor), big data applications (e.g., Walmart), and IoT applications (e.g., Telematics).

CONCEPTS IN PRACTICE

Specialized Database Systems

Database systems are fascinating because they must be given the ability to model various types of data at various levels of abstractions and various levels of details; therefore, it is possible to create specialized systems that can help people in various research domains and industries represent and use their data in the best possible way. For example, SciDB is an open-source data management system intended primarily for use in application areas that involve very large-scale array data. SciDB can be used to support a large variety of scientific applications used in astronomy, remote sensing and climate modeling, bioscience information management, risk management systems for financial applications, and web log data analysis.

Elements of Database Systems

The major elements of a database management system include hardware, software, data, procedures, language, and users. Hardware includes the physical devices such as computers and hard disks. Software refers to the set of programs that are used to manage and control a database. Data includes raw data and information organized and processed within the database. Procedures are instructions used to manage the database. A **database user** is a person who has the privileges to access, analyze, update, and maintain the data (e.g., information architect, database designer, database administrator, database application developer,

and business user). A **database language** is used to write instructions to access and update data in the database. Table 8.1 shows different examples of database languages along with definitions, and examples.

Language	Definition	Relational DBMS (SQL)
database description language (DDL)	Used to create, update, and delete storage structures in a database management system	`CREATE TABLE table_name ( column1 datatype, column2 datatype, column3 datatype, … );`
data manipulation language (DML)	Used to create, update, and delete data in a database management system	`UPDATE table_name SET column1 = value1, column2 = value2, column3 = value3, … WHERE condition;`
data query language (DQL)	Used to query the database	`SELECT expressions FROM table_name WHERE conditions;`
data control language (DCL)	Used to control the use of features that are available in the database management system.	`GRANT [privilege] ON [object] TO [user]`

Table 8.1 Database Languages

Database Systems and Database Management

There are many advantages to using a DBMS such as controlling redundancy in data storage. When the same piece of data is held in two separate places in the database, **data redundancy** can occur. DBMS improves development and maintenance efforts, sharing of data among multiple users, and restricting unauthorized access to data. For example, only the DBA staff uses privileged commands and facilities providing persistent storage for program objects (e.g., object-oriented DBMSs make program objects persistent), providing storage structures (e.g., indexes) for efficient query processing. Additionally, a DBMS provides data independence, which helps the user to easily make changes to the database.

There are two types of data independence. The first, **physical data independence**, separates the conceptual level from the physical level (e.g., using a new storage device such as a hard drive); the second, **logical data independence**, separates any changes in the data from the data format. DBMS provides integrity rules by adding a primary key to guarantee that every record is unique. DBMS allows you to manage structured, semistructured, and unstructured data.

Data that have been organized into a formatted database and have relational keys (e.g., relational data) are called **structured data**. Data that are not organized in a formatted database but have some organized properties (e.g., XML data) are called **semistructured data**. Data that are not organized in a formatted database and do not have organized properties (e.g., PDF) are called **unstructured data**. The DBMS provides a backup copy of the entire database, which is backed up as decided, perhaps once a day. The backup copy should be stored in a secured location and is used to restore the database in the event of failure, loss, or damage to the original data.

TECHNOLOGY IN EVERYDAY LIFE

Databases Impact All Industries

Databases, database technology, and “big data” (complex datasets) have a tremendous impact in industries such as banking, insurance, retail, health care, real estate, e-commerce, law, education, and more recently, social networks. Data collection is also impacting advancements in the fields of medicine, environmental studies, science, and mobile technology.

Can you think of any other industries in which the use of big data is surprising? Does big data benefit the users or the collectors? Will big data enable more scientific research and/or discoveries? Why or why not?

DBMS Facets

DBMS includes various components as in Figure 8.6. **DBMS interface** is the main line of communication between the database and the user. There are many types of interfaces such as web-based, stand-alone query language, command line, forms-based, graphical user interface (GUI), natural language, admin, and network.

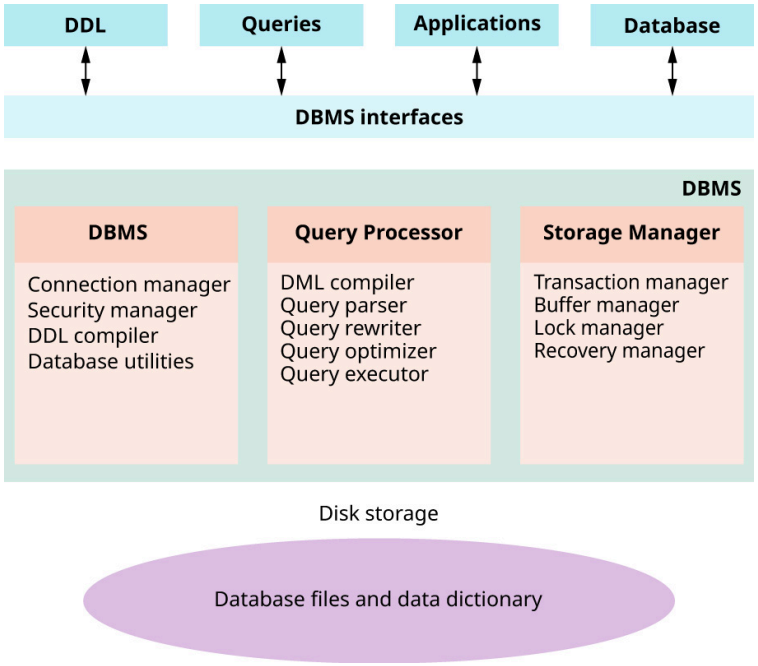


Figure 8.6 DBMS includes multiple components: DBMS interface, query processor, storage manager, connection manager, security manager, DDL compiler, and database utilities. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

A **connection manager** manages reports, books, objects, and batches. For each database, the connection manager provides the connection name, data source type, and value. A **security manager** is a collection of processes used to secure the database from threats. The **data description language (DDL) compiler** translates statements in a high-level language into low-level instructions that the query evaluation engine understands. A **query processor** acts as an intermediary between users and the DBMS data engine to communicate query requests including DML compiler, query parser, query rewriter, query optimizer, and query executor. A **storage manager** is a program that is responsible for editing, storing, updating, deleting, and retrieving data in the database such as transaction manager, buffer manager, lock manager, and recovery manager. **DBMS utilities** are a set of utilities for managing and controlling database activities such as loading utility, reorganization utility, performance-monitoring utilities, user management utilities, and backup and recovery utility. Figure 8.7 illustrates the navigation window, results window, log window, and query window.

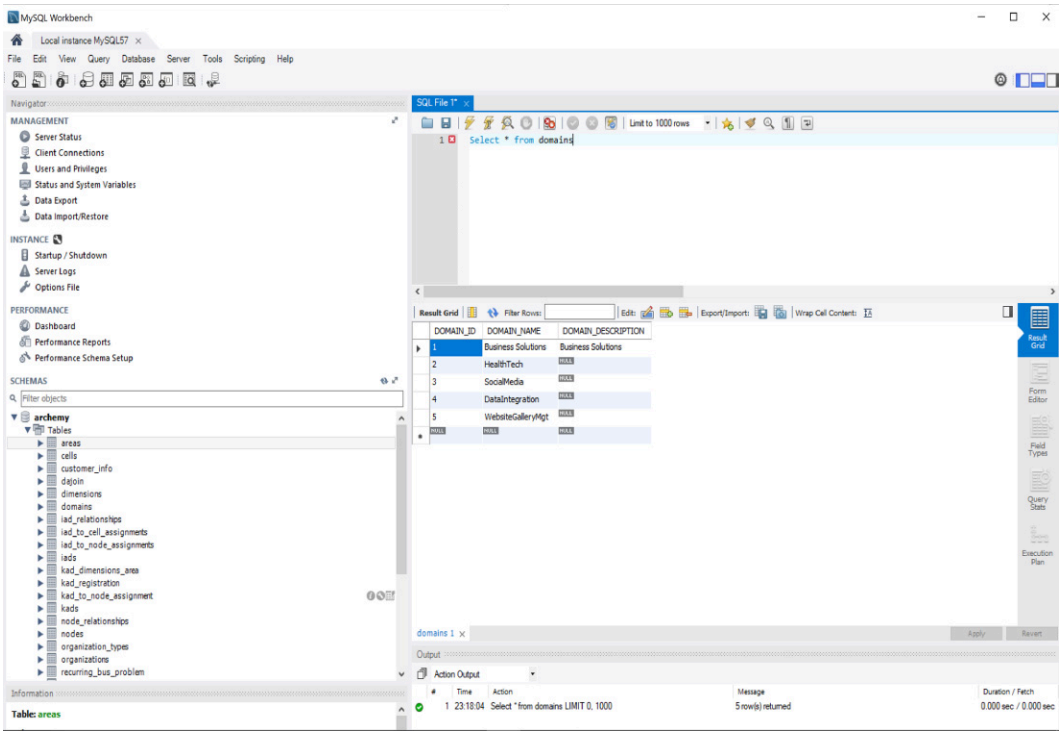


Figure 8.7 This sample MySQL Workbench graphical user interface shows the navigation window, query window, and result window. (credit: used with permission from Oracle)

Data Model Categories

Data models can be conceptual (represented using EER or UML notations), logical, physical, or self-describing. The various logical data models can be categorized as follows:

- A **hierarchical DBMS** is a data model in which the data are organized into a treelike model, DML is procedural and record-oriented, the query processor is logical, and internal data models are intertwined (e.g., IMS from IBM).
- A **network DBMS** is a data model in which the data are organized into a network model, DML is procedural and record-oriented, the query processor is logical, and internal data models are intertwined (e.g., CA-IDMS from Computer Associates).
- A **relational DBMS** is a data model in which the data are organized into a relational data model, use SQL as a declarative and set-oriented database, the query processor has a strict separation between the logical and internal data model. Relational DBMSs are the most popular in the industry (e.g., MySQL open-source database from Oracle, Oracle DBMS, DB2 from IBM, and Microsoft SQL).
- An **object-oriented DBMS** is a data model in which the data are organized into an OO data model, avoiding impedance mismatch when used with an OO host language. It is also called OODBMS or ODB (e.g., db4o open-source database from Versant, Caché from Intersystems, GemStone/S from GemTalk Systems, which are only successful in niche markets, due to their complexity).
- An **XML DBMS** is a data model in which the data are using the XML data model to store data. XML could be native XML DBMS (e.g., BaseX and eXist), which map the tree structure of an XML document to a physical storage structure, or XML-enabled DBMS (e.g., Oracle and IBM Db2) are existing DBMS that are extended with facilities to store XML data.
- A not-only SQL DBMS, or **NoSQL DBMS**, comes in a variety of big unstructured data classified as a document, a graph, key-value stores, and column-oriented databases. NoSQL DBMS focuses on scalability and the ability to cope with irregular or highly volatile data structures (e.g., Apache Hadoop, MongoDB, Neo4j). The irregular data structures appear when the data don't follow a specific order or nature. The volatile data are usualy stored in cache memory and is easy to lose.

Single vs. Multiuser DBMSs

With a **single-user DBMS**, only one user at a time can use the database. A **multiuser DBMS** allows many users to use the database concurrently. A multiuser DBMS is illustrated in Figure 8.8.

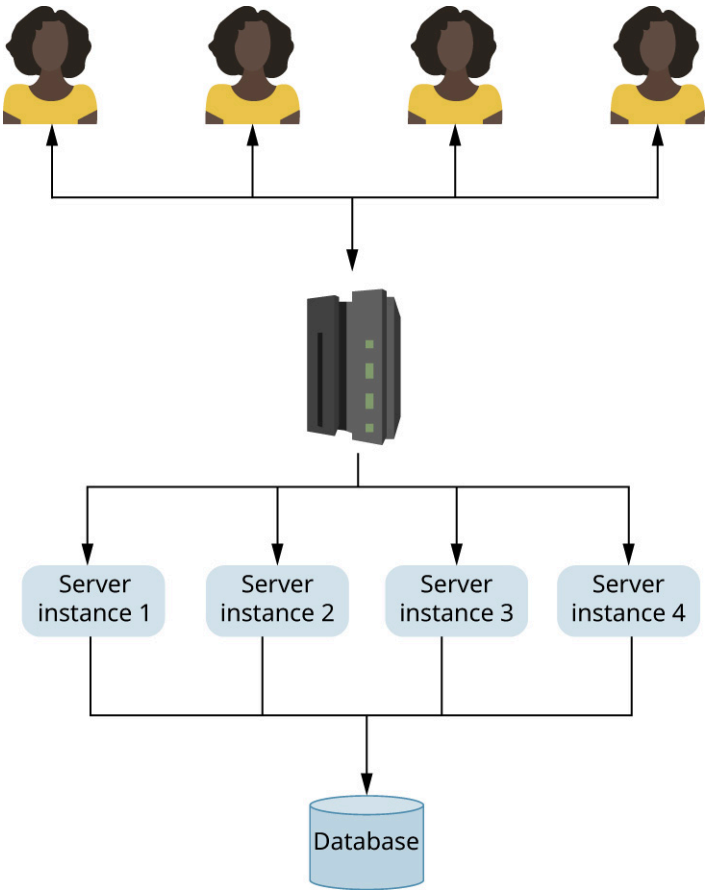


Figure 8.8 Multiple users are accessing the same system at the same time in this multiuser DBMS. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

DBMS Users

DBMS users may be divided into actors on the scene who use and control the database content; design, develop, and maintain database applications such as database administrators and database designers; and workers behind the scenes who design and develop the DBMS software and related tools as well as the computer systems operators, including system designers and developers.

DBMS Architectures

There are various types of database architectures as follows: centralized DBMS architecture, n-tier DBMS architecture, cloud DBMS architecture, federated DBMS, and in-memory DBMS. In **centralized DBMS architecture**, the data are maintained on a centralized server at a single location while client-server DBMS architecture has one or more client computers connected to a central server over a network. Active clients request services from passive servers. An **n-tier DBMS architecture** usually divides an application into three tiers/layers: the presentation tier, the logic tier, and the data tier (e.g., a client with GUI functionality, an application server with applications, a database server with a DBMS and a database, and a web server for web-based access). In **cloud DBMS architecture**, the DBMS and database are hosted by a third-party cloud provider (e.g., Apache Cassandra project and Google's BigTable). A **federated DBMS** provides a uniform interface to multiple underlying data sources which hide the underlying storage details to facilitate data access. An **in-memory DBMS** stores all data in internal memory instead of slower external storage (e.g., disk; often used for real-time purposes such as HANA from SAP).

DBMS Usage Areas

DBMS applications include the following: online transaction processing, online analytical processing, big data and analytics, multimedia DBMS, and open-source DBMS. In **online transaction processing (OLTP)**, the focus is on managing operational or transactional data. The database server must be able to process lots of simple transactions per unit of time. A DBMS must have good support for processing a high volume of short and simple queries (e.g., RDBMS). In **online analytical processing (OLAP)**, the focus is on using operational data for tactical or strategical decision-making. A limited number of users formulate complex queries to analyze the data. Therefore, the DBMS should support efficient processing of complex queries, which often come in smaller volumes (e.g., data warehouses). For example, complex SQL queries involve using queries beyond just the SELECT and WHERE commands. OLAP uses cube as a data structure that represents multiple dimensions to help in data analysis. In big data and analytics, the focus is on more flexible or even schemaless database structure, storing unstructured information such as emails, text documents, X (formerly Twitter) posts, Facebook posts (e.g., NoSQL databases). A **multimedia DBMS** provides storage of multimedia data such as text, images, audio, video, and 3-D games; it should also provide content-based query facilities. An **open-source DBMS** is publicly available and can be extended by anyone (e.g., MySQL from Oracle).

8.3 Relational Database Management Systems

Learning Objectives

By the end of this section, you will be able to:

- Discuss the relational model
- Describe the theoretical and practical aspects of the Structured Query Language (SQL)
- Explain how to create a logical relational database design
- Demonstrate how to create a physical relational database design
- Discuss application programming interfaces and techniques used to program relational database applications
- Identify the various components of a relational database management system

A relational database management system (RDBMS) is one type of DBMS that stores related data elements in a row-based table structure. An RDBMS transaction is a single logical unit of work that accesses and possibly modifies the contents of a database using read-and-write operations. Multiple SQL commands may be performed as part of a single RDBMS transaction as a single logical unit of work. For example, a student may register for multiple courses using a university's registration system. To maintain consistency in a RDBMS, transactions satisfy properties known as **atomicity, consistency, isolation, and durability (ACID)**. These properties are necessary for maintaining data consistency, integrity, and reliability while performing transactions via a RDMBS. Atomicity ensures that multiple SQL commands that are part of the same transaction are executed atomically (i.e., as a unit). In the example mentioned earlier, once the registration transaction is complete, the student will be registered for all three courses. If that is not possible because one of the courses is full, the transaction will not complete successfully. Consistency ensures that only valid data are written in the database. In the student registration example, it simply means that the registration information stored in the database will be consistent with the registration transaction that was completed. Isolation guarantees that multiple users that access the RDBMS concurrently will not be affected by the transactions performed by others. In the course registration example, multiple students can register concurrently without any of them noticing any side effects that result from the RDBMS being accessed by multiple students concurrently. Finally, durability guarantees that once information has been stored in the database, it will be stay there permanently. In the course registration example, if one student registers successfully, the registration information will not change unless they decide to go back and perform another drop/add transaction. While ACID transaction properties ensure that stored data are reliable and accurate, they also come at a cost. For example, guaranteeing full isolation in a RDBMS requires significant additional processing time.

Due to the use of database logical schemas that constrain the row-based table structure and require each field to abide to specific type constraints, RDBMS are not good at storing unstructured or semistructured data. The rigid schema also makes RDBMS more expensive to set up, maintain, and grow. RDBMS requires users to have specific use cases in advance, and any changes to the schema are usually difficult and time-consuming. In addition, traditional RDBMS were designed to run on a single computer node, which means their speed is significantly slower when processing large volumes of data. Sharding RDBMS is a partitioning method that splits the database into smaller components, which makes the management process faster, but using it while maintaining ACID compliance is challenging.

Relational Model

The relational model of data is based on the concept of a relation. A **relation** is a mathematical concept based on the ideas of sets. The model was first proposed by Dr. Edgar F. Codd of IBM Research in 1970. A relation typically contains a set of rows. The data elements in each row represent certain facts that correspond to a real-world entity or relationship. In the formal model, rows are called tuples, each column has a column header that indicates the meaning of the data items in that column, and the column header is referred to as an attribute name or just as an **attribute**. Multiple attributes may be selected to form superkeys that uniquely identify individual tuples. Minimal superkeys are referred to as candidate keys. Candidate keys are considered minimal because they should only include a minimum number of attributes to uniquely identify individual tuples. For example, a car database may use the vehicle identification number (VIN) and the make of the vehicle attributes as a candidate key. Another possible candidate key could be obtained by using the license plate and state of the vehicle. Either one of the candidate keys can be selected as a **primary key**, which then provides a unique identifier for each table record (Table 8.2). A constraint on the primary key is called a **key constraint** and provides a unique value, assigns a default value to a column if none is specified, and checks for predefined conditions before inserting the data.

License_number	Engine_serial_number	Make	Model	Year
Texas ABC-739	A69352	Ford	Mustang	02
Florida TVP-347	B43696	Oldsmobile	Cutlass	05
New York MPO-22	X83554	Oldsmobile	Delta	01
California 432-TFY	C43742	Mercedes	190-D	99
California RSK-629	Y82935	Toyota	Camry	04
Texas RPN-624	U028365	Jaguar	XJS	04

Table 8.2 Primary Key Example in a Car Table for a RDBMS The primary key in a table is typically underlined.

Structured Query Language

The declarative language used in managing and querying structured data in a RDBMS is called **Structured Query Language (SQL)**. SQL implements the four sublanguages mentioned earlier: DDL, DML, DQL, and DCL. SQL is based on relational algebra with many extensions. SQL supports primary keys, foreign keys, inserting data, deleting data, updating data, indexing, recovery, concurrency, and security. SQL focuses on efficiency in addition to specifying the data needed. The main commands in SQL are `CREATE TABLE`, which is used to create a new table in a database such as:

```
CREATE TABLE Student (
    ID int,
    LastName varchar(255),
    ZipCode int
);
```

which will create a table name Student with three columns: ID, LastName, and ZipCode. `DROP TABLE` is used to drop an existing table in a database, and `ALTER TABLE`, which is used to add, delete, or modify columns in an existing table. SQL schemas contain tables (i.e., relations), and the RDBMS catalog stores these schemas. SQL constraints are used to specify rules for the data in a table to ensure the accuracy and reliability of that data. There are additional features of SQL such as triggers, views, nested/recursive queries, privileges, and metadata.

Relational Algebra

The theoretical framework for querying relational databases that uses unary and binary operators on relations to manipulate and retrieve tuples is called **relational algebra**. The following set of relational algebra operations is a complete set: Select, Project, Union, Rename, Difference, and Cartesian product. It is a complete set because any relational algebra operation can be expressed as a sequence of operations from this set.

Relational algebra operators include set operators and relational operators. The set operators include Union, Intersection, Difference, and Product, which are all binary operators because they use two relations as operands and produce a new relation as a result. In some cases, the operand relations need to be union compatible, which implies that they have the same number of attribute types defined on the same domains. In what follows, we represent a relation as R and its cardinality or number of tuples as |R|.

Union Operator

The Union operator applied to two union-compatible relations P and Q results into a new relation $R = P \cap Q$ so:

$R = \{t | t \in P \, OR \, t \in Q\}$ with $\max(|P|, \ |Q|) \leq |R| \leq |P| + |Q|$

The resulting relation R consists of the tuples that either belong to P or Q, or both. Duplicate tuples are eliminated.

Intersection Operator

The Intersection operator applied to two union-compatible relations P and Q results in a new relation $R = P \cap Q$ such that:

$R = \{t | t \in P \, AND \, t \in Q\}$ with $0 \leq |R| \leq \min(|P|, |Q|)$

The resulting relation R consists of the tuples that belong to both P and Q.

Difference Operator

The Difference operator applied to two union-compatible relations P and Q results in a new relation $\mathrm{R} = \mathrm{P} - \mathrm{Q}$ such that:

$R = \{t | t \in P \, AND \, t \notin Q\}$ with $0 \leq |R| \leq |P|$

The resulting relation R consists of the tuples that belong to P but not to Q. Note that when using the Difference operator, the order of the operands, is important because it concerns a noncommutative operation.

Product Operator

The Product operator (also known as the Cartesian product, Cross Product, or Cross Join operator) returns the Cartesian Product of two relations. Consider the following two relations: a supplier relation P and a product relation Q. The Cartesian Product of both relations consists of all possible combinations of tuples from P and Q. The resulting relation $\mathrm{R} = \mathrm{P} \times \mathrm{Q}$ consists of a set of tuples $\mathrm{r} = \mathrm{pq}$ such that:

$R = \{r = pq | p \in P\ AND\ q \in Q\}$ with $|R| = |P| \times |Q|$

The relational operators are Select, Project, Rename, Join, and Division. The Select, Project, and Rename operators are unary operators because they only use one operand. The Join and Division operators are binary operators because they use two operands.

Select Operator

The Select operator is used to return a subset of tuples of a relation based on a selection condition. The latter can consist of predicates combined using Boolean operators (and, or, not). Every predicate consists of a comparison between the values of two domain-compatible attribute types, or between an attribute type and a constant value from the domain of the attribute type.

The result of applying a Select operator with a selection condition S on a relation P can be represented as:

$\mathrm{R} = \sigma_S(\mathrm{P})$ with Select operator σ

Project Operator

The Project operator is used to project a relation on a list of attribute types. The result of applying a Project operator with a list L on a relation P can be represented as:

$\mathrm{R} = \pi_L(\mathrm{P})$ with Project operator π

Note that applying a Project operator to a relation results in a vertical subset of the relation with $|R| \leq |P|$ because duplicate tuples will be eliminated.

Rename Operator

The Rename operator is used to rename an attribute type of a relation (in case, for example, naming conflicts can occur). The result of applying a Rename operator on a relation P to rename attribute type B to A can be represented as:

$\mathrm{R} = \rho_{\mathrm{A|B}}(\mathrm{P})$ with Rename operator ρ

Join Operator

The Join operator allows combining tuples of two relations based on specific conditions, also called join conditions. The result of a join is the combined effect of a Cartesian product (×) and selection. The result of applying a Join operator to two relations P and Q with join condition j can be represented as follows:

$R = P \bowtie_j Q$ with Join operator ⋈ and join condition *j*.

The resulting relation R consists of a set of combinations of pq tuples such that a combined tuple r ∈ R is characterized by:

$R = \{r = pq \mid p \in P\ AND\ q \in Q\ AND\ j\}$ with $|R| \leq |P| \times |Q|$

Division Operator

The Division operator is a binary operator that divides a relation P by a relation Q with $\mathrm{Q} \subseteq \mathrm{P}$. The result of a Division operator can be represented as:

$\mathrm{R} = \mathrm{P} \div \mathrm{Q}$ with Division operator ÷

The resulting relation R includes all tuples that belong to P combined with every tuple in Q. Note that the Division operator is not directly implemented in SQL.

The Select (σ), Project (π), Difference (–), Product (×), and Union (∪) operators are often referred to as the five primitive operators because all other relational algebra operators can be derived from them (derived operators).

SQL also allows aggregate functions and grouping operations. For join operations, there are many options, such as the following:

- A **theta join** allows merging two tables based on a theta condition; theta (θ) refers to the comparison operator in the join condition. Depending upon theta, these joins can be distinguished: greater-than-join, less-than-join, greater-than-or-equal-join, less-than-or-equal-join, and an equi-join (if an equals occurs). An **equi-join** combines tables based on matching values in specified columns; it is equivalent to a theta join, with theta being the equal comparison operator in the join condition.
- A **natural join** creates an implicit join based on the common columns in two tables; it is a variant of the equi-join in which one of the shared join-attribute types is removed from the result.
- An **inner join** represents the intersection of two tables; all joins discussed thus far are inner joins because they do not include tuples lacking corresponding join values in the result.
- An **outer join** represents the union of two tables; the outer join operator will also include tuples lacking corresponding join values in the result.

A distinction can be made between a left outer join, right outer join, and full outer join. A left outer join includes all tuples from the left relation (P) in the result, either matched with a corresponding tuple from the right relation (Q) based on the join condition or augmented with null values in case no match with a tuple from Q can be made. A left outer join can be represented as $R = P \bowtie_j Q$. A right outer join includes all tuples from the right relation (Q) in the result, either matched with a corresponding tuple from the left relation (P) based on the join condition or augmented with null values in case no match with a tuple from P can be made. A right outer join can be represented as $R = P \bowtie_j Q$. A full outer join includes all tuples from P and Q in the result either matched with the counterparts according to the join condition or augmented with null values in case no match can be found. A full outer join can be represented as $R = P \bowtie_j Q$.

An example of data structure representation for the relational algebra expression is a **query tree**.

Tuple Relational Calculus and Domain Relational Calculus

In a relational database, a **tuple** is one row with a collection of values separated by a comma and enclosed in parenthesis. An example of a tuple in Python represents the employee Smith, his phone number, age, and zip code.

```
tuple = {'Smith', 8882355151, 50, 48505}
```

Data inside a tuple can be of any type such as integer, string, float value, or a tuple type.

Tuple relational calculus uses tuple variables as key building blocks. A tuple variable refers to a tuple of a corresponding relation (also called range relation). A tuple relational calculus expression specifies a range relation for each tuple variable, a condition to select (combinations of) tuples, and a list of attribute types from which the values should be retrieved. As a result, a tuple relational calculus expression looks as follows:

```
{t(A_i) | Cond(t)}
```

whereby t represents the tuple variable, A_i the attribute type of which the value needs to be retrieved, and Cond(t) the range relation and extra conditions to select relevant tuples. Consider the following example:

```
{t.StudentID, t.StudentName | Student(t) AND t.StudentProgram = 3}
```

A condition is specified using a well-formed (calculus) formula (wff), which can consist of various predicate calculus atoms combined using Boolean operators (and, or, not). The result of applying a formula to a tuple of a range relation can either be true, false, or unknown. If the result is true, the tuple will be selected in the result. Relational calculus also includes quantifiers such as the existential quantifier (∃ or EXISTS) and universal quantifier (∀ or FOR ALL). A formula with an existential quantifier (∃) evaluates to true if at least one tuple satisfies it. A formula with a universal quantifier (∀) evaluates to true if every tuple from the range relation satisfies the conditions stated.

Instead of tuple variables, domain relational calculus defines domain variables that range over single values of domains of attribute types. A domain calculus expression for a relation of degree n looks as follows:

$_1$, v_2, …v_n, | COND(v_1, v_2, …v_n) }{v1, v2, …vn, | COND(v1, v2, …vn) }

where v_1, v_2, ...v_n represent the domain variables that range over domains of attribute types, and COND represents the extra conditions.

The earlier tuple relational calculus query selecting the student ID and student name of all students who study in program number 3 would now look as follows:

```
{ab| ∃(e) (Student(abcde) AND e = 3)}
```

In this case, we defined five domain variables as *a* for the domain of StudentID, *b* for the domain of StudentName, *c* for the domain of StudentAddress, *d* for the domain of StudentCity, and *e* for the domain of StudentProgram. The condition *e* = 3 is a selection condition.

The existential quantifier (∃ or EXISTS) and universal quantifier (∀ or FOR ALL) are also supported in domain relational calculus.

Query-by-Example (QBE) Language

The relational database language based on domain relational calculus is called **query-by-example (QBE)**. QBE queries are expressed via a graphical query language, using visual tables where the user enters commands, example elements, and conditions. A parser converts the QBE graphical queries into statements expressed in a database manipulation language, such as SQL.

Logical Design

The process of **logical design** involves designing a database based on a specific data model but independent of physical details. The logical design for a relational DBMS includes specifications for tables, relationships, and constraints. The logical design is performed in four steps: map conceptual model to logical model components, validate the logical model using normalization, validate logical model constraints, and validate the logical model against user requirements.

To determine the quality of relation schema design, we follow the informal design guidelines for relational schemas and use simple measures of quality such as making sure attribute semantics are clear, reducing redundant information, and reducing null values. Null values cause redundant work, which wastes storage space and increases the difficulty of performing operations.

TECHNOLOGY IN EVERYDAY LIFE

RDBMS

Airplanes have to be designed and tested for safety. Airplane wings are complex items that include a large number of parts, come in various shapes and forms (e.g., rectangular wings for small airplane, elliptical wings, tapered wings and trapezoidal wings), and require multiple designers to collaborate on their design.

Airplane wings are typically designed using computer-aided design (CAD) software systems that use data management systems to store and retrieve data.

Is a RDBMS the best solution to store and retrieve these types of items? Why or why not? Are there other, better ways to track the development, design, and testing of mechanical parts crucial to passenger safety?

Relational Model Constraints

The constraints that ensure database correctness are called relational model constraints in DBMS. The different types of constraints are domain, uniqueness, key, and entity integrity constraints. The **domain constraint** defines the domain of values for an attribute. The **uniqueness constraint** specifies that all the tuples must be unique. The **key constraint** specifies that all the values of the primary key must be unique. The **entity integrity constraint** specifies that no primary key contains a null. Additionally, **functional dependency (FD)** is a constraint that specifies the relationship between two sets of attributes and provides a formal tool for the analysis of relational schemas. FDs enable the detection and description of business rules in precise terms. FDs can be used to help normalize relational schemas. A **multivalued dependency (MVD)** occurs when two attributes in a table are independent of each other but both depend on a third attribute. MVDs are required to achieve higher forms of normalization (e.g., 4NF). The normalization process can be automated using algorithms that can convert a given relation into sets of relations in a given normal form.

Normalization

The process of structuring a relational database to reduce data redundancy and improve data integrity is called **database normalization**. A normal form applies to a table/relation, not to the database. The main types of normalization are

- first normal form (1NF),
- second normal form (2NF),
- third normal form (3NF),
- Boyce-Codd normal form (BCNF), and
- fourth normal form (4NF).

There are additional normal forms, which are more complex. Each cell in 1NF must contain only a single (atomic) value and every column must be uniquely named. The relational model requires tables to be in 1NF form; tables in 2NF must be in 1NF and not have any partial dependency. Tables in 3NF must be in 2NF and have no transitive functional dependencies. BCNF is a higher version of the 3NF and is used to address the anomalies. Tables in 4NF must be in BCNF/3NF and not have MVDs. The following example converts a table to 1NF, 2NF, and then 3NF. Table 8.3 contains the original data, that is then converted to 1NF (Table 8.4), 2NF (Table 8.5), and 3NF (Table 8.6, Table 8.7, and Table 8.8). It is assumed here that the Customer Name is a composite attribute that contains First Name and Last Name and therefore must be broken down into its component attributes to be stored in a relational table that satisfies 1NF.

Customer Name	Item 1	Item 2
Shirl Adam	Milk	Bread
Jeff Mark	Juice	Water
Rana Park	Bread	Milk

Table 8.3 Data Table

First Name	Last Name	Item 1	Item 2
Shirl	Adam	Milk	Bread
Jeff	Mark	Juice	Water
Rana	Park	Bread	Milk

Table 8.4 Table Converted to 1NF

First Name	Last Name	Item
Shirl	Adam	Milk
Shirl	Adam	Bread
Jeff	Mark	Juice
Jeff	Mark	Water
Rana	Park	Bread
Rana	Park	Milk

Table 8.5 Table Converted to 2NF

Customer ID	First Name	Last Name
1	Shirl	Adam
2	Jeff	Mark
3	Rana	Park

Table 8.6 Table Converted to 3NF: Customer

Item ID	Item
1	Milk
2	Bread
3	Juice
4	Water

Table 8.7 Table Converted to 3NF: Item

Customer ID	Item ID
1	1
1	2
2	3
2	4

Table 8.8 Table Converted to 3NF: Relational Table

Relational Database Design

Modeling data into a set of tables with rows and columns is called **relational database design (RDD)**. Each row represents a record, and each column represents an attribute. RDD includes five phases:

1. Requirements analysis phase to assess the informational needs of an organization.
2. Logical design phase to create a conceptual model (entity-relationship [ER]) based on phase 1. ER describes interrelated attributes in a specific domain of knowledge.
3. Physical design phase to maximize database efficiency (Unified Modeling Language [UML]). UML helps developers visualize the relationships between the different pieces.
4. Implementation phase to convert the tables developed in the ER diagram into SQL.
5. Monitoring and maintenance phase to ensure that RDBMS is functioning properly.

INDUSTRY SPOTLIGHT

RDBMS and Banking

RDBMs are ideal in banking applications for tracking and storing account numbers, orders, and payments. Most banks run Oracle applications because of the powerful integration of technology and business applications. Oracle includes built-in functionality that is designed specifically for banks such as Oracle banking platform, which is the leading choice for banks looking to change their core systems for customer-centric retail banking.

Mapping a Conceptual EER Model to a Relational Model

After designing the ER diagram, we need to convert it to relational models that can be directly implemented by any RDBMS. ER modeling helps create a conceptual model that relies on entities and their interrelationships. The degree of a relationship establishes how many entities it interrelates. Therefore, relationships can be unary, binary, ternary, and more generally *n*-ary.

Cardinality ratios are used to specify how many tuples are interrelated as part of a binary relationship (i.e., 1-1, 1-N, N-1, M-N).

Multivalued attributes are allowed in ER models. They are used to represent an attribute that contains a list of values. For example, a car entity may have a color attribute that is multivalued and include a list of all the car colors.

In ER modeling, a **weak entity** is a type of entity that cannot be uniquely identified based on its attributes alone and must rely on a strong entity to provide the context necessary for identification. For example, an

employee in a company may have dependents for health insurance purposes. In this case, a dependent entity is a weak entity and the ID of the employee, who is the strong entity, needs to be used to fully identify a dependent.

The following steps must be followed to map an EER model to a relational model:

1. Mapping of Regular Entity Types. Create a table that includes all of its simple attributes and select one of the key attributes as the primary key. If the key chosen is from a composite attribute in the EER model, the corresponding set of simple attributes will together form the primary key.
2. Mapping of Weak Entity Types. For each weak entity, create a table that includes all of its simple attributes, and include the attributes that make up the key of the associated strong entity. The primary key in this case is the combination of the primary key of the associated strong entity and the partial key of the weak entity if any.
3. Mapping of Binary 1:1 Relation Types. One solution is to create a relation for each participating entity, choose a primary key in one of the two relations, and make a foreign key referencing the primary key in the second relation. In some cases, it is more appropriate to merge both relations (from the participating entities) into one, or create a cross-reference (i.e., relationship relation) between the two relations.
4. Mapping of Binary 1:N Relationship Types. Create a relation for each participating entity. The primary key from the "one" side of the relationship is added to the "many" side as a foreign key. An alternative approach is to use a relationship relation, but it is rarely done.
5. Mapping of Binary M:N Relationship Types. Create a relation for each participating entity. Also create a relationship relation and include as foreign keys attributes the primary keys of the relations that represent the participating entity types. Their combination will form the primary key of the relationship relation.
6. Mapping of Multivalued Attributes. For each multivalued attribute, create a new relation that includes an attribute corresponding to the multivalued attribute (in the original entity type) and include as a foreign key the primary key of the relation that represents the entity that includes the multivalued attribute. The primary key in the new relation created is the combination of the attribute that corresponds to the multivalued attribute (in the original entity type) and the foreign key.
7. Mapping of *n*-ary Relationship Types. Create relationship relations for each *n*-ary relationship and include as foreign keys attributes the primary keys of the relations that represent the participating entity types.
8. Mapping Specialization or Generalization. Convert each specialization with m subclasses $\{S_1, S_2, \ldots, S_m\}$ and generalized superclass C, where the attributes of C are $\{k, a_1, \ldots a_n\}$ and k is the (primary) key, into relational schemas using one of the four following options: Option 8A: Multiple relations-Superclass and subclasses; Option 8B: Multiple relations-Subclass relations only; Option 8C: Single relation with one type attribute; Option 8D: Single relation with multiple type attributes.
9. Mapping of Union Types. For mapping a category whose defining superclass has different keys, it is customary to specify a new key attribute, called a surrogate key, when creating a relation to correspond to the category.

Physical Design

Attributing logical concepts to physical constructs is called **physical database design**. The input to the physical design step is a logical representation of the system. It is a joint responsibility of the database designer and DBA. The main responsibility of the physical design is optimizing performance while ensuring data integrity. The main issues addressed in physical design are storage media, file structures, and indexes. The database is stored on a disk using a "standard" file system, not one "tailored" to the database. The database runs on an unmodified general-purpose operating system.

Disk Storage and Basic File Structures

Database tables are stored in **disk storage**—which is the memory device that stores the data—such as hard disks, flash memory, magnetic disks, optical disks, and tapes as shown in Figure 8.9. The records in the tables could be of fixed length, which means the records are exactly the same length, or variable length, which means the records differ in their length. All records are classified into blocks, and the length of the record can exceed the size of a block, which is called a **spanned record**. Assume that each relation is stored as a file and each tuple is a record in the file. The files could be ordered (i.e., the records are sequenced on some field) or unordered (i.e., the records are not in any order). The average record access time is the average time taken for a system to access the record. A **redundant array of inexpensive disks (RAID)** stores information across an array of low-cost hard disks.

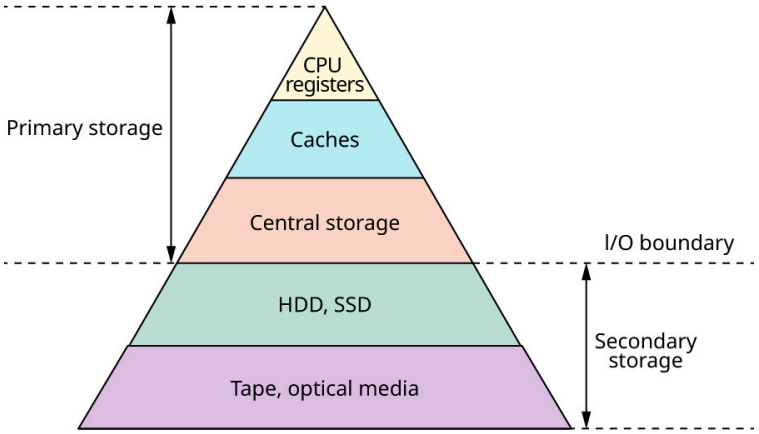


Figure 8.9 This storage hierarchy includes primary and secondary storage. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Physical File Organization and Indexing

File organization and indexing are used to minimize the number of block accesses for frequent queries. There are many types of file organization, with the most popular being sequential, relative, and indexed organization. In a **sequential file organization**, records are organized in the order they are stored and any new record is added at the end. In a **relative organization**, each record is assigned a numeric key to rearrange the order of the records at any time. Similar to the relative organization, **indexed organization** uses a key, but the key is unique and fixed. In addition, there are many other file organization types such as heap, hash, B+, and cluster file organization. A heap in file organization refers to an unordered collection of records where new records are placed wherever there is free space, and no particular ordering is enforced. Figure 8.10 shows an example of a heap file organization.

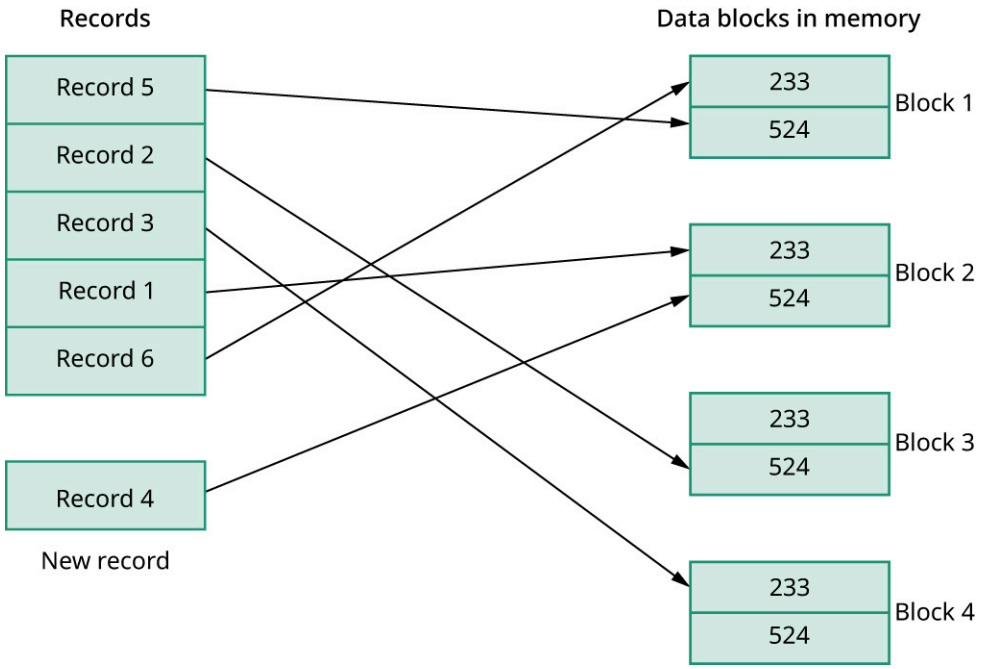


Figure 8.10 This heap file organization example shows how to insert a new record (record 4) into the data blocks in the memory. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Physical Database Organization

A **tablespace** is where tables are stored physically in the memory (i.e., the physical container of database objects). The use of tablespaces has an impact on physical database design concerning intraquery parallelism, where different subsets of data can be searched in parallel for a complex query, and interquery parallelism (i.e., many simple queries executed in parallel).

Query Execution Concepts

Processing the query includes five steps:

1. parsing: checking the syntax of the query, user's privileges, table, and attribute names
2. **translation**: translating from high-level language to machine language
3. **optimizer**: selecting the best plan to execute
4. execution: running the selected plan
5. evaluation: returning the query result

The query optimizer determines the lowest cost strategy for query execution using the internal representations of a query (i.e., query tree and query graph). The mathematical technique for processing the query quickly is called **heuristics optimization**. The main heuristic is to first apply the operations that reduce the size of intermediate results.

Physical Database Design and Tuning

Assume you are looking for a specific record in a huge database and that the record you are looking for is at the end. To retrieve that record, the system moves sequentially record by record and takes a long time to process. Let us follow this SQL exercise using Table 8.9.

Employee			
Number	FName	LName	Country
1	Sam	Smith	USA
2	Adam	Bell	UK
3	Matt	John	USA
4	John	Zack	USA
5	Andy	Dave	UK

Table 8.9 Employee Table

We are looking for “Andy” using the following SQL statement:

```
SELECT * FROM employee WHERE Fname = 'Andy';
```

Looking for Andy takes five comparisons because Andy is at the end of the table. However, if we sort the data alphabetically (Table 8.10), searching for a name happens faster (two comparisons).

Employee_sort		
FName	LName	Index
Adam	Bell	2
Andy	Dave	5
John	Zack	4
Matt	John	3
Sam	Smith	1

Table 8.10 Employee Sort Table

An index holds the field being sorted and points from each record to the corresponding record where the data are stored (Figure 8.11).

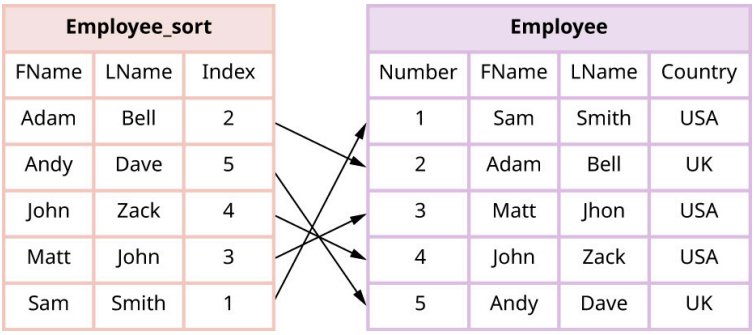


Figure 8.11 Using indexing can improve the query performance. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Indexing helps improve the performance of the query process. Creating an index in SQL retrieves data from the database more quickly. The following are SQL syntaxes for creating an index and creating a unique index. Note that a UNIQUE refers to an index where all rows of the index must be unique. That is, a row may not have identical non-null values for all columns in this index as another row.

```
CREATE INDEX index_name ON table_name (column1, ...);
CREATE UNIQUE INDEX index_name ON table_name (column1, ...);
```

In addition, you can delete the index anytime using the drop statement:

```
DROP INDEX index_name ON table_name;
```

A key is needed to index an attribute; however, an index can be constructed with one or multiple attributes. If a table requires more than one index, we need to decide which attribute we cluster. In general cases, the indexing is implemented using binary tree, B-tree, or B+ tree. In practice, B+ trees are more commonly used in database systems. Binary tree is a tree in which every node has up to two children. B-tree generalizes the binary tree with more than two children and may have one child. B+ tree is a tree with a large number of children per node but in some cases it is more efficient to use hash indexing (e.g., equality conditions). The normalized database is the database with eliminating data redundancy to enhance the data integrity in the table. Denormalization is a strategy to increase the performance of a normalized database. It adds redundant data such as extra attributes, adds new tables, or creates instances. The main goal is to decrease the running time of queries by making data more accessible.

THINK IT THROUGH

The Relational Model and Constructors

The relational model uses two types of constructors. One is “aggregate of,” which enables the creation of tables by aggregating various columns that contain data of different types. The other is “set of,” which enables the gathering of multiple records within a table.

Is there a limitation with this approach?

RDBMS APIs and Programming Techniques

A **database architecture** is a representation of the design that helps design, develop, implement, and maintain the DBMS. This section covers RDBMS architectures, database APIs, object persistence and object-relational mappings (ORMs), and sample RDBMS applications.

RDBMS Architectures

A relational database management system (RDBMS) architecture is categorized as a centralized system architecture and a tiered system architecture. In centralized system architecture, the DBMS's responsibilities are handled by one centralized entity—the mainframe database, which has become rare, expensive, and difficult to maintain. The tiered system architectures aim to decouple the centralized setup by combining the computing capabilities of powerful central computers with the flexibility of personal computers (PCs).

There are multiple variants of the tiered system architecture such as two-tier architecture or client-server architecture. The **fat client variant** is where presentation logic and application logic are handled by the client and is common in cases where it makes sense to couple an application's workflow with its look and feel; in that case, the DBMS now fully runs on the database server. A **thin client variant** is one where only the presentation logic is handled by the client and applications and database commands are executed on the server; it is common when application logic and database logic are tightly coupled or similar. Three-tier architecture decouples application logic from the DBMS and puts it in a separate layer (i.e., application server). Note: application server or database server may consist of multiple physical, distributed machines.

Database APIs

In a tiered DBMS system architecture, client applications can query database servers and receive results. It is possible to use in-process DBMSs (e.g., SQLite) or access stand-alone DBMS servers (e.g., Postgres). Client applications that wish to utilize the services provided by a DBMS use a specific API provided by the DBMS. This database API exposes an interface through which client applications can access and query a DBMS. The database server receives calls made by clients and executes the relevant operations before returning the results. In many cases, the client and server interfaces are implemented to work over a computer network using network sockets. The main goal of a database API is to expose an interface through which other parties can utilize the services provided by the DBMS. Most DBMS vendors provide a proprietary, DBMS-specific API (the disadvantage is that client applications must be aware of the DBMS that will be utilized on the server side). Alternatively, generic, vendor-agnostic universal APIs have been proposed, and they allow to easily port applications to multiple DBMSs.

APIs can be embedded or call-level. Embedded API embeds SQL statements in the host programming language, meaning that SQL statements are part of the source code. (Before the program is compiled, a SQL precompiler parses the SQL-related instructions and replaces them with source code instructions native to the host programming language used. Converted source code is then sent to the actual compiler.) Call-level APIs (SQL/CLI) work by passing SQL instructions to the DBMS by means of direct calls to a series of procedures, functions, or methods as provided by the API to perform the necessary actions (e.g., ODBC/JDBC).

Object Persistence and Object-Relational Mappings (ORMs)

API technologies such as JDBC and ADO.NET represent database-related entities (e.g., tables, records) in an object-oriented (OO) way. As plain domain entities such as Book and Author are represented as objects using a programming language's representational capabilities and syntax, object persistence ensures that the corresponding object data are persisted behind the scenes to a database or other data source. Language-integrated query technologies apply similar ideas while object persistence APIs go a step further as they also describe the full business domain (i.e., the definition of data entities) within the host language. They allow for efficient querying of objects; such entities are frequently mapped to a relational database model using an object relational mapper (ORM). It is not strictly necessary to utilize an ORM to enable object persistence, though most APIs tightly couple both concepts.

Sample RDBMS Applications

On the Web, there are two types of calls: asynchronous and synchronous. In the **asynchronous call**, the client sends a request without waiting for a response. In the **synchronous call**, the client sends a request and waits

for a response from the service. API and HTTP calls are examples of synchronous calls. An example of a database fully managed on the Web is Oracle's NoSQL data services.

RDBMS Features

RDBMS includes multiple features such as transaction management, concurrency control, data distribution, distributed transaction management, distributed and parallel processing, recovery, and security.

Transaction Management

Most database systems are multiuser. While transaction management allows concurrent access to the same data by multiple users, it may induce different types of anomalies. As a result, errors may occur in the DBMS or its environment. RDBMS must ensure that transactions support ACID (atomicity, consistency, isolation, durability) properties as explained in the following paragraphs.

A **transaction** is a set of database operations induced by a single user or application that should be considered as one undividable unit of work (e.g., transfer between two bank accounts of the same customer). A transaction either succeeds or fails in its entirety.

A transaction renders the database from one consistent state into another consistent state. The transaction manager supervises the execution of database transactions. A **database transaction** is a sequence of read/write operations considered to be an atomic unit. The transaction manager creates a schedule with interleaved read/write operations, guarantees ACID properties, and can COMMIT a transaction upon successful execution or ROLLBACK a transaction upon unsuccessful execution. Delineating transactions within the transaction life cycle is called **transaction management**. Various components are involved in transaction management (i.e., scheduler, recovery manager, stored data manager, and buffer manager), and RDBMSs use a log file to register the current state of the transaction (active, committed, or aborted) and facilitate the implementation of checkpoint for rollback purposes.

Concurrency Control

The coordination of transactions that execute simultaneously on the same data so that they do not cause inconsistencies because of mutual interference is called **concurrency control**. A lock manager provides concurrency control, which ensures data integrity at all times. The two types of locks are read locks and write locks. The lock manager is responsible for assigning, releasing, and recording locks in the catalog. The lock manager makes use of a locking protocol that describes the locking rules, and a lock table that contains the lock information. Concurrency problems occur when multiple transactions execute concurrently without control.

Data Distribution

The process of storing data in more than one site to improve the data availability and retrieval performance is called **data replication**. Multiple databases located at different sites may need to be synchronized in real time to ensure consistency and optimal performance (e.g., fragmentation). There are different types of fragmentation:

- **vertical fragmentation**: consists of a subset of columns of data; global view can be retrieved with JOIN query; useful if only some of a tuple's attributes are relevant to a node
- **horizontal fragmentation (sharding)**: the fragment consists of rows that satisfy a query predicate; global view can be retrieved with UNION query; common in NoSQL databases
- **mixed fragmentation**: combines horizontal and vertical fragmentation; global view can be retrieved with JOIN + UNION query

Distributed Transaction Management

A **distributed transaction** is a set of operations that are performed across multiple database systems. Distributed transaction management coordinates the resources between multiple databases. The transaction

manager decides whether to commit or rollback a transaction using a two-phase commit (2PC). The steps of 2PC are described in Figure 8.12.

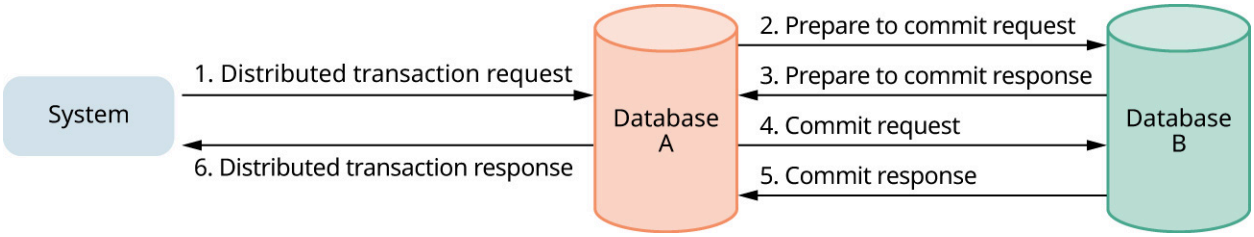


Figure 8.12 Two-phase commit protocol illustrates the six steps needed for a distributed transaction. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The transaction management server manages the transactions in loosely coupled and tightly coupled modes. In loosely coupled, the different database servers coordinate transactions without sharing resources; in the tightly coupled, resources are shared.

Parallel and Distributed Processing

Many databases including NoSQL use **parallel processing**, a technique in which multiple processors work simultaneously on different tasks or different parts of a task to enable concurrent processing of large amounts of data. Distributed database systems distribute data and data retrieval functionality over multiple data sources or locations. Figure 8.13 shows different architectures of distributed databases. In shared-memory architecture, multiple interconnected processors that run the DBMS software share the same central storage and secondary storage. With shared-disk architecture, each processor has its central storage but shares secondary storage with other processors using a Network Attached Storage (NAS) or Storage Area Network (SAN). In shared-nothing architecture, each processor has its central storage and hard disk units, and data sharing occurs through the processors communicating with one another over the network.

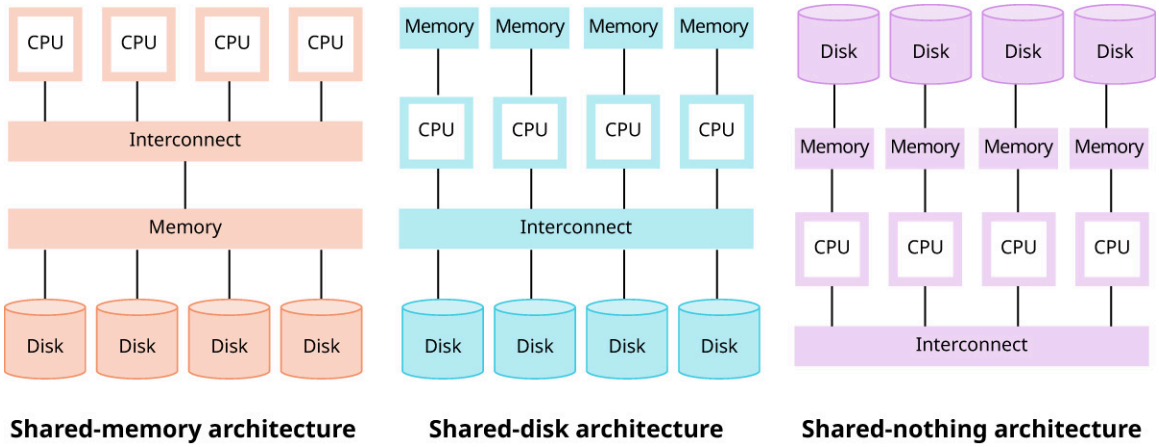


Figure 8.13 The different distributed architectures are shared-memory architecture, shared-disk architecture, and shared-nothing architecture. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Scalability is the measure of a system's ability to scale (increase or decrease) the performance and/or cost in response to any system change. Scalability can be achieved through vertical scalability, where the capacity of each node can be increased, and horizontal scalability in which more nodes can be added. Parallel databases focus on data distribution for performance and intraquery versus interquery parallelism. Federated databases use nodes in a shared-nothing architecture, each running an independent DBMS instance with horizontal data fragmentation.

In distributed query processing, the optimizer should not only consider the elements of a stand-alone setting but also the properties of respective fragments, communication costs, and location of the data in the network. Metadata may also be distributed, both globally (across all nodes) and locally (within a single node). Query

optimization is needed.

Recovery

The activity of setting the database in a consistent state without any data loss in the event of a failure or when any problem occurs is called **database recovery**. System failure occurs when a system goes down in the middle of a transaction, and media failure occurs when a database file or the device storing the database file stops working. There are many recovery techniques such as mirroring by adding two complete copies of the database, transaction logs by recording the transitions' changes, shadow paging by dividing the database into pages, and backups. Backups are the most-used method for recovery and include immediate backup and archival backup. An **immediate backup** stores the copies in disks, and an **archival backup** stores the data in different servers/sites.

Security

Using a set of controls to secure data, guaranteeing a high level of confidentiality, is considered **database security**. Security issues facing a database include human errors such as password sharing and weak passwords; SQL injection, which involves the use of SQL attack strings in database queries; overflow attacks, which writes a large amount of data to a fixed-length block of the memory, causing overflow; Denial of Service Attacks (DoS) that use a large amount of requests to crash the server; and internal and external attacks (Figure 8.14).

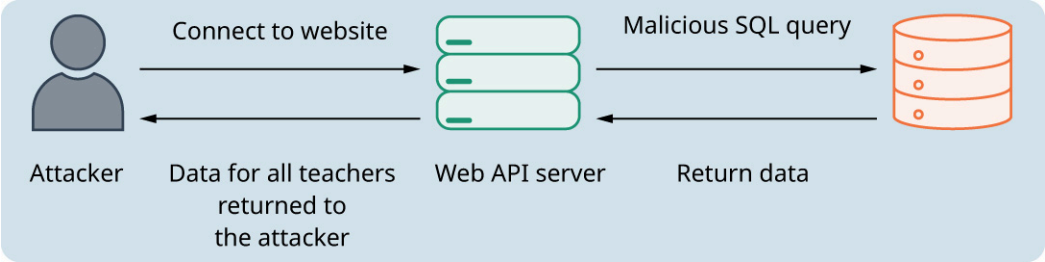


Figure 8.14 SQL injection gives attackers access to confidential client data such as teacher data. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

To secure the database, there are many methods such as discretionary access control, which may grant or reject object access using a set of policies; mandatory access control, which restricts the ability to grant or reject access to resource objects; statistical database security, which focuses on the protection of statistical values stored in the database; encryption, which converts the data into secret code to hide the true meaning; and public key infrastructure, which is a set of roles and policies needed to manage the database.

LINK TO LEARNING

There are many places on the Internet where you can learn more about programming and databases. For example, should you want to build your own website, you will need to know something about structured query language (SQL), which is a standard language for accessing and manipulating databases. Visit an introduction to SQL (https://openstax.org/r/76IntroSQL) to help you better understand SQL, what SQL can do, and how to use SQL when building a website.

8.4 Nonrelational Database Management Systems

Learning Objectives

By the end of this section, you will be able to:

- Discuss the various types of legacy database management systems
- Explain the functionality of non-first normal form (NFNF) database management systems
- Identify various characteristics of object databases and object persistence
- Differentiate the various relational database management system extensions
- Describe XML databases
- Summarize unstructured data and the advent of not only SQL (NoSQL) databases
- Differentiate cloud-related data management services and other types of databases

A **nonrelational database** is a database that does not use a traditional method for storing data such as rows and columns. Instead, nonrelational databases use a different storage model with specific requirements based on the type of data being stored. Relational databases typically store data in tables, with columns representing the attributes of the data and rows representing individual records. This structure is optimized for fast retrieval and manipulation of data using SQL queries. In contrast, nonrelational databases are optimized for storing and querying large amounts of data and are typically stored in a specialized format that allows for efficient indexing and querying. This format may include the use of columnar storage and compression techniques.

Legacy Databases

Flat file database and multifile relational database are the two main legacy DBMSs. A **flat file database** uses a simple structure to store data in a text file. Each line in the file is holding one record. A **multifile relational database** is more flexible than flat file structures as it provides more functionality for creating and updating data. It contains multiple tables of data with rows and columns that relate to each other.

Hierarchical Model

A **hierarchical model** is a model in which data are stored in the form of records and organized into a tree structure. The structure is parent-child and each parent may connect to one or more child nodes as in Figure 8.15. As an example, IBM IMS is a hierarchical database management software system for OLTP.

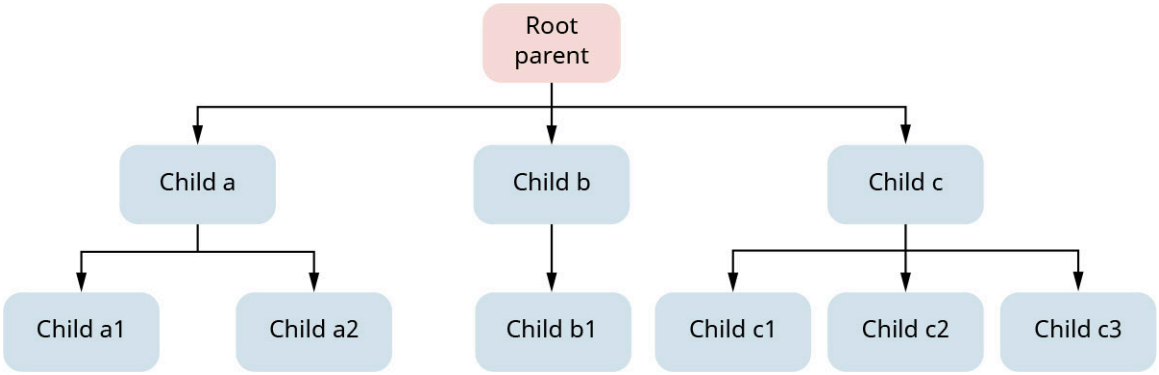


Figure 8.15 A hierarchical model shows a root parent connected to three child nodes. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

A hierarchical model has many limitations such as the complexity of navigating the system, its requirement that data be repetitively stored due to a treelike structure with a high level of redundancy, and its need for sequential searching.

Non-First Normal Form DBMSs

The database data model that does not meet any of the conditions of database normalization defined by the relational model is called a **non-first normal form (NFNF)**. However, data in this form may require complex

joins, which can make querying more complicated, although it does not lead to data duplication.

Object Databases and Object Persistence

An object-oriented database management system (OODBMS) offers unique object identifiers to access each object. The persistence is established with time and space, and the object exists even after its parent object has been deleted, which means it is persistent with time. An example of an OODBMS is db4o, which is an open-source, embeddable object database for Java and .NET environments. It allows developers to store and retrieve objects directly rather than having to map objects to a relational database schema.

Principles of Object Persistence

When an object is not deleted until a need emerges to remove it from memory, **object persistence** appears. In other words, the transient object is only needed during program execution and can be discarded when the program terminates. Persistence strategies include persistence by class, persistence by creation, persistence by marking, persistence by inheritance, and persistence by reachability.

In **persistence orthogonality**, the environment does not require any actions by a program to retrieve or save their state. In **persistence independence**, an object is independent of how a program manipulates it. There are many types of orthogonality. For example, all objects can be made persistent irrespective of their type or size, and they can achieve transitive persistence (refers to persistence by reachability). Persistent programming languages extend an OO language with a set of class libraries for object persistence. Serialization translates an object's state into a format that can be stored (for example, in a file) and reconstructed later.

OODBMSs and ODBs

OODBMSs appeared around 1985 and originated as extensions of OO programming languages. Object-oriented programmers store the developed products as objects and modify existing objects to make new objects within the OODBMS. OODBMSs store persistent objects in a transparent way, support persistence orthogonality, and guarantee the ACID properties. OODBMS has many limitations such as lack of a universal data model, lack of experience compared to RDBMS, lack of support, and complexity. The object data management group (ODMG) was created in 1991 and is based on object management group (OMG) standard; its main idea is to create a set of specifications to develop applications for an object database. There have been five revisions of ODMG and the last version is ODMG 3.0. Most mainstream database applications are built using an OO programming language in combination with an RDBMS rather than using an object-relational mapping (ORM).

INDUSTRY SPOTLIGHT

MongoDB

MongoDB is the most popular NoSQL database; it has delivered substantial values for some businesses that have been struggling to handle their unstructured data with the traditional RDBMS approach. After MetLife spent years trying to build a centralized customer database on a RDBMS that could handle all its insurance products, someone at an internal hackathon built one with MongoDB within hours, which went to production in 90 days.

Extended Relational Databases

Extended relational databases (ERDBMSs) combine characteristics of RDBMS and OODBMS. The products of ERDBMSs provide a relational data model and query language that have been extended to include features of OODBMSs.

Limitations of the Relational Model

The relational model has a flat structure, and expensive joins are needed to defragment the data before it can be successfully used, which increases the complexity of the objects due to normalization. Specialization, categorization, and aggregation cannot be directly supported. A tuple constructor can only be used on atomic values while a set constructor can only be used on tuples; however, both constructors are not orthogonal, cannot model behavior or store functions, and both provide poor support for multimedia.

Active RDBMS Extensions

A **trigger** is a statement consisting of declarative and/or procedural instructions and is stored in the catalog of the RDBMS. Triggers can also reference attribute types in other tables. Triggers are automatically executed when a triggering event occurs such as any change in the database. Triggers are similar to procedures stored in the database but differ in that they need to be explicitly invoked. Triggers are easy to code, are useful in the validation process, allow calling other procedures from the trigger, and allow recursion.

Object-Relational RDBMS Extensions

Object-relational DBMSs (ORDBMSs) keep the relation as the fundamental building block and SQL as the core DDL/DML, but they use the following OO extensions: user-defined types (UDTs), user-defined functions (UDFs), inheritance, behavior, polymorphism, collection types, large objects (LOBs), and recursive SQL queries.

TECHNOLOGY IN EVERYDAY LIFE

Nonrelational Database Management System

You are told that a new type of data structure has been unveiled and that it characterizes new forms of data being collected from sensors located at the edge of the network.

Would you conclude that a new kind of nonrelational database management system needs to be developed? What would be your approach for selecting the best possible type of nonrelational database management system for this application?

XML Databases

Extensible Markup Language (XML) is a markup language similar to HTML, but users define their tags; predefined tags such as HTML are not used. The oldest schema language for XML is the document type definition (DTD). An XML Schema is the metadata that describes the structure of an XML document. Extensible Stylesheet Language (XSL) defines the features and syntax of XML and consists of a language for transforming XML documents (XSLT) and an XML vocabulary for specifying formatting (XSL-FO). An XML namespace is a collection of names that can be used as element names in an XML document. XML Path Language (XPath) uses path expressions to select nodes in an XML document. An XML database is a data persistence system whereby the data are specified and stored in XML format. It is easy to code but has no universally accepted rules for the XML database.

Differences between XML and Relational Data

While XML data are hierarchical, relational data are represented in a model of relationships. XML data are self-describing, but relational data are not. XML data have inherent ordering, but relational data do not. RDBMS only supports atomic data types such as integer, string, and date. XML DTDs do not support atomic data types. XML Schema supports both atomic and aggregated types (aggregated types modeled in object-relational databases using user-defined types). XML data are semistructured (can include certain anomalies, and a change to DTD or XSD necessitates regeneration of tables). For example, suppose a banking application needs to transfer money from one account to another. The transaction may involve multiple steps, such as deducting

the amount from the sender's account and adding it to the recipient's account. If any of these steps fail, the entire transaction should be rolled back to ensure that the database remains consistent. Atomicity ensures that the transaction either completes successfully and all the changes are committed, or it fails and none of the changes are committed, leaving the database in a consistent state.

Mappings between XML and Object-Relational Data

An **ontology** is semantic data used to describe entities of the real world and the relationship between the entities using the Web Ontology Language (OWL) (Figure 8.16). To convert and map XML data into relational databases, we can use table-based mapping, which copies data from an external source such as XML into the selected table. A database schema is the structure of a database that defines how data are organized within a database. Schema-oblivious mapping stores both OWL class and instance data in a single table, which allows files to be easily parsed and loaded into the database. Schema-aware mapping maps the OWL schema into corresponding tables and then loads the instance data into the selected tables.

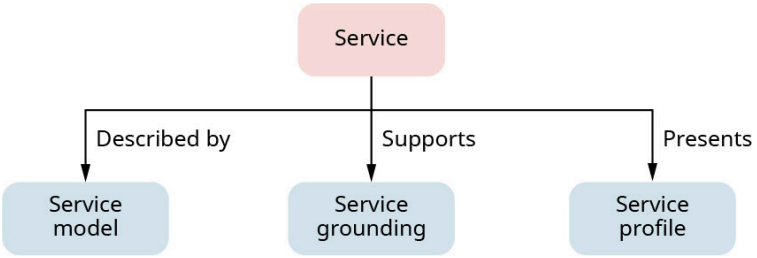


Figure 8.16 The service profile tells what the service does, the service grounding tells how to access it, and the service model tells how it works. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

GLOBAL ISSUES IN TECHNOLOGY

IT Solutions and Security Issues

IT solutions can help to address a range of security issues that organizations may face. Some common IT solutions for security issues include encryption and antivirus software.

Given the diversity of nonrelational database management systems and the fact that multiple systems may sometimes be used as part of the same solution, what are possible security issues that need to be considered?

Unstructured Data and NoSQL Databases

Unstructured data are data that are not arranged according to the data model and cannot be stored in a traditional relational database. NoSQL databases are schema-agnostic and provide the flexibility needed to store and manipulate large volumes of unstructured and semistructured data. Users do not need to know what types of data are stored during setup, and the system can accommodate changes in data types and schema. Designed to distribute data across different nodes, NoSQL databases are generally more horizontally scalable and fault-tolerant. MongoDB Atlas is an example of a NoSQL database system that is fully managed and operates on the cloud.

NoSQL DBMS is not ACID compliant and data consistency is not guaranteed. NoSQL DBMSs provide eventual consistency instead: when old data are getting overwritten, results that are a little wrong are temporarily returned. For example, Google’s search engine index cannot overwrite its data while people are simultaneously searching a given term so it does not give the most up-to-date results during a search, but it gives the best answer it can. While this setup does not work in situations where data consistency is necessary, such as financial transactions, it is just fine for tasks that require speed rather than pinpoint accuracy.

Key-Value Stores

A **key-value store** is a simple database that uses an associative array such as Redis, DynamoDB, and Cosmos DB. It stores only key-value pairs, provides basic functionality for retrieving the value associated with a known key, and works best with a simple database schema.

Tuple and Document Stores

A **tuple and document store** database stores data in XML or JSON format with the document name as key and the contents of the document as value.

```xml
<employee>
<fName>Salma</fName>
<lName>Issam</lName>
<hireYear>2021</hireYear>
</employee>
```

(a)

```json
{
  "employee": {
    "fName": "Salma",
    "lName": "Issam",
    "hireYear": 2021
  }
}
```

(b)

Figure 8.17 The same code written in (a) XML and (b) JSON shows the differences between the formats. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Documents can contain many different value types and can be nested, making them particularly well-suited to manage semistructured data across distributed systems. For example, MongoDB and Couchbase are document-oriented databases that store data in flexible, schemaless documents, which allows for a high degree of flexibility in how data are stored and queried. However, MongoDB uses a flexible document model that allows for complex nested structures and hierarchical data, while Couchbase uses a key-value data model with JSON documents. This makes MongoDB more suited for complex data structures and Couchbase more suited for simpler data structures.

Column-Oriented Databases

A **column-oriented database** stores data in column families or tables and are built to manage petabytes of data across a massive distributed system (e.g., Cassandra and Hbase). Both Cassandra and HBase use a column-family data model to store and manage data and are both NoSQL databases that are designed to handle large volumes of data. However, Cassandra uses a partitioned row-store data model, while HBase uses a column-oriented data model. This means that Cassandra is optimized for high-speed reads and writes of individual rows, while HBase is optimized for scanning large tables of data.

Graph-Based Databases

A **graph-based database** represents data as a network of related nodes or objects to facilitate data visualizations and graph analytics. The graph may represent any relationship such as 1:1, 1:N, and N:M. It is useful for analyzing the relationships between heterogeneous data points, such as in fraud prevention or Facebook's friend graph.

Other NoSQL Databases

There are many NoSQL databases such as XML databases, OO databases, database systems to deal with time series and streaming events, and database systems to store and query geospatial data (Spatial data refer to data that are associated with a specific location or geographic area. Spatial data can be managed and analyzed using a geographical information system.). There are also database systems such as BayesDB. BayesDB is a probabilistic programming platform that is based on a Bayesian approach to modeling data relationships. The structure of BayesDB can be broken down into several key components, including data tables and models that lets users query the probable implication of their data.

Transaction Management and Concurrency in NoSQL Databases

With the increasing amount of data, the number of parallel transactions has increased. In this case, the capacity can be increased by extending the system capabilities (vertical scaling) or arranging multiple servers in a cluster (horizontal scaling). NoSQL databases distribute data over a cluster of database nodes for the sake of performance and availability.

In many NoSQL implementations (e.g., Cassandra, Google's BigTable, Amazon's DynamoDB). all nodes implement the same functionality and are able to perform the role of request coordinator using a membership protocol. The membership protocol checks the availability of each node, which makes it easier to apply index between them but does not guarantee that every node is aware of every other node at all times. Hashing is the process of transforming a key to another value to implement a hash table. Consistent hashing avoids mapping each key to a new node in case nodes are added or deleted. Consistent hashing uses a modulo operator (%) to distribute keys over servers. The syntax is (H(key) = key % N) where N is the number of servers. Eventual consistency means the data and their replicas become consistent at some point in time after each transaction. NoSQL databases are categorized as eventual consistency because it guarantees up-to-date information; therefore, many NoSQL databases guarantee so-called eventual consistency based on the CAP theorem. The **CAP theorem** states that a distributed computer system cannot attain the following three properties concurrently: consistency (all nodes see the same data at the same time), availability (guarantees that every request receives a response, indicating a success or failure result), and partition tolerance (the system continues to work even if nodes go down or are added).

THINK IT THROUGH

C++ and Databases

You are asked to develop a program in C++ that needs to store and retrieve data for later use.

Which type of nonrelational database management system would you recommend for this application?

Query Languages and APIs for NoSQL Databases

Processing an XML within a database application requires parsing the document and then processing the code using an API such as Document Object Model (DOM API). DOM API is a tree-based API (i.e., the memory representation is a tree). Figure 8.18 shows an example of XML code and the corresponding DOM tree.

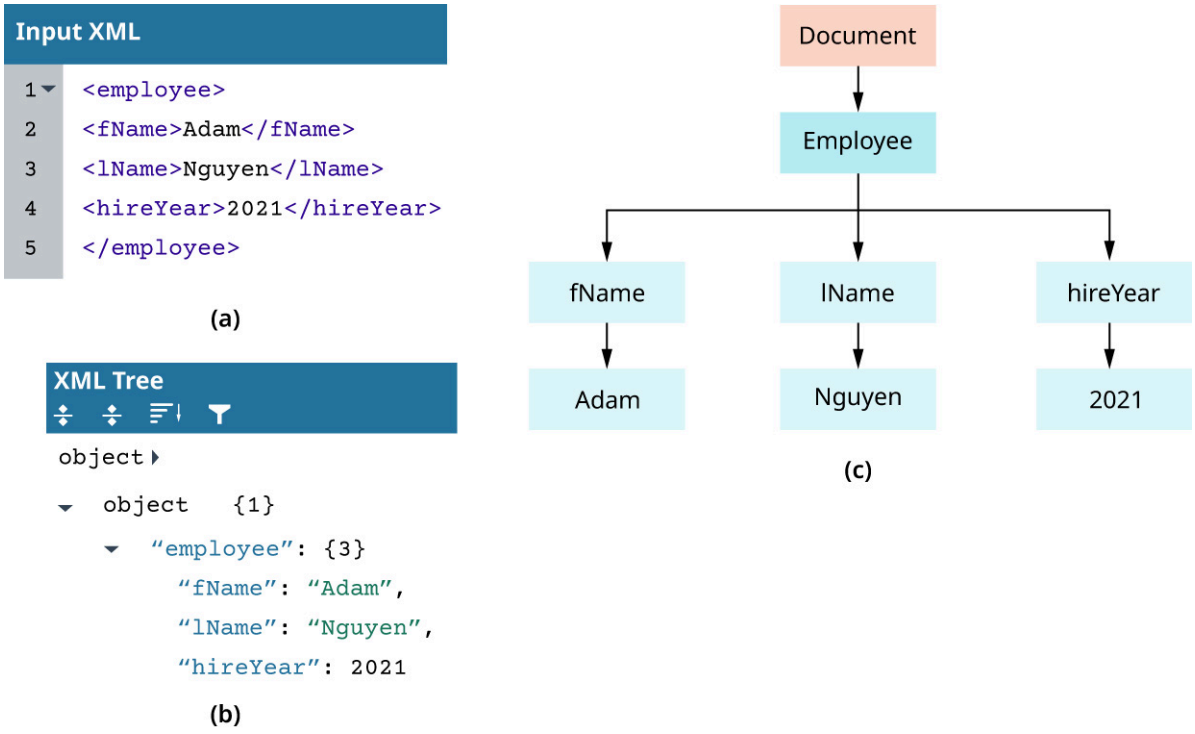


Figure 8.18 This figure visualizes the tree-based DOM API: (a) shows the Input XML code, (b) highlights in-memory tree representation, and (c) demonstrates the corresponding DOM tree. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

NoSQL databases use filters to apply the simple query; that is, they use variables in the query that are populated on the server. Additionally, most NoSQL databases support complex queries using aggregation with MapReduce. **MapReduce** is an open-source software framework used to apply complex queries. It implements a map function, which is the conversion of a dataset into a tuple, and reduces function, which reduces the map into a smaller set.

Cloud and Other Data Management Services

Data as a Service (DaaS) is a data management technique that uses the cloud to store, process, and manage data. DaaS uses web services to handle data located on the Internet. Many vendors provide DaaS such as Azure, AWS, GCP, and IBM Cloud. Amazon Web Service (AWS) offers many solutions for DaaS, such as Aurora, RDS, and Redshift for relational database and DynamoDB for the key-value database.

Cloud Data as a Service (DaaS)

DaaS is a data management strategy that includes many technologies such as information life cycle solutions, data modeling, replication, and content management. DaaS is on-demand and subscription-based, which means the customer only pays for the services they need. The functionalities of DaaS are data quality, storing, managing, securing, and analyzing data. One of the key advantages of cloud-based database systems is their ability to be "elastic" and provide compute and storage resources on-demand. This means that as the size of a dataset grows or decreases as more or fewer users access the database, the system can quickly scale up or down its resources to handle the demand.

Blockchain DBMSs

The technology that records transitions securely is called blockchain. It has the world's attention because of its use of cryptocurrencies such as Bitcoin. In health care, blockchain technology can be used to create a secure and transparent system for managing health-care data. This can help to improve patient privacy, reduce fraud, and improve the efficiency of health-care systems. A **blockchain DBMS** is a database that stores data as a block data structure, and each block is connected to other blocks by providing cryptographic security and

immutability.

LINK TO LEARNING

One of the most popular NoSQL databases is the DB-Engines Ranking (https://openstax.org/r/76DBEngines) which ranks database management systems according to their popularity. The ranking is updated monthly. You can read about the method of calculating the scores.

8.5 Data Warehousing, Data Lakes, and Business Intelligence

Learning Objectives

By the end of this section, you will be able to:

- Outline the characteristics of data warehouses
- Explain the extraction, transformation, and loading (ETL) process
- Discuss data marts
- Describe the technology behind virtual data warehouses and data marts
- Summarize the nature of operational data stores
- Identify data lakes and their functionality
- Compare business intelligence and related tools

With the proliferation of data, new techniques that store and handle the data are required. Data warehousing and data lakes are used for storing big data. Business intelligence analyzes gathered data in data warehouses and data lakes to improve strategic decision-making.

Data Warehouse Characteristics

In the late 1980s, the concept of data warehouse started at IBM when researchers Barry Devlin and Paul Murphy developed the first business warehouse. A **data warehouse** centralizes an enterprise's data from its databases. It supports the flow of data from operational systems to analytics/decision systems by creating a single repository of data from various sources both internal and external. In most cases, a data warehouse is a relational database that stores processed data that are optimized for gathering business insights. It collects data with predetermined structures and schema coming from transactional systems and business applications, and the data are typically used for operational reporting and analysis. A data warehouse is a collection of data designed to support management's decision-making process and is meant to be:

- Subject-oriented: data are organized based on subjects such as products.
- Integrated: it integrates data from multiple resources and different formats.
- Time-variant: it stores a time series of periodic snapshots to always be up-to-date.
- Nonvolatile: data are read-only to complete the process of updating or removing data.

Data warehouse supports the types of decision-making at the operational, tactical, and strategic levels. Operational decisions happen frequently based on day-to-day operations and are structured based on a specific predefined role (i.e., limited). Tactical-level decisions occur with greater frequency (i.e., monthly) and are semistructured based on the data requirement. The strategic level has the highest level of organizational business decisions, and their decisions are unstructured and/or infrequent.

In designing a data warehouse, many schemas can be adopted such as star schema, snowflake schema, and fact constellation. A **star schema** is a data model with one large fact table connected to smaller tables. The fact table contains keys referring to each dimension table as shown in Figure 8.19.

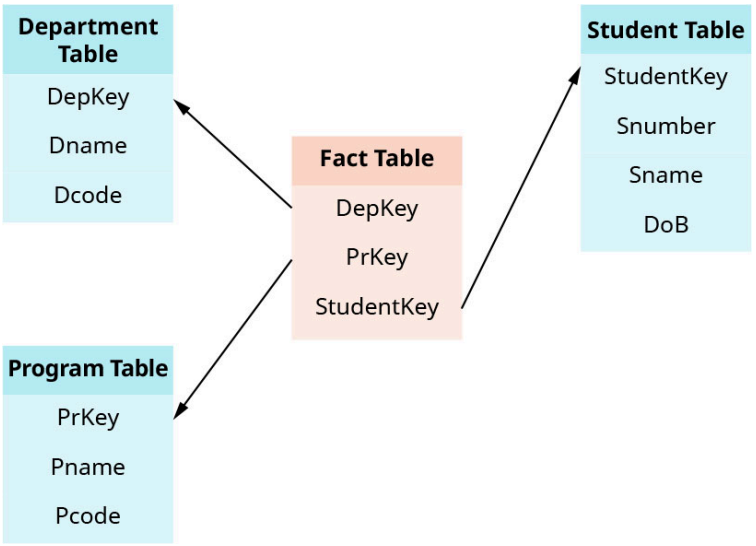


Figure 8.19 A graphic shows an example of a star schema. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The **snowflake schema** is a data model that normalizes the dimension table (Figure 8.20). It has one fact table, and it creates smaller tables with primary–foreign key relationships. A **fact constellation** has more than one fact table connected to other smaller dimension tables.

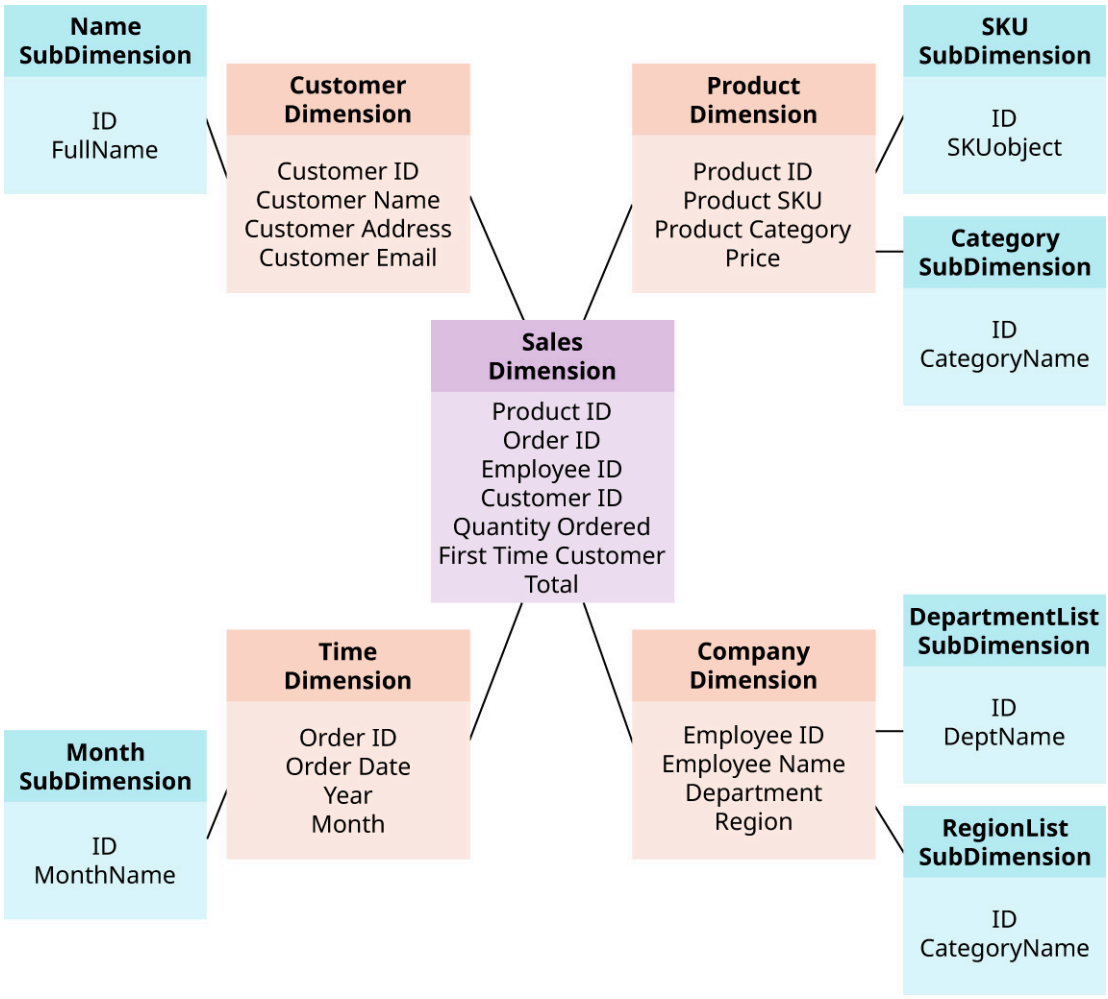


Figure 8.20 This graphic is an example of a snowflake schema. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Extraction, Transformation, and Loading Process

After designing the schema, the next process to start is **extraction, transformation, and loading (ETL)**, which is data integration that combines data from multiple sources, fixes the data format, and loads the data into a data warehouse. The first step is extracting the data from the system, which can be full or incremental extraction. Extraction uses **change data capture (CDC)**, which is a technology that detects any data update event.

The second step is transforming the data, which includes formatting the data to be consistent, cleansing the data to get rid of missing data, aggregating data by merging some attributes, and enriching by adding external data. The last step is loading the data in parallel as fact and dimension tables to the data warehouse as shown in Figure 8.21. ETL steps will make changes to the data, which should be documented for better understanding and future maintenance. The documentation includes structural metadata about the data structure and semantic metadata about the meaning.

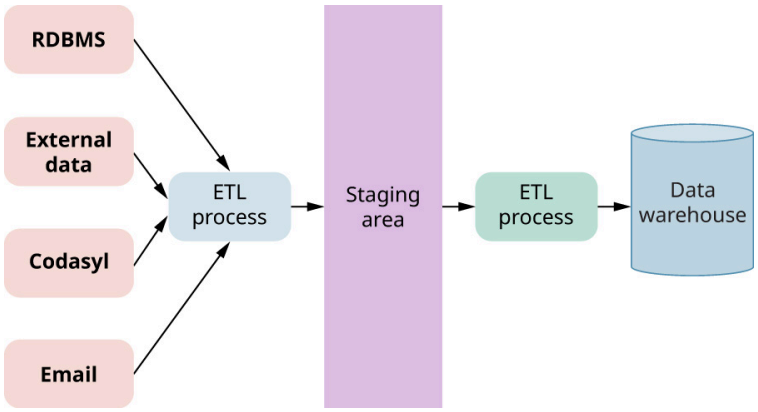


Figure 8.21 The extraction, transformation, and loading (ETL) steps of data are shown as the data makes its way to the data warehouse. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Data Marts

A **data mart** is a scaled-down version of a data warehouse aimed at meeting the information needs of a homogeneous small group of end users such as a department or business unit. It provides focused content and improves query performance. Dependent data marts pull their data directly from a central data warehouse. Independent data marts are stand-alone systems drawing data directly from the operational systems, external sources, or a combination of both.

GLOBAL ISSUES IN TECHNOLOGY

Storing Data

For the past decade, the trend has been to store data in any shape or form without knowing whether it will be used in the future. This has been facilitated by the advent of cloud technology that has allowed people to store unlimited amounts of data at low cost into data lakes. The amount of data stored will keep increasing over the next few decades as daily usage of computer processes keeps increasing.

What are some of the global issues you foresee with today's ability to store an unlimited amount of data that may be analyzed at a later time?

Virtual Data Warehouses and Virtual Data Marts

A virtual data mart has no physical data but provides a single point of access to a set of underlying physical data stores; data are only accessed ("pulled") at query time. A **virtual data warehouse** can be built as a set of

SQL views directly on the underlying operational data sources as an extra layer on top of a collection of physical independent data marts. The metadata model contains the schema mappings between the schemas of the underlying data stores and the schema of the virtual data warehouse (involves query reformulation). A **virtual data mart** is usually defined as a single SQL view. There can be virtual-independent versus virtual-dependent data marts. Disadvantages are the extra processing capacity from the underlying (operational) data sources and the fact that it is not possible to keep track of historical data.

Operational Data Stores

An **operational data store (ODS)** is a staging area that provides query facilities. It is good for analysis tools that need data that are closer to real time. More complex analyses are still conducted on the actual data warehouse.

Data Lakes

A **data lake** stores all of an enterprise's structured and unstructured data at any scale. Data lakes are large data repositories that store raw data and can be set up without having to first define the data structure and schema. They allow users to run analytics without having to move the data to a separate analytics system, enabling businesses to gain insights from new sources of data not available for analysis before. For example, by building machine learning models using data from log files (e.g., log files generated by an application that writes to operating system event), click-streams (e.g., information about visited pages on the Web), social media (e.g., LinkedIn), and IoT devices (e.g., smartwatch). By making all of the enterprise data readily available for analysis, computer/data scientists can answer a new set of business questions or tackle old questions with new data. Figure 8.22 shows an example of a data lake process.

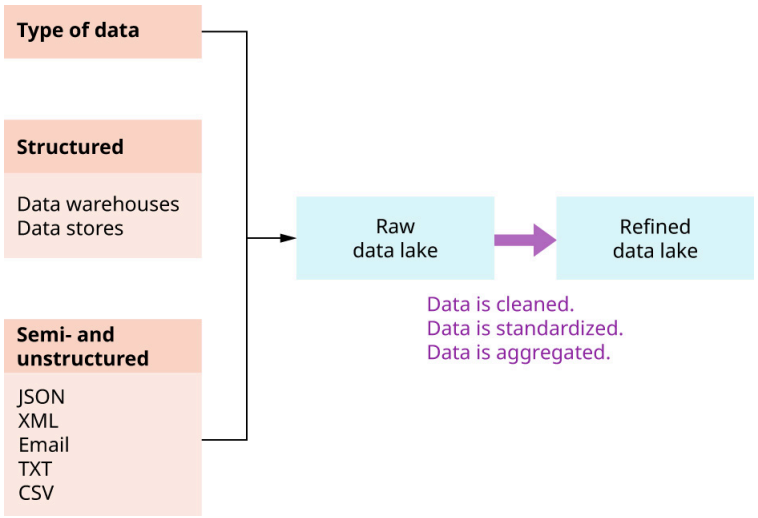


Figure 8.22 A data lake stores structured, semistructured, and unstructured data. The store process stores the data as raw data lake then applies many processes such as cleansing and aggregation to load the data as a refined data lake. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Data Warehouses vs. Data Lakes

A data lake is an architectural approach specifically designed to handle data of every variety, ingestion velocity, and storage volume. A data lake allows the storage of massive amounts of data into a central location so it's readily available to be categorized, processed, analyzed, and consumed by diverse groups within an organization. Because data can be stored as-is, there is no need to convert it to a predefined schema as typically required in traditional RDBMS-driven architectures. Consumer usage patterns and the sourcing of the data itself directly influence how data are collected, stored, processed, moved, transformed, automated, and visualized. Data are the ultimate asset with boundless usage patterns now being generated and consumed by humans, machines, devices, sensors, and applications. There are some differences between data warehouse and data lakes listed in Table 8.11.

Characteristic	Data Warehouse	Data Lake
Data	Relational	Nonrelational and relational
Schema	Schema-on-write Designed before the implementation	Schema-on-write Written at the time of analysis
Storage	Expensive	Low cost
Performance	Fastest query results	Query results getting faster
Users	Decision-makers	Data scientist
Analysis	Batch reporting, business intelligence, and visualization	Machine learning, predictive analysis

Table 8.11 Difference between a Data Warehouse and a Data Lake

Data Lake Development

A traditional approach to information management is no longer suitable due to cost and the inability to adapt. Seventy percent of development costs from ETL include an effort to consolidate, prepare, standardize, and transform data for downstream analytics. Costs involve initial capital investment for hardware, software licensing for databases, data integration, and analytics platforms. There is a need to react to emerging changes in the proliferation of data, new and emerging technologies, and cloud-based integrated service platforms. Traditional data storage and analytic tools can no longer provide the agility and flexibility required to deliver relevant business insights and competitive advantage. Figure 8.23 shows the timeline of a data lake.

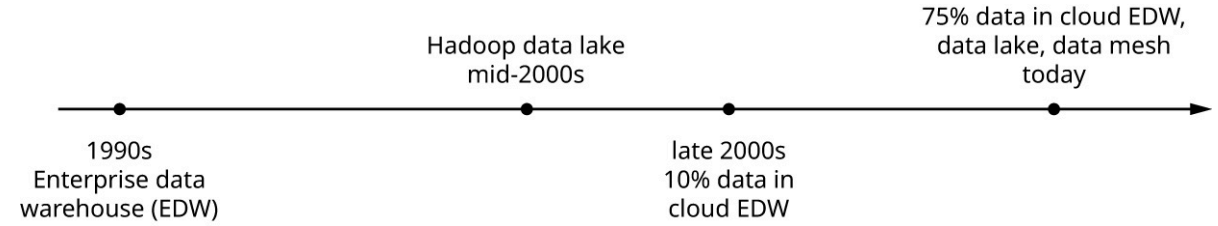


Figure 8.23 An illustration of a data lake shows its evolution from an on-premises tool to a cloud version to a hybrid model. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

From Data Lakes to Data Swamps

A common challenge with the data lake architecture is that without the appropriate data quality and governance framework in place, when terabytes of structured and unstructured data flow into the data lakes, it often becomes extremely difficult to sort through their content. Data lakes can turn into data swamps as the stored data become too messy to be usable. A **data swamp** is data stored without organization to make retrieval easy. Many organizations are now calling for more data governance and metadata management practices to prevent data swamps from forming.

LINK TO LEARNING

Data lakes are used to collect very large amounts of both relational and nonrelational data, such as a streaming service collecting data on users' watching habits, while data warehousing collects relational data, such as when a retailer collects data to use to analyze shoppers' habits and to control inventory.

Where does business intelligence fit into this data system? Business intelligence analyzes this data to help in making decisions. They use data for marketing and targeted sales in particular.

It can be confusing to understand the ins and outs of each process and how they relate to each other. A blog post about the difference between data lakes, data warehouses, and databases (https://openstax.org/r/76DataSystems) does a great job of distinguishing each and how each contributes to the use of understanding data.

Business Intelligence

The main goal of the data warehouse is to support decision-making. In order to do that, we need intelligent tools. The set of activities, techniques, and tools aimed at understanding patterns in past data to predict the future is called **business intelligence (BI)**. Modern BI is facilitated through Tableau (https://openstax.org/r/76Tableau), ClickView (https://openstax.org/r/76ClickView), and Microsoft Power BI. The quality of data controls the quality of results, which means that bad data gives bad insights (remember GIGO). BI techniques include query and reporting, which provide a graphical user interface (GUI) in which the user can graphically design a report and pivot table, which is a data summarization tool. In addition, one of the BI tools is online analytical processing (OLAP). OLAP is a computer-based approach to analyzing data that enables users to extract and view data from multiple dimensions. It is used to analyze large volumes of data and is commonly used in BI applications. OLAP allows users to perform complex analysis of data by providing a multidimensional view of the data. It can aggregate data across multiple dimensions, such as time, location, product, and customer, enabling users to see trends and relationships that might be hidden in a traditional two-dimensional view of the data.

8.6 Data Management for Shallow and Deep Learning Applications

Learning Objectives

By the end of this section, you will be able to:

- Define big data and explain its related functionality
- Discuss big data analytics and its impact on computer systems
- Identify and describe the tools that are used to perform shallow machine learning
- Describe cognitive analytics and artificial intelligence
- Identify the tools that are used to perform deep learning
- Explain massively parallel processing (MPP) database management systems

In the last few years, the amount of data has increased exponentially in many businesses to be big data. Storing, analyzing, and retrieving data has become complex and challenging. Traditional databases and storage cannot deal with big data, which creates new software and hardware business opportunities. Data integration aims to provide a unified view and/or unified access over heterogeneous, and possibly distributed, data sources. Process integration deals with the sequencing of tasks in a business process but also governs data flows in these processes. Both data and processes are considered in data integration. The emergence of BI and analytics triggered the need to consolidate data into a data warehouse.

Big Data

The term *big data* has been in use since the early 1990s and the term was credited to computer scientist John R. Mashey who is considered the father of big data. Big data is a high volume of data in the shape of structure, semistructured, or unstructured data. Every minute, more than 300,000 tweets are created, Netflix subscribers stream more than 70,000 hours of video at once, Apple users download 30,000 apps, and Instagram users like almost two million photos. In this section, we study the five Vs of big data, big data examples, new sources of data, data and process integration, data quality, and data governance, as well as the privacy, security, and

ethical use of data.

The Five Vs of Big Data

Big data helps the decision-makers to improve their decisions, which in turn improves the quality of their products and services. Big data has many characteristics. Researchers defined the scope of big data as five Vs (Figure 8.24):

- **volume**: the amount of data; also referred to as data at rest
- **velocity**: the speed at which data comes in and goes out; data in motion
- **variety**: the range of data types and sources that are used; data in its many forms
- **veracity**: the uncertainty of the data; data in doubt
- **value**: the actual value derived using the total cost of ownership (TCO) and return on investment (ROI) of the data

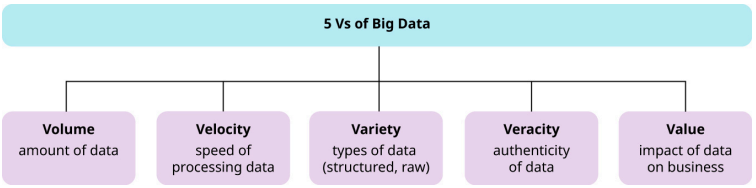


Figure 8.24 The five Vs of big data are volume, velocity, variety, veracity, and value. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

New Sources of Data

IoT sources become more numerous and diverse every day. Other sources of data are network data, publicly available data, macroeconomic data, textual data, audio, images, videos, fingerprint, location (GPS), geospatial, RFID data, and many more. We will cover the IoT in more details in Chapter 13 Hybrid Multicloud Digital Solutions Development.

Data and Process Integration

Traditionally, a business has various departments that have independent data and systems (i.e., silo) from each other. Departments such as human resources and accounting did not integrate and do not share available resources, which makes it hard to answer queries or check updates. To start the convergence of analytical and operational data requires **data integration**, which provides a consistent view of all organization data. There are different data integration patterns such as data consolidation, data federation, and data propagation. The use of ETL to capture data from multiple sources and integrate it into a single store such as a data warehouse is **data consolidation**. The use of enterprise information integration (EII) to provide a unified view over data sources is **data federation**. The use of enterprise application integration (EAI) corresponding to the synchronous or asynchronous propagation of updates in a source system to a target system is **data propagation**.

A more recent approach to data integration is **data virtualization**, which is a technique that hides the physical location of data and uses data integration patterns to produce a unified data view. The aim is not only to integrate the data, but also to integrate the process. Process integration aims to integrate the procedures within the business process to improve performance. The main challenge of combining traditional data types with new data types is integrating the diverse data types in such a way that they can be processed and analyzed efficiently.

Data Quality and Master Data Management

Data quality involves various criteria to assess the quality of a dataset. We need to guarantee a high-quality level to achieve accurate results (GIGO). A **data quality dimension** includes accuracy, completeness, consistency, and accessibility. The causes of data quality issues are often deeply rooted within core organizational processes and culture. Data preprocessing activities are corrective measures for dealing with

data quality issues. Transparent and well-defined collaboration between data stewards and data owners is key to sustainably improving data quality. Data integration can both improve and hamper data quality (e.g., environments where different integration approaches have been combined, leading to a jungle of systems). The series of processes, policies, standards, and tools to help organizations define and provide a single point of reference for all data that are mastered is **master data management (MDM)**. Setting up an MDM initiative involves many steps and tools, including data source identification, mapping out the systems architecture, constructing data transformation, cleansing and normalization rules, providing data storage capabilities, monitoring, and governance facilities. MDM is the generic term of enterprise data management (EDM). EDM is a data management platform for validating the customer data; it is mainly associated with securities data.

Data Governance

Organizations are increasingly implementing company-wide data governance initiatives to govern and oversee data quality and data integration. In data governance, an organization aims to set up a company-wide controlled and supported approach toward data quality that is accompanied by data quality management processes (i.e., managing data as an asset rather than a liability). Different frameworks and standards have been introduced for data governance. A well-articulated data governance program is a good starting point. Approaches include centralized (i.e., central department of data scientists handling all analytics requests), decentralized (i.e., all data scientists are directly assigned to business units), and mixed (i.e., centrally coordinated center of analytical excellence with analytics organized at the business unit level). Businesses should aim for top-down, data-driven culture to catalyze trickledown effects. The board of directors and senior management should be actively involved in analytical model building, implementation, and monitoring processes.

Data governance has different standards such as Total Data Quality Management (TDQM), Capability Maturity Model Integration (CMMI), Data Management Body of Knowledge (DMBOK), Control Objectives for Information and Related Technology (COBIT), and Information Technology Infrastructure Library (ITIL). TDQM is a model that defines, measures, and improves the quality of the data in the organization. CMMI is a model that helps organizations to improve the reliability by decreasing risks in services and products by developing the system behaviors. DMBOK is a collection of best practices for each data management process such as data modeling, data quality, documentation, data security, and metadata. COBIT is a framework that helps organizations that are looking to improve and monitor their data management system. ITIL is a framework that improves efficiency to achieve the predictable services using standardized design.

Privacy and Security of Data

The concept of **data security** pertains to the following concerns: guaranteeing data integrity and data availability (i.e., ensuring that data are accurate and available when needed), authentication (i.e., verifying the identity of a user), access control (i.e., controlling who has access to data and what actions they can perform), guaranteeing confidentiality (i.e., ensuring that data are kept confidential and are not disclosed to unauthorized users), auditing (i.e., tracking all access and activity related to data), and vulnerabilities (i.e., identifying vulnerabilities in systems and data). To better understand privacy, we should start with the RACI (responsible, accountable, consulted, and informed). *Responsible* defines who is responsible for developing the data. *Accountable* defines the people who decide what should be done with the data. *Consulted* defines domain expertise to advise the data scientist. *Informed* defines a set of people who should be up-to-date on the working process.

To access internal data, a data scientist should file a data access request to specify the target data and the length of time needed for access. There are many available privacy regulations such as the General Data Protection Regulation (GDPR), Privacy Act of 1974, Health Insurance Portability and Accountability Act (HIPAA) of 1996, the Electronic Communications Privacy Act (ECPA) of 1986, and the Privacy Shield. The Privacy Shield is a framework for exchanges of personal data between the European Union and the United States.

Big Data Analytics

While storage and computing needs have grown by leaps and bounds in the last several decades, traditional hardware has not advanced enough to keep up. Enterprise data no longer fits neatly into standard storage, and the computational power required to handle most big data analytics tasks may take weeks or months, or may be impossible to complete. To overcome this deficiency, many new technologies have evolved to include multiple computers working together, distributing the database to thousands of commodity servers. When a network of computers is connected and works together to accomplish the same task, the computers form a cluster. A **cluster** can be thought of as a single computer but can dramatically improve the performance, availability, and scalability over a single, more powerful machine at a lower cost by using commodity hardware.

CONCEPTS IN PRACTICE

Private Data and Analytics

All social media and search engine websites use data management for machine and deep learning applications. Today there are major concerns related to the use of private data to perform analytics and support targeted sales or broadcast fake news to select customers. Being able to control the use of private data is essential so it does not get manipulated and end up being misused for various reasons. Data compliance is attempting to address this very problem and various regulations have already been put in place in the United Kingdom and Europe via the GDPR regulations. The same is happening in the United States, with regulations being put in place in some states including California in particular.

Analytics Process Model

An **analytics process model** provides a statistical analysis using a set of processes to solve system problems and find a new market opportunity. There are many sample applications for analyzing data such as risk analytics (e.g., credit scoring, fraud detection), marketing analytics (i.e., using data to evaluate the success of marketing strategies), response modeling (i.e., statistical platform to model the relationship between the customers' responses and the predicted values), customer segmentation (i.e., dividing the customers into groups with each group sharing the same characteristics to improve the marketing strategy), recommender systems (i.e., a filtering system that provides suggestions for products based on customer rating), and text analytics (i.e., identifying a pattern to understand the data).

Creating a viable data management infrastructure for analytics applications involves much more than just building a simple machine learning model. It requires an understanding of how all the parts of the enterprise's ecosystem work together—where/how the data flows into the data team, the environment where the data are processed/transformed, the enterprise's conventions for visualizing/presenting data, and how the model output will be converted as input for some other enterprise applications. The main goal is to build a process that is easy to maintain and where models are iterated on, the performance is reproducible, and a model's output can be easily understood and visualized for other stakeholders so that they may make informed business decisions. Achieving those goals requires selecting the right tools as well as an understanding of what others in the industry are doing along with best practices. Figure 8.25 shows the three-stage process of the analytics process model, which we discuss next.

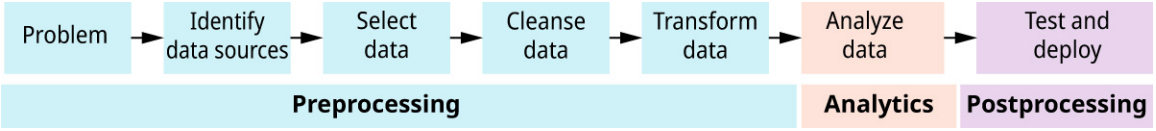


Figure 8.25 The three-stages of the analytics process model are shown. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Data Preprocessing

To preprocess the data, a data scientist may use denormalizing, the merging process, sampling, exploratory analysis, missing values, and outlier detection and handling. The process of merging several normalized data tables into an aggregated, denormalized data table is called **denormalizing**. The **merging process** involves selecting information from different tables about a specific entity and then copying it to an aggregated table. Selecting a subset of historical data to build an analytical model is called **sampling**. The process of summarizing and visualizing data for initial insight is called **exploratory analysis**. Filling the empty field or deleting it involves resolving a **missing value**. An **outlier** is a value outside the population that should be detected in order to apply the handling process on it.

Types of Analytics

After finishing the preprocessing, the analytic step will start (Figure 8.26). This step aims to extract a decision model from the preprocessed data. To build such a model, there are many models, such as predictive analytics, evaluating predictive models, descriptive analytics, and social network analytics.

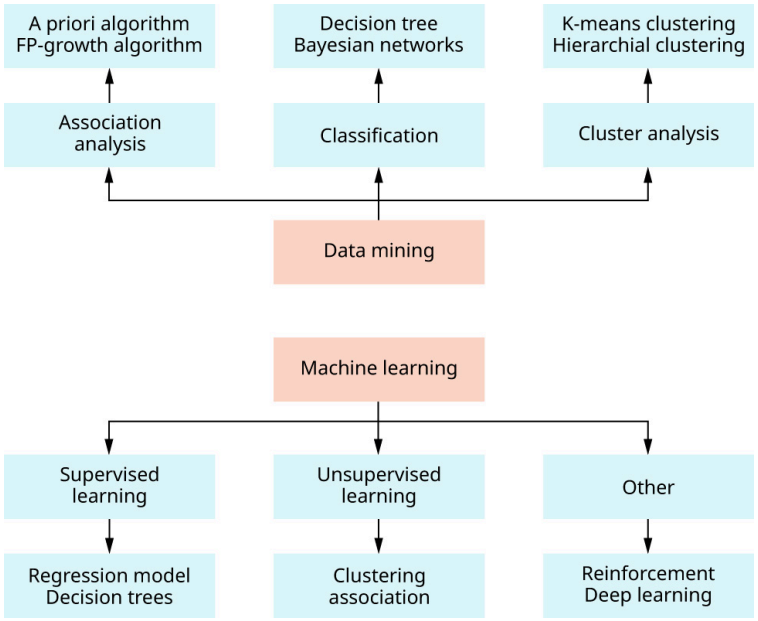


Figure 8.26 Many techniques can be used to analyze data in order to make a business decision. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Predicting the target measure of interest using regression (e.g., linear regression, logistic regression) and classification (e.g., decision trees) is **predictive analysis**. Evaluating predictive models splits up the dataset with specific performance measures. Patterns of customer behavior (e.g., association rules, sequence rules, and clustering) are considered **descriptive analytics**.

GLOBAL ISSUES IN TECHNOLOGY

Predictive Analytics

Predictive analytics is a set of techniques that uses data, statistics, and machine learning to make predictions about future events and/or behaviors. These predictions rely on data being accurate and unbiased to provide value. As we move toward generative artificial intelligence (GenAI) technology that makes use of existing data to create large language models, issues associated with the possibility of using biased data to train these models are becoming critical. Recent research is focusing on fairness in the decisions taken by models trained with biased data, and on designing methods to increase the

transparency of automated decision-making processes so that possible bias issues may be easily spotted and “fixed” by removing bias.

Postprocessing of Analytical Models

The last step is the postprocessing step, and the first activities in this step are interpretation and validation. Business experts validate the data and detect any unknown pattern. In addition, sensitivity analysis takes place in postprocessing to verify the robustness of the created model. After that, experts approve the deployment of the model and the production activity can start. Finally, the expert applies the backtesting activity to be sure the model produces the correct output.

Evaluating Analytics

Analytics models should solve the business problem for which it was developed (business relevance) and should be acceptable statically (statistical performance and validity). The analytical model should be understandable to the decision-maker (interpretability) and operationally efficient. Measuring the model performance uses TCO and ROI. The **total cost of ownership (TCO)** represents the cost of owning and operating the analytical model over time. The calculation of **return on investment (ROI)** determines the ratio of net profits divided by the investment of resources.

THINK IT THROUGH

Machine Learning

You are given a dataset that pertains to hospital patients affected with a particular condition and you are asked to create a predictive model that could be used to assess testing new individuals for this condition.

Would you use shallow or deep machine learning to solve this problem? Explain your dataset and logic for solving the health-care problem.

Big Data Analytics Frameworks for Shallow Machine Learning

Big data analytics creates a model from a set of data, and to learn from the data, we need machine learning (ML) algorithms. This section discusses MapReduce, Hadoop framework, SQL on Hadoop, Apache Spark framework, streaming big data analytics, on-premises versus cloud solutions, and searching unstructured data and enterprise search.

MapReduce

MapReduce is a two-step computational approach for processing large (multiterabyte or greater) datasets distributed across large clusters of commodity hardware in a reliable, fault-tolerant way (Figure 8.27). The first step is distributing data across multiple computers (Map) with each performing a computation on its slice of the data in parallel. The next step combines those results in a pair-wise manner (Reduce).

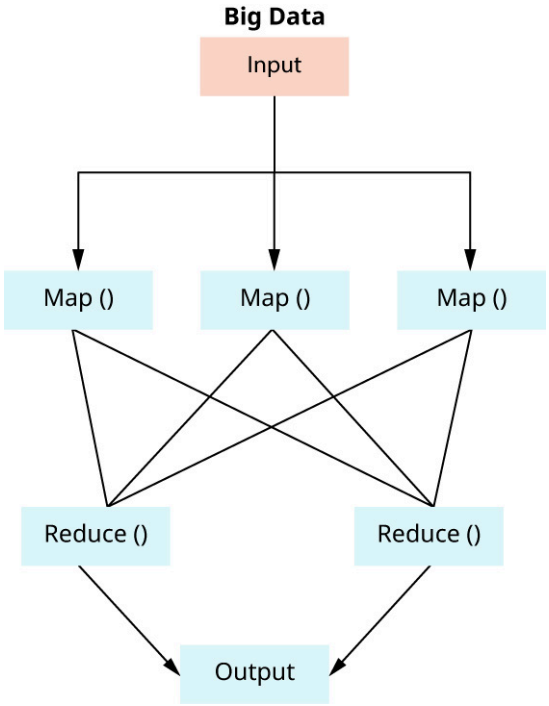


Figure 8.27 After the data input stage, Map will start then Reduce to produce the output. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Hadoop Framework

Hadoop is distributed data infrastructures that leverage clusters to store and process massive amounts of data. An open-source software framework used for distributed storage and processing of big datasets can be set up over a cluster of computers built from normal, commodity hardware. Many vendors such as Amazon, Cloudera, Dell, Oracle, and Microsoft offer their implementation of a Hadoop stack.

Hadoop leverages distribution and consists of three main components:

- Hadoop Distributed File System (HDFS): A way to store and keep track of data across multiple (distributed) physical hard drives
- MapReduce: A framework for processing data across distributed processors
- Yet Another Resource Negotiator (YARN): A cluster management framework that orchestrates the distribution of CPU usage, memory, and network bandwidth allocation across distributed computers

Hadoop is built for iterative computations. It scans massive amounts of data in a single operation from disk, distributes the processing across multiple nodes, and stores the results back on disk. Hadoop is typically used to generate complex analytics models or high-volume data storage applications, such as retrospective and predictive analytics that involve analyzing past data to identify trends, machine learning and pattern matching that involve using algorithms to automatically identify patterns and trends in data, customer segmentation and churn analysis that involve dividing customers into groups based on shared characteristics or behaviors and using this information to better understand their needs and preferences, and active archives that involve storing data in a way that allows it to be easily accessed and used for analytics purposes.

SQL on Hadoop

Because of the complexity of MapReduce in a database query, many industries create other solutions. In 2007, Hadoop included the first version of Hbase as a data storage platform. HBase offers a simplified structure and query language for big data. Similar to an RDBMS, HBase organizes data in tables with rows and columns. Yahoo developed Pig, which is a high-level platform for creating programs that run on Hadoop, and its language is Pig Latin, which uses MapReduce underneath. It somewhat resembles the querying facilities of

SQL. Facebook developed Hive, which is a data warehouse solution offering SQL querying facilities on top of Hadoop. It converts SQL-like queries to a MapReduce pipeline. It also offers a JDBC and ODBC interface and can run on top of HDFS as well as other file systems.

Apache Spark Framework

MapReduce processes data in batches; therefore, it is not suitable for processing real-time data. Apache Spark is a parallel data processing tool that is optimized for speed and efficiency by processing data in-memory. It operates under the same MapReduce principle but runs much faster by completing most of the computation in memory and only writing to disk when memory is full or the computation is complete.

Streaming Big Data Analytics

Data streams come from devices, sensors, websites, social media, and applications. Streaming analytics performs an analytic process on streaming data, which is useful for real-time flow of data. There are many big data streaming analytics platforms such as Amazon Kinesis Data Firehose, which is a streaming data service to capture, process, and store data streams at any scale. The Array of Things (AoT) is an open-source network that collects and returns urban data in real time. Azure Stream Analytics is a real-time analytics event-processing engine to analyze big data streaming from multiple sources.

On-Premises vs. Cloud Solutions

Another innovation that has completely transformed enterprise big data analytics capabilities is the rise of cloud services. Before cloud services were available, businesses had to buy on-premises data storage and analytics solutions from software and hardware vendors, pay up front for perpetual software license fees and annual hardware maintenance, pay service fees along with the costs of things such as power, cooling, security, disaster protection, and IT staff for building and maintaining the on-premises infrastructure. Even when it was technically possible to store and process big data, most businesses found it cost prohibitive to do so at scale. Scaling with on-premises infrastructure also requires an extensive design and procurement process, which takes a long time to implement and requires substantial up-front capital. Many potentially valuable data collection and analytics possibilities were ignored as a result. Some key benefits of cloud computing include scalability, flexibility, cost savings, reliability, and disaster recovery.

LINK TO LEARNING

Businesses can significantly reduce costs and improve operational efficiencies with cloud services because they can develop and produce their products more quickly with the out-of-the-box cloud resources with built-in scalability. Cloud services remove the up-front costs and time commitment to build on-premises infrastructure. Cloud services also lower the barriers to adopt big data tools, which has the effect of democratizing big data analytics for small and midsize businesses. Using Cloud services allows start-up and small companies to develop and scale solutions quickly, making it possible for them to compete with larger organizations. You can discover how companies and their data scientists use the cloud (https://openstax.org/r/76CloudServices) to deploy data science solutions to production or to expand computing power.

Searching Unstructured Data and Enterprise Search

Searching for information in documents using retrieval models that specify matching functions and query representation is **information retrieval**. Enterprises use a variety of retrieval models for intranet, web search, and analysis such as keyword queries that use a keyword to retrieve documents, Boolean queries that use logical operators to retrieve the documents, phrase queries that perform exact phase retrieval, proximity queries that check how close to each other within a record multiple entities are, wild card queries that support matching expressions, and natural language queries that try to formulate answers to a specific question from

retrieved results. Searching unstructured data is challenging. A **full-text search** is selecting individual text documents from a collection of documents according to the presence of a single or a combination of search terms in the document. Indexing full-text documents is the process of adding an index for every search term that consists of term and pointer, with each pointer referring to a document that contains the term. Web search engines search a web database and gather information related to a specific term. An **enterprise search** is the process of making content stemming from the databases by offering tools that can be used within the enterprise.

Cognitive Analytics and Artificial Intelligence

The technology that tries to simulate a human's way of solving problems (e.g., Siri and Alexa) is called **cognitive computing**. Artificial intelligence (AI) is a system that creates intelligent ways to solve problems that previously required human interaction.

Cognitive Computing

Three components must interact to achieve AI (Figure 8.28): syntax (structure), semantics (meaning), and inference (reasoning/planning). Human intelligence requires that these three components rely on some form of data management and relate to deep learning of other brain functions such as restricted Boltzmann machines, stacked autoencoder, and deep belief networks. Restricted Boltzmann machines are artificial networks that can learn a probability distribution using a set of input. Stacked autoencoder is an artificial network used to learn unlabeled data through efficient coding. Deep belief networks are intelligent networks used to invent a solution for a specific problem when the traditional intelligent network could not solve it.

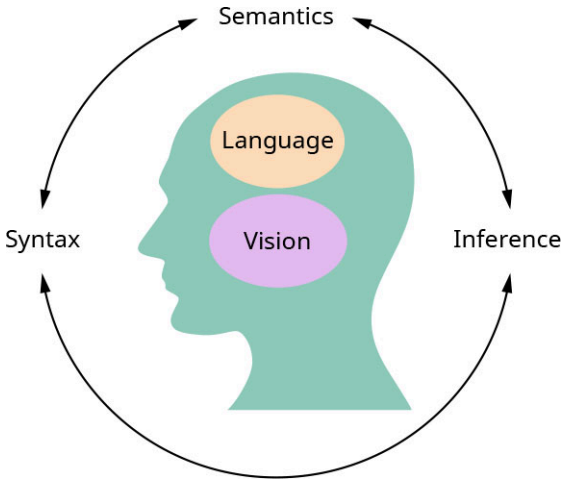


Figure 8.28 Components of artificial intelligence include syntax, semantics, and inference. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Sample AI Applications

Amazon provides various ML services. They allow developers to integrate Cloud ML into mobile and other applications. Amazon Lex allows users to incorporate voice input into applications (extension of Amazon's Echo product) so that users can ask questions about everything from the weather to news and streaming music. Amazon Polly is the opposite of Lex; Polly turns text into speech in 27 languages. Amazon Rekognition, which is at the cutting edge of deep learning applications, takes an image as input and returns a textual description of the items that it sees in that image by performing detailed facial analysis and comparisons.

Reinforcement and Transfer Learning

The machine learning method based on encouraging desired behaviors and removing undesired behaviors is called **reinforcement**. The machine learning method based on reusing the result of a specific task to start a new task is called **transfer learning**. A **deep learning network** is a type of machine learning method based on artificial neural networks (Figure 8.29). Artificial neural networks simulate the network of neurons to make a

computer learn and make decisions like the human brain does (e.g., recurrent neural networks or RNNs, which are an artificial neural network, uses time series data). There are various classes of deep learning networks such as:

- Cloud-based deep learning frameworks such as Microsoft Cognitive Toolkit, which is an open-source toolkit for commercial-grade distributed deep learning
- Google TensorFlow system applications for neural network computing and deep learning such as handwritten digit recognition and cognitive services
- Predictive software libraries for cognitive applications such as Keras, which is an open-source software library that provides a Python interface for artificial neural networks

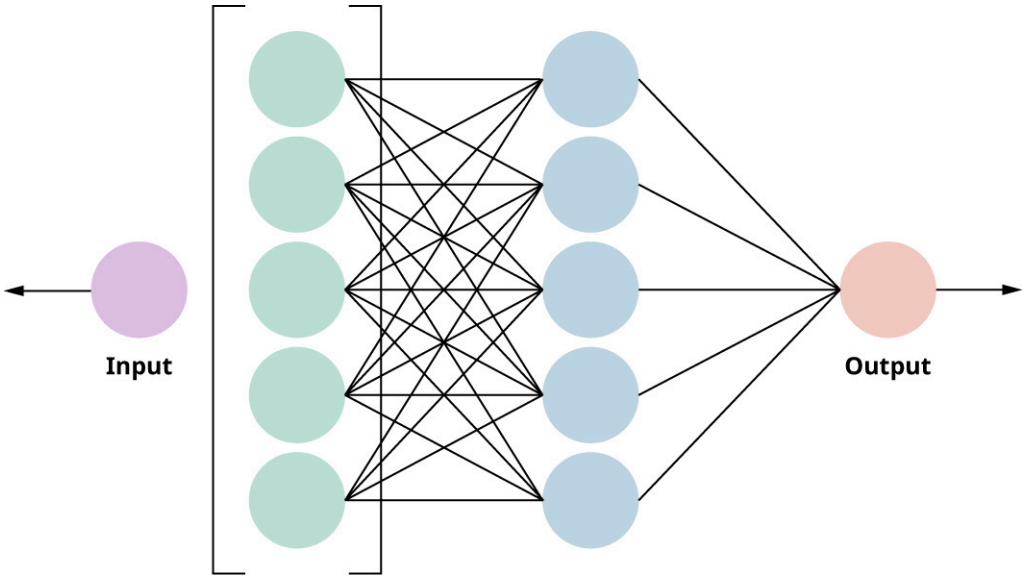


Figure 8.29 The deep learning process uses artificial neural networks. (credit: modification of "NeuralNetwork" by Loxaxs/Wikimedia Commons, CC0)

Cognitive Analytics Frameworks for Deep Machine Learning

In this section, we introduce examples of Cognitive analytics frameworks for deep machine learning applications such as Spark MLlib, Amazon Machine Learning Platform and MXNet, Google TensorFlow, Azure Machine Learning Platform, and Microsoft Cognitive Toolkit.

Spark MLlib

Spark ML includes a High-level API (Spark Machine Learning Library of MLlib) for creating ML pipelines. Data can be fed into data frames, and the library enables the quick creation of a machine learning processing pipeline by combining transformers and estimators.

MXNet

MXNet is an open-source library for distributed parallel machine learning that was developed at Carnegie Mellon University, the University of Washington, and Stanford University. MXNet can be programmed with Python, Julia, R, Go, Matlab, or C++ and runs on many different platforms including clusters and GPUs; it is also now the deep learning framework of choice for Amazon.

Google TensorFlow

Google's TensorFlow is a frequently discussed and used deep learning toolkit; if you have installed the Amazon Deep Learning AMI, you already have TensorFlow installed, and you can begin experimenting right away.

Azure Machine Learning Platform and Microsoft Cognitive Toolkit

The Microsoft Cognitive Toolkit software is available for download in a variety of formats so that deep learning

examples can be run on Azure as clusters of Docker containers (i.e., multiple nodes joined using a special configuration).

INDUSTRY SPOTLIGHT

MPP Is NYSE VIP

The New York Stock Exchange (NYSE) receives 4 to 5 TB of data daily and conducts complex analytics, market surveillance, capacity planning, and monitoring. It had been using a traditional database that could not handle the workload; it took hours to load and had poor query speed. Moving to an MPP database reduced their daily analysis run time by eight hours.

Massively Parallel Processing (MPP) Databases

Similar to MapReduce, massively parallel processing (MPP) databases are referred to as NewSQL as opposed to NoSQL. MPP distributes data processing across multiple nodes, and the nodes process the data in parallel for faster speed. Unlike Hadoop, MPP is used in RDBMS and utilizes a "share-nothing" architecture. Each node processes its own slice of the data using multicore processors, making them many times faster than traditional RDBMS. Some MPP databases, such as Pivotal Greenplum, have mature machine learning libraries that allow for in-database analytics. In an MPP system, all the nodes are interconnected and data could be exchanged across the network.[3] However, as with traditional RDBMS, most MPP databases do not support unstructured data, and even structured data will require some processing to fit the MPP infrastructure. Therefore, it takes additional time and resources to set up the data pipeline for an MPP database. Because MPP databases are ACID-compliant and deliver much faster speed than traditional RDBMS, they are usually employed in high-end enterprise data warehousing solutions such as Amazon Redshift, which is a data warehouse cloud platform for Amazon Web Services, and Pivotal Greenplum, which is a big data technology based on MPP architecture combined with an open-source database.

8.7 Informatics and Data Management

Learning Objectives

By the end of this section, you will be able to:

- Define informatics and discuss its applications in various industries
- Explain the role and life cycle of information systems

As discussed earlier, the data analysis process produces reports that help the decision-maker at all management levels. Informatics is collaborating activities that involve humans and technologies to apply data management tools.

Definition and Applications

The term **informatics** broadly describes the study, design, and development of information technology for the good of people, organizations, and society. Informatics focuses on computer systems from a user-centered perspective and studies the structure, behavior, and interactions of natural and artificial systems that store, process, and communicate information.

Informatics has applications in many areas including sports informatics, behavior informatics, business informatics, privacy informatics, community informatics, geoinformatics, health informatics, imaging informatics, museum informatics, research informatics, social informatics, and urban informatics.

3 IBM. Parallel processing technologies. Last updated March 21, 2023. Available at https://www.ibm.com/docs/en/iis/11.5?topic=topologies-parallel-processing.

Informatics as a Data Management Solution

Informatics leverages information sciences (big data analytics, electronic records management, digital assets management, and information security and governance), human computer interaction (human-centered design), information system analysis and design, telecommunications structure, and information architecture and management. All these fields are supporting components of end-to-end data management solutions. Therefore, one can view informatics as a data management solution.

Informatics Information Systems

Information systems support informatics and provide an organizational context for using database systems to help collect, organize, store, analyze, preserve, retrieve, and govern data and records relevant to an organization. These systems also help informatics professionals turn data and information into actionable knowledge from a user-centered perspective within the context of specific disciplines or industry such as health or sports. Helping uncover critical information that can improve the user experience may ultimately lead to achieving a company's goals and is an example of how valuable informatics is to a business.

THINK IT THROUGH

Informatics Approach

Suppose you work for a health-care provider and you have been tasked with developing and launching a new patient portal to help patients better communicate with health-care providers while providing patients easy access to their health-care data.

What will your technical requirements be in relation to software, hardware, data, and network? What will you need to support an informatics solution? What approach will you take?

Information System Creation Life Cycle

Information systems provide resources involved in the collection, management, use, and dissemination of information resources of organizations. The **macro life cycle** for the creation of an information system includes feasibility analysis, requirements collection and analysis, design, implementation, and validation and acceptance testing. The **micro life cycle** for the creation of an information system focuses on system definition, database design, database implementation, loading or data conversion, application conversion, testing and validation, operation, monitoring, and maintenance.

Chapter Review

Key Terms

analytics process model provides a statistical analysis using a set of processes to solve system problems and find a new market opportunity
archival backup storing the data in different servers/sites
asynchronous call client sends a request without waiting for the response
atomicity, consistency, isolation, durability (ACID) properties that impose a number of constraints to ensure that stored data are reliable and accurate
attribute column header
blockchain DBMS database that stores data as a block data structure and each block is connected to other blocks by providing cryptographic security and immutability
business intelligence (BI) set of activities, techniques, and tools aimed at understanding patterns in past data to predict the future
CAP theorem states that a distributed computer system cannot guarantee consistency, availability, and partition tolerance at the same time
centralized DBMS architecture data are maintained on a centralized server at a single location
change data capture (CDC) technology that detects any data update event and keeps track of versions
cloud DBMS architecture DBMS and database are hosted by a third-party cloud provider
cluster single computer that can dramatically improve the performance, availability, and scalability over a single, more powerful machine and at a lower cost by using commodity hardware
cognitive computing technology tries to simulate human's way in solving problems
column-oriented database database that stores data in column families or tables and is built to manage petabytes of data across a massive, distributed system
computer scientist person who has theoretical and practical knowledge of computer science.
concurrency control coordination of transactions that execute simultaneously on the same data so that they do not cause inconsistencies because of mutual interference
connection manager manages reports, books, objects, and batches
data information and facts that are stored digitally by a computer
data accuracy whether the data values stored for an object are the correct values and are often correlated with other DQ dimensions
Data as a Service (DaaS) data management technique that uses the cloud to store, process, and manage data
data completeness degree to which all data in a specific dataset are available with a minimum percentage of missing data
data compliance process that ensures that data practices align with external legal requirements and industry standards
data consistency keeping data consistent as it moves between various parts of the system
data consolidation use of ETL to capture data from multiple sources and integrate it into a single store such as a data warehouse
data control language (DCL) language used to control access to data stored in a database
data description language (DDL) language used to create and modify the object structure in a database
data description language (DDL) compiler translates statements in a high-level language into low-level instructions that the query evaluation engine understands
data dictionary set of information describing the contents, format, and structure of a database
data federation use of enterprise information integration (EII) to provide a unified view over data sources
data governance set of clear roles, policies, and responsibilities that enables the enterprise to manage and safeguard data quality
data integration providing a consistent view of all organization data

data lake large data repository that stores raw data and can be set up without having to first define the data structure and schema
data management study of managing data effectively
data manipulation language (DML) language used to manipulate and edit data in a database
data mart scaled-down version of a data warehouse aimed at meeting the information needs of a homogeneous small group of end users
data model abstract model that contains a set of concepts to describe the structure of a database, the operations for manipulating these structures, and certain constraints that the database should obey
data owner person with the authority to ultimately decide on the access to, and usage of, the data
data propagation use of enterprise application integration (EAI) corresponding to the synchronous or asynchronous propagation of updates in a source system to a target system
data quality (DQ) measure of how well the data represents its purpose or fitness for use
data quality dimension includes accuracy, completeness, consistency, and accessibility
data query language (DQL) language used to make various queries in a database
data redundancy happens when the same piece of data is held in two separate places in the database
data replication storing data in more than one site to improve the data availability and retrieval performance
data scientist person who has theoretical and practical knowledge of managing data
data security pertains to guaranteeing data integrity, guaranteeing data availability, authentication, access control, guaranteeing confidentiality, auditing, mitigating, and vulnerabilities
data steward person who ensures that the enterprise's actual business data and the metadata are accurate, accessible, secure, and safe
data swamp data stored without organization to make retrieval easy
data virtualization technique that hides the physical location of data and uses data integration patterns to produce a unified data view
data warehouse centralizes an enterprise’s data from its databases; it supports the flow of data from operational systems to analytics/decision systems by creating a single repository of data from various sources both internal and external
database administrator (DBA) person responsible for the implementation and monitoring of a database and ensuring databases run efficiently
database application program or piece of software designed to collect, store, access, retrieve, and manage information efficiently and securely
database architecture representation of the design that helps design, develop, implement, and maintain the DBMS
database designer person responsible for creating, implementing, and maintaining the database management system
database language used to write instructions to access and update data in the database
database management system (DBMS) approach where metadata are stored in a catalog
database normalization process of structuring a relational database to reduce data redundancy and improve data integrity
database recovery activity of setting the database in a consistent state without any data loss in the event of a failure or when any problem occurs
database security using a set of controls to secure data, guaranteeing a high level of confidentiality
database transaction sequence of read/write operations considered to be an atomic unit
database user person with the privileges to access, analyze, update, and maintain the data
DBMS interface main line of communication between the database and the user
DBMS utility utility for managing and controlling database activities such as loading utility, reorganization utility, performance-monitoring utilities, user management utilities, backup and recovery utility
deep learning network machine learning method based on artificial neural networks
denormalizing process of merging several normalized data tables into an aggregated, denormalized data

table

descriptive analytics patterns of customer behavior

disk storage memory device that stores the data such as hard disks, flash memory, magnetic disks, optical disks, and tapes

distributed transaction set of operations that are performed across multiple database systems

domain constraint defines the domain of values for an attribute

enterprise search process of making content stemming from databases by offering tools that can be used within the enterprise

entity integrity constraint specifies that no primary key contains a null

equi-join join that combines tables based on matching values in specified columns

exploratory analysis process of summarizing and visualizing data for initial insight

extraction, transformation, and loading (ETL) data integration that combines data from multiple sources, fixes the data format, and loads the data into a data warehouse

fact constellation more than one fact table connected to other smaller dimension tables

fat client variant where presentation logic and application logic are handled by the client; common in cases where it makes sense to couple an application's workflow

federated DBMS provides a uniform interface to multiple underlying data sources

flat file database database that uses a simple structure to store data in a text file; each line in the file holds one record

full-text search selection of individual text documents from a collection of documents according to the presence of a single or a combination of search terms in the document

functional dependency (FD) constraint that specifies the relationship between two sets of attributes and provides a formal tool for the analysis of relational schemas

garbage in, garbage out (GIGO) quality of output is determined by the quality of the input

graph-based database database that represents data as a network of related nodes or objects to facilitate data visualizations and graph analytics

Hadoop distributed data infrastructures that leverage clusters to store and process massive amounts of data

heuristics optimization mathematical technique for processing a query quickly

hierarchical DBMS data model in which the data are organized into a treelike model, DML is procedural and record-oriented, the query processor is logical, and internal data models are intertwined

hierarchical model model in which data are stored in the form of records and organized into a tree structure

horizontal fragmentation (sharding) rows that satisfy a query predicate, global view with UNION query, and common in NoSQL databases

immediate backup storing the copies in disks

in-memory DBMS stores all data in internal memory instead of slower external storage

indexed organization uses a key, similar to relative organization, but the key is unique and fixed

informatics study, design, and development of information technology for the good of people, organizations, and society

information architect (also, *data architect* or *information analyst*) a person responsible for designing the conceptual data model (blueprints) to bridge the gap between the business processes and the IT environment

information retrieval searching for information in documents using retrieval models that specify matching functions and query representation

inner join represents the intersection of two tables

key constraint specifies that all the values of the primary key must be unique

key-value store simple database that uses an associative array such as Redis, DynamoDB, and Cosmos DB

logical data independence separates any changes in the data from the data format

logical design designing a database based on a specific data model but independent of physical details

macro life cycle includes feasibility analysis, requirements collection and analysis, design, implementation, and validation and acceptance testing

MapReduce open-source software framework used to apply complex queries
master data management (MDM) series of processes, policies, standards, and tools to help organizations define and provide a single point of reference for all data that are mastered
merging process selection of information from different tables about a specific entity and copying it to an aggregated table
metadata modeling business presentation of metadata
micro life cycle focuses on system definition, database design, database implementation, loading or data conversion, application conversion, testing and validation, operation, monitoring, and maintenance
miniworld (also, *universe of discourse [UoD]*) represents some aspect of the real-world data that is stored in a database
missing value filling an empty field or deleting the field
mixed fragmentation combines horizontal and vertical fragmentation
multifile relational database database that is more flexible than flat file structures by providing more functionality for creating and updating data
multimedia DBMS provides storage of multimedia data such as text, images, audio, and video
multiuser DBMS allows many users to use the database concurrently
multivalued dependency (MVD) occurs when two attributes in a table are independent of each other but both depend on a third attribute
n-tier DBMS multitier architecture that usually divides an application into three tiers
natural join creates an implicit join based on the common columns in two tables
network DBMS data are organized into a network model, DML is procedural and record-oriented, the query processor is logical, and internal data models are intertwined
non-first normal form (NFNF) database data model that does not meet any of the conditions of database normalization defined by the relational model
nonrelational database database that does not use a traditional method for storing data such as rows and columns
NoSQL DBMS big unstructured data classified as document, graph, key-value stores, and column-oriented databases
object persistence refers to when an object is not deleted until a need emerges to remove it from memory
object-oriented DBMS data model in which the data are organized into an OO data model as no impedance mismatch in combination with the OO host language
online analytical processing (OLAP) focuses on using operational data for tactical or strategical decision-making
online transaction processing (OLTP) focuses on managing operational or transactional data; the database server must be able to process lots of simple transactions per unit of time
ontology semantic data used to describe entities of the real world and the relationship between the entities using the Web Ontology Language (OWL)
open-source DBMS publicly available DBMS that can be extended by anyone
operational data store (ODS) staging area that provides query facilities
optimizer process of selecting the best plan to execute
outer join union of two tables
outlier value that is outside the population that should be detected in order to apply the handling process on it
parallel processing technique in which multiple processors work simultaneously on different tasks or different parts of a task to enable concurrent processing of large amounts of data
persistence independence when an object is independent from how a program manipulates it
persistence orthogonality concept means that the environment does not require any actions by a program to retrieve or save their state
physical data independence separates the conceptual level from the physical level
physical database design attributes logical concepts to physical constructs

predictive analytics predicts the target measure of interest using regression and classification
primary key special unique identifier for each table record
query processor acts as an intermediary between users and the DBMS data engine to communicate query requests including DML compiler, query parser, query rewriter, query optimizer, and query executor
query tree example of data structure representation for the relational algebra expression
query-by-example (QBE) database query language for relational databases based on domain relational calculus
redundant array of inexpensive disks (RAID) stores information across an array of low-cost hard disks
reinforcement machine learning method based on encouraging desired behaviors and removing undesired behaviors
relation mathematical concept based on the ideas of sets
relational algebra query language that uses unary or binary operators to perform queries
relational database design (RDD) models data into a set of tables with rows and columns
relational DBMS data model in which the data are organized into a relational data model, use SQL as a declarative and set-oriented database, the query processor has a strict separation between the logical and internal data model
relative organization when each record is assigned a numeric key to rearrange the order of the records at any time
return on investment (ROI) ratio of net profits divided by the investment of resources
sampling selecting a subset of historical data to build an analytical model
security manager collection of processes used to secure the database from threats
semistructured data data that are not organized in a formatted database but have some organized properties
sequential file organization records are organized in the order stored and any new record is added at the end
single-user DBMS only one user at a time can use the database
snowflake schema data model that normalizes the dimension table
spanned record when all records are classified into blocks and the length of the record can exceed the size of a block
star schema data model with one large fact table connected to smaller tables
storage manager program that is responsible for editing, storing, updating, deleting, and retrieving data in the database such as transaction manager, buffer manager, lock manager, and recovery manager
structured data data that have been organized into a formatted database and have relational keys
Structured Query Language (SQL) programming language used in programming and managing structured data located in an RDBMS
synchronous call when the client sends a request and waits for a response from the service
tablespace where tables are stored physically in the memory
theta join allows merging two tables based on a theta condition
thin client variant where only the presentation logic is handled by the client and applications and database commands are executed on the server; it is common when application logic and database logic are tightly coupled or similar
total cost of ownership (TCO) cost of owning and operating the analytical model over time
transaction set of database operations induced by a single user or application that should be considered as one undividable unit of work
transaction management delineating transactions within the transaction life cycle
transfer learning machine learning method based on reusing the result of a specific task to start a new task
translation process of translating from high-level language to machine language
trigger statement consisting of declarative and/or procedural instructions and stored in the catalog of the RDBMS
tuple one row with a collection of values separated by a comma and enclosed in parenthesis

tuple and document store database that stores data in XML or JSON format with the document name as key and the contents of the document as value
uniqueness constraint specifies that all the tuples must be unique
unstructured data data that are not organized in a formatted database and do not have organized properties
value actual value derived using the total cost of ownership (TCO) and return on investment (ROI) of the data
variety range of data types and sources that are used; data in its many forms
velocity speed at which data comes in and goes out; data in motion
veracity uncertainty of the data; data in doubt
vertical fragmentation subset of columns of data, global view with JOIN query, and useful if only some of a tuple's attributes are relevant to a node
virtual data mart usually defined as a single SQL view
virtual data warehouse can be built as a set of SQL views directly on the underlying operational data sources as an extra layer on top of a collection of physical independent data marts
volume amount of data; data at rest
weak entity type of entity that cannot be uniquely identified based on its attributes alone and must rely on a strong entity to provide the context necessary for identification
XML DBMS data model in which the data are using the XML data model to store data

Summary

8.1 Data Management Focus

- To make a decision, data should be formatted and converted to information. Knowledge comes after processing the information.
- Metadata are data about data and are stored in catalogs. The catalog provides an important source of information for end users.
- Data quality represents the measure of how well the data represents its purpose. A data quality framework categorizes the different dimensions of data quality such as intrinsic, contextual, representation, and access.
- Data governance is a set of clear roles, policies, and responsibilities that enables the enterprise to manage and safeguard data quality.
- There are various data management roles: information architect, database designer, data owner, data steward, database administrator, computer scientist, and data scientist.
- The data management road map has multiple steps, starting from collecting and storing the data to having a final product or decision.

8.2 Data Management Systems

- To store, retrieve, edit, and maintain the related data in the database, we need a system that is a database management system (DBMS).
- A database can be defined as a collection of related data items within a specific business process or problem setting.
- A DBMS is the software package used to define, create, use, and maintain a database while considering appropriate security measures.
- There are many characteristics for DBMSs such as loose coupling, efficiency, consistency, and maintenance.
- A DBMS includes various components such as DBMS interface, connection manager, security manager, DDL compiler, query processor, storage manager and DBMS utilities.
- Logical data model categories include hierarchical DBMSs, network DBMSs, relational DBMSs, object-oriented DBMSs, XML DBMSs, and NoSQL DBMSs.
- DBMS users may be divided into actors on the scene and workers behind the scene.
- There are various types of database architectures such as centralized DBMS architecture, client server

DBMS architecture, n-tier DBMS architecture, cloud DBMS architecture, federated DBMS, and in-memory DBMS.

8.3 Relational Database Management Systems

- The relational model of data is based on the mathematical concept of a relation.
- Relational database management systems (RDBMSs) are one type of DBMS that stores related data elements in a row-based table structure.
- SQL is a language used in programming and managing structures data located in a RDBMS. SQL is based on relational algebra with many extensions.
- Relational algebra is a query language that uses operators to perform queries.
- The logical design is designing a database based on a specific data model but independent of physical details.
- Database normalization is the process of structuring a relational database to reduce data redundancy and improve data integrity.
- Relational database design (RDD) models data into a set of tables with rows and columns. Each row represents a record, and each column represents an attribute.
- Database tables are stored in a disk storage such as hard disks, flash memory, magnetic disks, optical disks, and tapes.
- File organization and indexing are used to minimize the number of block accesses for frequent queries, and the most popular are sequential, relative, and indexed organization.
- API technologies represent database-related entities in an OO way.
- Concurrency control is the coordination of transactions that execute simultaneously on the same data so that they do not cause inconsistencies due to mutual interference.
- Data replication is the storage of data in more than one site to improve the data availability and retrieval performance.
- Database recovery is the activity of setting the database in a consistent state without any data loss in the event of a failure or when a problem occurs.
- Database security uses a set of controls to secure data and guarantee a high level of confidentiality.

8.4 Nonrelational Database Management Systems

- A nonrelational database is a database that does not use traditional ways for storing data.
- Flat file databases and multifile relational databases are the two main legacy DBMS.
- A hierarchical model is a model in which data are stored in the form of records and organized into a tree structure.
- Non-first normal form (NFNF) is a database data model that does not meet any of the conditions of database normalization defined by the relational model.
- Object persistence appears when an object is not deleted until a need emerges to remove it from the memory.
- Persistence independence means that an object is independent of how a program manipulates it.
- The relational model has a flat structure, and expensive joins are needed to defragment the data before it can be successfully used, which increases the complexity of the objects due to the normalization.
- An XML database is a data persistence system in which the data are specified and stored in XML format.
- Mapping strategies to map XML data into relational databases are table-based mapping, schema-oblivious mapping, and schema-aware mapping.
- Unstructured data are managed by key-value stores, tuple and document stores, column-oriented databases, graph-based databases, and other NoSQL databases.
- DaaS is a data management strategy that includes many of technologies such as information life cycle solutions, data modeling, replication, and content management.

8.5 Data Warehousing, Data Lakes, and Business Intelligence

- A data warehouse is a relational database that stores processed data that are optimized for gathering

business insights to support decision-making process.

- In designing a data warehouse, many schemas can be adopted such as star schema, snowflake schema, and fact constellation.
- ETL is the data extraction, transformation, and loading process.
- A data mart is a scaled-down version of a data warehouse aimed at meeting the information needs of a homogeneous small group of end users.
- Virtualization uses middleware to create a logical or virtual data warehouse.
- An operational data store (ODS) is a staging area that provides query facilities.
- Data lakes are large data repository that store raw data and can be set up without having to first define the data structure and schema.
- Business intelligence (BI) is the set of activities, techniques, and tools aimed at understanding patterns in past data to predict the future.

8.6 Data Management for Shallow and Deep Learning Applications

- Data integration aims to provide a unified view and/or unified access over heterogeneous, and possibly distributed, data sources.
- Big data encompasses both structured and highly unstructured forms of data.
- The scope of big data has five Vs: Volume, Velocity, Variety, Veracity, and Value.
- Data virtualization is a technique that hides the physical location of the data and uses data integration patterns to produce a unified data view.
- Data quality involves various criteria to assess the quality of a dataset.
- The aim of data governance is to set up a company-wide controlled and supported approach toward data quality that is accompanied by data quality management processes.
- An analytics process model includes prepressing, analytics, and postprocessing.
- Big data analytics creates a model from a set of data; to learn from the data we need machine learning algorithms.
- Streaming analytics performs analytic processes on streaming data.
- Cognitive computing is a technology tries to simulate human's way in solving problems
- Artificial intelligence is a system that creates intelligent ways to solve problems that previously required human interaction

8.7 Informatics and Data Management

- Informatics describes the study, design, and development of information technology for the good of people, organizations, and society.
- Information systems clearly support informatics and provide an organizational context for using database systems to help collect, organize, store, analyze, preserve, retrieve, and govern data and records relevant to an organization.
- Information systems provide resources involved in collection, management, use, and dissemination of information resources of organizations.
- An information system life cycle includes feasibility analysis, requirements collection and analysis, design, implementation, and validation and acceptance testing.

Review Questions

1. What statement best describes the differences among data, information, and knowledge?
   a. Data are collected from random sources. Information is gathered from specific sources. Knowledge is the process of reading and understanding data and information.
   b. Data are information that was targeted to be collected and stored by a system or application. Information is the bulk data that have not been analyzed to determine to keep or delete. Knowledge is the underlying decision on what information to turn into data.
   c. Knowledge is the process of reading and understating information and data. Information is data that have been collected from targeted research of online and academic sources. Data are information that

was collected from research and development experiments.

d. Data are the raw data without meaning. Information is processing and formatting the data. Knowledge is the process of using and analyzing the information.

2. What does a data steward do?

a. They ensure that the enterprise's actual business data and the metadata are accurate, accessible, secure, and safe.

b. They have the authority to ultimately decide on the access to, and usage of, the data.

c. They are responsible for creating, implementing, and maintaining the database management system.

d. They design conceptual data model (blueprints) to bridge the gap between the business processes and the IT environment.

3. What are three examples of data quality parameters?

a. size, accuracy, and variety of sources

b. accuracy, age of data, and size

c. accuracy, security, and variety of sources

d. accessibility, accuracy, and security

4. Define data governance.

5. What is metadata and what is involved in cataloging them?

6. Data science is a combination of what domains?

a. computer science, and mathematics and statistics

b. computer science and business area applications

c. computer science, mathematics and statistics, and business area applications

d. mathematics and statistics, and business area applications

7. What type of data model focuses on scalability and easily copes with irregular or highly volatile data structures?

a. relational DBMS

b. XML DBMS

c. hierarchical DBMS

d. NoSQL DBMS

8. A federated DBMS architecture can be described as

a. a uniform interface to multiple underlying data sources, which hide the underlying storage details to facilitate data access.

b. where the data are maintained on a centralized server at a single location.

c. where the DBMS and database are hosted by a third-party cloud provider.

d. an architecture that stores all data in internal memory instead of slower external storage.

9. What are the three layers into which an n-tier DBMS architecture divides an application?

a. client side, server side, virtual

b. physical, logic, virtual

c. presentation, logic, data

d. front-end, back-end, network

10. What is physical data independence?

a. the use of multiple physical storage devices such as local servers and external hard drives where data on one device is not dependent on data on other physical storage devices

b. a separation of the conceptual level from the physical level

c. where the data stored on local devices is not dependent on data stored in the cloud

d. a separation of logical layers and presentation layers of data

11. Give a few examples of data models and explain how they differ from one another.

12. Give examples of DBMS users.

13. What type of systems is described where the presentation layer is only on the client device but the application and data layers are on the server devices?
- **a.** thin-client
- **b.** fat-client
- **c.** distributed application
- **d.** parallel processing

14. What is the primary key?
- **a.** the special string of characters to unlock a table for editing
- **b.** a special unique identifier for each table record
- **c.** the first object in the table that locks in the format for that table
- **d.** the first field of an object

15. What are some of the relational algebra operations?

16. Explain how QBE works.

17. What is a key constraint in the relational model?

18. What is an NFNF DBMS?
- **a.** a database data model that does meet any of the conditions of database normalization defined by the relational model
- **b.** a database data model that does not meet any of the conditions of database normalization defined by the relational model
- **c.** a database data model that does meet all of the conditions of database normalization defined by the relational model
- **d.** a database data model that takes a nonrelational data model and formats the tables to conform to a standard format

19. Which term is described as a simple database that uses an associative array that stores only key-value pairs, provides basic functionality for retrieving the value associated with a known key, and works best with a simple database schema?
- **a.** tuple and document stores
- **b.** graph-based databases
- **c.** key-value stores
- **d.** column-oriented databases

20. What is a data lake?
- **a.** a scaled-down version of a data warehouse aimed at meeting the information needs of a homogeneous small group of end users
- **b.** data stored without organization to make retrieval easy
- **c.** a large data repository that stores raw data and can be set up without having to first define the data structure and schema
- **d.** centralizes an enterprise's data from its databases, supporting the flow of data from operational systems to analytics/decision systems by creating a single repository of data from various sources both internal and external

21. How many schemas can a data warehouse consist of?
- **a.** Many; there is no defined number of schemas.
- **b.** One; all tables must conform to one schema.
- **c.** None; schemas are undefined in data warehouses.

d. There is one for each table in the data warehouse.

22. Which is one of the five Vs of big data?
a. verified data
b. validated data
c. volatile data
d. volume of data

23. What is the definition of data governance?
a. It is a technique that hides the physical location of data and uses data integration patterns to produce a unified data view.
b. It uses enterprise application integration (EAI) corresponding to the synchronous or asynchronous propagation of updates in a source system to a target system.
c. It aims to set up a company-wide controlled and supported approach toward data quality that is accompanied by data quality management processes.
d. It uses enterprise information integration (EII) to provide a unified view over data sources.

24. Is MapReduce part of Hadoop?

25. What is informatics and how does it relate to information systems?

Conceptual Questions

1. How can you measure the effectiveness of data governance in an organization, and can you quantify it?

2. What is the relationship between data governance and data quality?

3. Should the age of data be a factor in the quality of data?

4. We typically use the "pyramid of knowledge" to relate to the various level of abstractions encountered when managing data. These levels are known (going from top to bottom) as data, information, knowledge, and wisdom. Can you explain why we use a pyramid to represent the levels of abstraction? Give a specific example of a pyramid starting with actual data. Can you explain the thinking process that makes it possible to go from one level to another?

5. Why would it be easier to scale unstructured data than structured data?

6. Use a free trial version of ErWin to put together a universe of discourse (UoD) for a conceptual relational model involving the following entities: EMPLOYEE, DEPARTMENT, PROJECT, WORKS_ON.

7. Both NFNFs and ODBs allow nested queries. Can you explain why ODBs are better at supporting deeply nested queries? (*Hint*: Think about object identity.)

8. Draw a simple star schema for a problem involving product sales according to time, region, and price.

9. Explain the use of MapReduce on the simple example that consists of counting words in a very large document.

10. How can informatics be used to solve problems?

Practice Exercises

1. Draw a flow diagram that explains how information architects, database designers, data owners, data stewards, database administrators, and data scientists coordinate their activities.

2. Research how bad data can result in bad information and why sources impact the quality of the data.

3. Draw a diagram of a DBMS architecture that could be used to run a medium-scale website.

4. Research and provide a few examples of pros and cons of SQL databases compared to NoSQL databases.

5. Which statement is correct?
   - The Boyce-Codd normal form is more strict than the fourth normal form.
   - The Boyce-Codd normal form is more strict than the third normal form.
   - The second normal form is more strict than the Boyce-Codd normal form.
   - The first normal form is more strict than the Boyce-Codd normal form.

6. Give practical examples of applications where it would be best to use the following: key-value store, document store, column-oriented database, graph-based database

7. Use a simple example to explain the difference between hindsight, foresight, and insight. What type of technology can be used to derive each one of these?

8. Which Amazon tool would you use to turn text into speech? What are the other tools listed used for?
   - Amazon Lex
   - Amazon Echo
   - Amazon Polly
   - Amazon Rekognition

9. State a major constituent of informatics.

Problem Set A

1. You are working for a large retailer, and you are given access to radio frequency identification (RFID) devices, which give you the ability to tag every item the company has in stock. Describe the team you would recommend putting together to derive some real benefits about the tagging, and recommend an approach to drive improvements in sales.

2. Provide an example of data governance relating accuracy, accessibility, and security.

3. Give an example of data that could benefit from being represented using the hierarchical model.

4. Write a use case that would suggest that SQL is a better choice than NoSQL, and explain why it's a better choice.

5. Write a use case that would suggest NoSQL is a better choice than SQL would be, and explain why it's a better choice.

6. Develop a conceptual model using ErWin and map the conceptual schema to a relational database schema for a database target of your choice. Review the generated SQL statements and comment on them. Then populate the resulting database with sample data of your choice and create an SQL query that will let you retrieve all the employees that work on a given project.

7. Research some popular SQL and NoSQL tools. Create a list of tools that are available for each database type. Download a version of an SQL and NoSQL tool of your choice and create a table in each.

8. Contrast operational, tactical, and strategic decision-making. Illustrate with an example in
   - an online retail setting (e.g., Amazon, Netflix, eBay),
   - a bank setting, or
   - a university setting.

9. Ideally, data integration should include
   - only data.
   - only processes.
   - both processes and data.

10. What components does the base Hadoop stack include?

11. Explain how informatics relates to sports.
12. Explain how informatics relates to urban life.

Problem Set B

1. During the COVID-19 pandemic, testing for active COVID infection using nasal swabs increased significantly. Examples of the collected features are name, age, sex, and symptoms (cough, sore throat, loss of smell and taste, fever >100.2°F, shortness of breath, rash, headache, and congestion). Create a data dictionary table with the following attributes:
   - Name: attribute name
   - Definition: describe the attribute
   - Data type: attribute data type
   - Possible values: the possible values for the attributes
2. Explore the concept of the age of data by researching an example of a piece of data that changes rapidly and a piece of data that does not change over the course of time. Provide a suggestion on how to use the data and determine if the data are not relevant and no longer accurate.
3. Explain the difference between OLTP and OLAP using a practical example.
4. Suggest criteria for selecting an open-source versus a commercial database system.
5. A food delivery service wants to gather the personal information of new clients and uses an online form in which all fields are required to be filled. Which record organization technique would be preferred for this purpose? Why would it be the best choice?
6. Select a key-value store of your choice and implement a sample application using it.
7. Discuss four approaches to deal with slowly changing dimensions in a data warehouse. Can any of these approaches be used to deal with rapidly changing dimensions?
8. Discuss some application areas where the usage of streaming analytics (such as provided by Spark Streaming) might be valuable. Consider X (formerly Twitter), but also other contexts.
9. If Spark's GraphX library provides a number of interesting algorithms for graph-based analytics, do you think that graph-based NoSQL databases are still necessary? Explain why or why not. Search the Web on how to run Neo4j together with Spark and explain which roles they both serve in such an environment.
10. Design a sample end-to-end informatics solution for an area of your choice. If you have time, implement a simple prototype of your solution as a web-based application using a data management infrastructure of your choice.

Thought Provokers

1. Consider our start-up company that is 100% committed to leveraging innovative technologies as a business growth facilitator. Describe how it can best use data management knowledge to create products or services that can generate business. Give precise examples and explain how the start-up would be able to scale the resulting business (i.e., keep sustaining the cost of doing business while increasing its number of customers).
2. Consider our start-up company that is 100% committed to leveraging innovative technologies as a business growth facilitator. Describe how it could leverage a database system to support a new social media application that makes it possible to gather various types of data from various data sources including sensors located at the edge of the network. Give some precise examples and explain how the start-up would be able to scale this approach.

3. Consider our start-up company that is 100% committed to leveraging innovative technologies as a business growth facilitator. Describe how it could use a "universal" approach to apply a RDBMS to all possible data management situations. In particular, can you tell if such a solution already exists and if so, give precise examples. What are or would be the limitations of such a solution and is it even possible?
4. Consider our start-up company that is 100% committed to leveraging innovative technologies as a business growth facilitator. Describe how it could use a Hibernate to access many different types of database management systems. Are there specific DBMSs that would not be covered by this solution? Would it be a good idea to extend Hibernate to do so, if there is such a solution already in place? If not, what are or would be the limitations of doing so and is it even possible?
5. Consider our start-up company that is 100% committed to leveraging innovative technologies as a business growth facilitator. Describe how it can use data lakes as an umbrella for many existing data warehouses and add to them. Is there such a solution already in place? Elaborate on the benefits and drawbacks of this approach and explain how the start-up would be able to scale this approach.
6. Consider our start-up company that is 100% committed to leveraging innovative technologies as a business growth facilitator. Describe how it can use machine learning to create software solutions that can operate autonomously. Are there any such solutions available today? Give some precise examples and explain how the start-up would be able to scale this approach.
7. Consider our start-up company that is 100% committed to leveraging innovative technologies as a business growth facilitator. Describe how it could put together a generic framework to support informatics applications in any domain. Does such a solution exist already? Either way, how would the start-up be able to scale this approach?

Labs

1. Search online to learn what a virtual machine is. You are setting up a virtual machine (VM) on Microsoft Azure and would like to perform data science experiments. Research the best way to gain access to all the tooling you need without having to research and install the individual tools on your own.
2. Select three examples of commercial or open-source DBMSs that use different data models. Install the trial versions of each one of these DBMSs and illustrate their use via a simple tutorial example. Document your work and evaluate the benefits and drawbacks of each system based on your experience.
3. Explore MySQL and experiment with MySQL Workbench to build a simple website using Django. Refer to the instructions and tutorial (https://openstax.org/r/76DjangoProject) for more information.
4. Build a simple Django application that implements a social media website and uses a cloud-based data management service for data management. (*Hint*: You can use this article from Medium (https://openstax.org/r/76BuildDjango) that contains some guidance.)
5. Explore how to use AWS service areas when solutioning use cases for a data lake. Data are stored in a raw state initially, and some use cases will use raw data as is. More often, solutions require varying degrees of data preparedness based on a collection of query usage profiles that correlate to actual use cases. Based on the solution, data may be refined and staged with the intent to promote modularity and reuse. The goal is to not overprocess the dataset because it is intended for multiple purposes downstream, such as AWS RedShift for relational analytics, AWS Elasticsearch for text search, or an optimized distributed file system for low-cost active archive storage, which can be queried with an MPP SQL engine.
6. Investigate how to put together an end-to-end data management infrastructure for a recommender application being built by a start-up. The application is expected to collect hundreds of gigabytes of both structured (customer profiles, temperatures, prices, and transaction records) and unstructured (customers' posts/comments and image files) data from users daily. Predictive models will need to be

retrained with new data weekly and make recommendations instantaneously on demand. Data collection, storage, and analytics capacity would have to be extremely scalable. The questions at hand are: How can you design a scalable data science process and productionize the models? What are the tools needed to get the job done? You will need to explain how to set up a data pipeline,

7. Leverage the types of choices suggested in the associated diagram, decide between on-premises and cloud services, choose a cloud service provider if applicable (in particular, investigate the cloud service provider's ML/DL capabilities and build your solution to avoid cloud vendor lock-in), and develop robust cloud management practices.
8. Search the Internet for available informatics platforms and experiment with any of the ones you find.



9

Software Engineering

Figure 9.1 Software engineering is used to develop solutions that are used in everyday life, such as controlling digital instruments on automobiles' dashboards to optimize the driving experience. (credit left: modification of "Flickr - Nicholas T – Enduring" by Nicholas A. Tonelli/Wikimedia Commons, CC BY 2.0; credit middle: modification of "Lexus LF-A" by Nan Palmero/Flickr, CC BY 2.0; credit right: modification of "M25, Orbital, Approaching Bright - Heads up Display" by Jay Galvin/Flickr, CC BY 2.0)

Chapter Outline

9.1 Software Engineering Fundamentals
9.2 Software Engineering Process
9.3 Special Topics

Introduction

What is software engineering? Why is software engineering important? How is software engineering done? Well, software engineering impacts every industry and all aspects of life. One sector where software engineering has been revolutionary is the automobile industry. Software that has been written to automate the dashboard is being reinvented again and again to augment the driving experience with software solutions such as universal head-up displays for cars. These displays provide speed and navigation information at eye level and thus keep drivers from having to glance down at their dashboards while driving.

Software is ubiquitous nowadays, and its presence is changing traditional industries. For example, software is more important than ever before in automobiles, and automobile manufacturers are responding by investing more and more on its development. The development of digital health-care devices has similarly caused the health-care industry to invest in software development. Even customer support operations are changing by utilizing Generative AI (GenAI) software such as ChatGPT to answer questions about diseases and how they spread. As spending on software development increases, traditional companies become technology companies in which software and its development play important and most often crucial roles. Advances in embedded systems, communication technologies, machine learning, and intelligent autonomous solutions, such as autonomous cars, large language models (e.g., ChatGPT), and insulin pumps and other wearable health technologies, will further boost this trend across most industries.

Software engineering focuses on the design and development of software. As companies undergo digital transformations, software engineers must know and apply sound principles to the solutions they help create.

They must collaborate with enterprise and solution architects and be familiar with software engineering life cycle processes to address the needs of the companies they work for and to use approaches, techniques, and tools to design and develop meaningful software.

As businesses evolve and technology changes, so do the expectations placed on a software engineer. For example, consider how an average automobile from forty years ago had little to no software engineering in its operation. In today's automobile industry, software engineers are creating a variety of solutions that are incorporated throughout a vehicle. This ranges from the software running on interactive screens on dashboards that feature music systems and navigational aids to the software that allows for control of hands-free phone calling. As software engineering solutions in automobiles continue to evolve and be applied to all aspects of driving, they are paving the way toward self-driving, autonomous vehicles. Software engineers face many challenges because software engineering processes always need to be tailored to each project, and software engineers must be able to help understand, design, construct, and deploy software solutions that move that project forward. Thus, the process of developing software for an automobile that supports embedded features such as heads-up displays, lane departure systems, and even embedded solutions that alert the driver about mechanical issues such as low air pressure in tires would be quite different from that for a smartwatch that features applications that help users track their exercise and monitor their sleep patterns. While software engineers follow a process to derive the solutions they build, they must constantly adapt to handle issues and changes associated to the business environment they work within as well as refine the technology and approaches they use.

Software engineers also need to understand business requirements to ensure the software solutions they create are successful. This is made more difficult as requirements constantly change, or as people involved in a project have conflicting expectations. As anyone who has sent a text to a friend about dinner plans only to discover that it has been changed to be a discussion of "ducks" and other ornithological matters, even a seemingly straightforward requirement like "automatically correct misspelled words" has multiple interpretations. Furthermore, assessing whether the misbehavior of solutions results from broken code or a misunderstanding of requirements often leads to confusion and disagreements between the software engineers and their customers. To minimize such issues, it is important for software engineers to understand their role and to familiarize themselves with software engineering processes, the types of solutions and technologies that are available, and their applicability to a given project at a given time.

Consider a scenario in which a software engineer is working at a company called TechWorks. The engineering team they are part of is tasked with a project to add new software features to the dashboard of an upcoming electric automobile. Because the process of software engineering cannot (in general) be fully automated, software engineers face the challenge of having to determine the best software process, approaches, and tools to be used for the project. They must do so while collaborating with a number of people and facing changes in requirements, and they still have to deliver solutions on time and within budget. In the example of the automobile dashboard, the design and development of a new digital speedometer requires coordinated efforts from all the team members and their alignment with the same goals. Fortunately, as you will learn in this chapter, there are a variety of software frameworks and tools that the software engineering team can apply to a project to increase its chance of success.

9.1 Software Engineering Fundamentals

Learning Objectives

By the end of this section, you will be able to:

- Describe the intent of software engineering and how it relates to computer science
- Recognize various categories of software
- Identify the skills that are required for a software engineer

This section introduces software engineering along with the challenges associated with the creation and

evolution of software. The various fundamental facets of software engineering are presented including the skills required to properly engineer software and the role that a software engineer is expected to play within a software team.

The Intent of Software Engineering

The IEEE[1] defines software engineering as the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. It is important to understand how software engineering fits into the larger context of computer science. It is also important to understand the role of a software engineer and how it fits within an organization.

The reference to "a systematic, disciplined, quantifiable approach" does not mean a bureaucratic, document-laden approach. Software engineering relies on various methodologies/processes, methods and tools that may be used to professionally develop and maintain quality software while optimizing the use of resources such as time and cost. Selecting the right processes and methods for a project and using tools to accelerate the ongoing development of a solution is challenging especially when requirements are changing and many software developers are working together to tackle a large project. Part of the issue is that software engineering as a process cannot (in general) be fully automated and therefore it is difficult to keep all the activities aligned with the end goal of delivering a solution on time and within budget.

Selecting and tailoring software engineering processes and methods as well as using the right tools in the right fashion to meet this end goal, which in essence defines a majority of what a software engineer does, is covered in detail as part of this chapter.

CONCEPTS IN PRACTICE

Software and Our Daily Lives

Software is everywhere, and most of us interact with a software system on a daily basis. Yet the importance of software is expected to grow even greater in the coming decades. Although most software revenue comes from a generic software, software systems tailored to specific needs can be found everywhere, from our cars to our smartwatches. Thus, quite a few of us can expect that we will be involved as customers in some software engineering process during our professional lives. People who are familiar with concepts of software engineering can better oversee the projects they are working on by interacting with customers so their needs are clearly specified and by working with developers to smooth out the development process and ensure the final product meets customers' expectations.

Software Engineering Teams

In most organizations, software is developed by a team of software engineers, as well as other people with different backgrounds and skills. To be effective, the project team must be organized in a way that maximizes the use of each person's skills and abilities and emphasizes the importance of teamwork to address all the aspects of the projects. This is especially true for complex projects. Effective project managers focus on problem-solving and end-product quality, and thus they may organize software teams in a variety of ways. The key factors considered when selecting a team organizational model are as follows:

- difficulty of the problem to be solved
- degree to which the problem can be modularized

1 While IEEE is an acronym for the Institute of Electrical and Electronics Engineers, the organization has expanded to include all technology professionals. This worldwide organization is "dedicated to advancing technology for the benefit of humanity." Examples of the work it does is publishing journals to disseminate the latest research, creating model curricula that help universities provide up-to-date degree programs, and defining standards that ensure electronic devices are interoperable. You can find more about the IEEE at www.ieee.org.

- required quality, reliability, and other nonfunctional requirements of the system to be built
- delivery date
- budget limitations
- team lifetime (i.e., overall period of time during which the team members will work together)

During the creation of a software solution, software engineer team members may take on various roles, such as:

- project manager, who is responsible for making sure that the project gets completed on time and on budget
- enterprise or solution portfolio architect, who makes high-level design decisions on high-level modeling or solution design building blocks and their interconnections to best address functional requirements as well an nonfunctional ones such as performance and usability
- user experience (UX) designer, who plans how users will interact with the software system; this involves tasks like designing the user interface (UI), making wireframes and prototypes, and conducting user testing to ensure the software is user-friendly and meets user needs
- software developer, who analyzes requirements, refines the design, writes and tests code, and collaborates with the operations team to deploy the software into testing and production environments
- quality assurance tester, who checks for bugs and validates that the system meets the requirements

It is common for one team member to act in several roles (for example, a solution architect might also be a software developer), and some roles may be filled by multiple team members (for example, there are typically multiple software developers on a project). Additionally, a software engineer may work directly with other people, including:

- product owners, who are generally project sponsors and provide the requirements and overall guidance on what is expected from the solution
- subject matter experts (SMEs), who are specialists in areas such as business, technology, products, or other topics as needed
- database administrators, who focus on areas such as data storage, organization, and access associated with a software solution
- operations specialists, who are IT and/or DevOps professionals with a focus on computer hardware and software operations

Because projects can involve a large number of people working in many different roles, communication and interaction among team members is critical for a project to succeed. To keep the team running smoothly, projects are ideally structured so they include the following:

- deadlines that provide sufficient time for team members to fully complete their tasks and ensure results that match the requirements
- standards for how business and technology decisions are made to avoid business or technology-related disagreements
- processes and procedures that make it easy for team members to coordinate their activities
- clear definitions of each team members' role and authority so that everyone knows who is accountable for each task and where questions should be directed

Cost Management

An important part of the role of a software engineer is to handle the project management triangle of timeliness, quality, and cost. The goal is to deliver projects as quickly as possible with the highest quality and the lowest cost. Doing this requires good communication, planning, and focus as well as the setting of realistic expectations. Furthermore, because the cost of maintaining a software solution over time can be significant, various cost-management strategies must be used to manage costs effectively. These strategies include the use of estimation techniques, resource allocation, and cost tracking tools. Estimation techniques help create

accurate estimates of project scope and effort, and cost, which helps with budgeting and resource allocation. Resource allocation involves assigning the right people (i.e., those with the necessary skills) to complete each task, and this helps optimize efficiency and cost. Cost tracking tools help track project costs and identify areas for potential savings.

The ongoing evolution of computing technology, such as web, mobile, and cloud computing, has had a significant impact on the way modern software engineering is done. Nevertheless, it is still common to get into trouble when designing and developing high-quality software on time and within budget. The major reasons for that are as follows:

- Software systems are the most complex types of systems humans create.
- The pace of change in computer and software technology is high. The programming techniques and related processes that work well when used individually or in a small team fail when applied in a large team developing a large complex system.
- There are not enough qualified software engineers, and thus software systems are designed and developed by people who have insufficient backgrounds or experience.

Software Engineering and Computer Science

Where does software engineering fit within computer science? Both software engineering and computer science deal with the creation of software, and while there appears to be an overlap between these two disciplines, software engineering tends to focus on software applications, more specifically, on the design and development of software solutions. This makes software engineering a branch or subarea of computer science.

Computer science focuses on the study of algorithms and their realization in terms of computers, computer systems, and related computational processes. It focuses heavily on the theoretical and mathematical principles involved with the creation of algorithms and data structures. While computer science includes software engineering, it also covers a variety of other disciplines. In comparison, software engineering focuses specifically on the application of engineering principles to understand, design, construct, and deploy production software. It starts with eliciting and analyzing requirements and results in providing a practical software solution that meets those requirements. Computer science goes beyond just creating software solutions, and includes software design and development. While software engineers need to have a solid knowledge of fundamental computer science (e.g., the basics of algorithms and data structures), they are not expected to exhibit the same level of theoretical knowledge as that of a computer scientist. For example, while a computer scientist would need to be able to establish the asymptotic time complexity of the algorithms they pick, a software engineer would not.

Software engineering departs from the general focus of computer science and is complementary in the following ways:

- Software engineering tends to focus on applying technology to create solutions, whereas computer science tends to focus much more on understanding the technology itself.
- Software engineering tends to focus on software. Computer science can focus on software, hardware, and the interaction between the two.
- A software engineer tends to focus on developing solutions that meet the requirements of an organization or project. A computer scientist tends to focus on the underlying computational processes required to create solutions in the optimal or most efficient ways.

The essence of the software engineering practice is based on the work of numerous researchers such as the mathematician George Pólya. Pólya's heuristic for generalized problem-solving includes the following steps:

1. Understand the problem.
2. Devise a plan.

3. Carry out the plan.
4. Look back and examine the solution.

The Nature and Impact of Software

Software has become an essential part of our lives. It has revolutionized how we communicate, work, learn, and entertain ourselves. From the smartphones in our pockets to the complex systems that power hospitals and transportation networks, software is everywhere.

Because the focus of software engineering is to build software, it is important to understand what software is. A piece of **software** is made up of instructions that can be executed and generally includes documentation that describes the software operation and use. These instructions tell computers what to do. They function like a recipe that tells a chef how to cook a meal. Software is designed to accomplish a specific objective or purpose. Word processing software has the objective or purpose to help people create documents. A game has the objective or purpose of providing entertainment. The software built into an automobile has many purposes, including helping a driver drive the car, helping the owner maintain the car, and helping a driver and/or passengers avoid getting lost.

Software is generally created based on a set of requirements, and although software does not wear out, these requirements often change over time. Software must, therefore, be updated often to respond to these changes. In addition, software can contain defects (i.e., bugs), which are expected to be fixed in a reasonable time after they are reported.

Over the years, software engineering researchers and practitioners have converged on defining a generic software engineering process model, or process framework, that can be used as a template to characterize the generic activities performed by all the software engineering process models that are used to support the **software development life cycle (SDLC)**. The SDLC is a structured set of the framework activities required to develop a software solution based on a set of requirements. A process framework generally encompasses four framework (or generic) activities: inception, elaboration, construction, and deployment. These activities are also known as phases.

You'll learn more about each of these phases later in the chapter. In the meantime, it should be noted that the deployment phase covers integration and user acceptance testing, installation, and maintenance/support activities. This means that maintenance is an integral part of the software engineering process. In fact, the cost of software maintenance/continuous deployment today often exceeds the cost of software construction, especially if software remains in use for a long time.

Categories of Software

There are three main categories of software. Software that enables you to control hardware and provides an environment in which other software can run is called **system software**. An example of this is an **operating system**, such as Microsoft Windows, Android, and MacOS, which runs on a laptop or a mobile phone and typically controls and provides access to the computer's basic functionality. This system software also includes driver software, which is the code used by hardware devices to interact with the operating system.

Software that enables you to fulfill common tasks, such as creating a text document, drawing a picture, or playing music, is called **application software**. Examples include Microsoft Word and Paint, MacOS Pages, and iOS Files. Application software may include specialized categories of software such as scientific (or engineering) software, which is software that aids you in solving mathematical, scientific, and engineering problems, such as finding the roots of an equation. It can also include software services, which are software programs that provide specific responses or actions based on a request, such as returning stock quotes, obtaining weather conditions, or finding what current jobs are available.

Software that is integrated with hardware and can include both application and system software features is called **embedded software**. Examples include software in a smartwatch, an automobile control system, or

microwave. Interaction with embedded software is usually limited, and users are often unaware that they are interacting with software. With the increased popularity of the Internet of Things (IoT), embedded software development is increasing.

Software engineers must often deal with **legacy software** that has been written in the past, relies on obsolete technology, and is still in use today. This software presents special challenges to those who maintain it, especially when they need to adapt it to a new computing environment or technology and when they are to extend its functionality so that it is interoperable with new software systems. The need for software maintenance of legacy software stems from the rapid changes that have taken place in hardware. For example, a common smartphone contains a more powerful processor and more memory than computers that were used just twenty years ago. Changes in hardware are accompanied by changes in software technologies, and these are quickly adopted by users who are then reluctant to use legacy software in its original shape. For example, the graphical user interface of a software system that was developed twenty years ago now looks outdated and leads to an unpleasant user experience.

There have also been dramatic changes in processor technology. Twenty years ago, most processors consisted of a single core, while now it is common for a processor to have multiple cores and coprocessors. This requires changes in how software is programmed because software that can utilize multiple processor cores has a competitive advantage compared with software that uses only a single core. To support these new capabilities, major programming languages have evolved toward supporting multithreading, which makes maintenance of a legacy code base even more challenging.

CONCEPTS IN PRACTICE

The Ever-Changing World of Software Platforms

Imagine the life of a software engineer developing programs in the 1980s, which required typing programs on a terminal and printing punch cards that were fed into a mainframe computer! The world of software development is constantly evolving, and the platforms where software runs change rapidly, too. Software engineers need to stay up-to-date on current platforms and also become aware of what is coming next.

A Trip through Time

Can you imagine using a computer without a mouse or a graphical interface? Let us take a look at the way software platforms have evolved over time:

- Early Days (1980s): Mainframes and minicomputers were giant, even room-sized, and software was often run from the command line.
- The Rise of the Desktop (1990s): Welcome to the age of Windows and Mac! Graphical interfaces made computers more user-friendly, and client-server systems allowed applications to run across multiple machines.
- The Web Takes Over (2000s): The Internet exploded, and Java and C# became popular languages for building web applications.
- Mobile Mania (2010s): Smartphones like the iPhone changed the game. Software development focused on mobile apps that took advantage of constant connectivity.

The Future Is Unknown but Exciting

Augmented and virtual reality (AR/VR), machine learning (ML), artificial intelligence (AI), the Internet of Things (IoT), and blockchain are some emerging technologies that are shaping up the future of Metaverse software platforms. Who knows what exciting new platforms await? Curious and adaptable software engineers will be best positioned to thrive in this ever-changing landscape.

Software Requirements

Software is developed based on requirements, which are categorized as functional and nonfunctional, as shown in Figure 9.2. Functional requirements state what the software needs to do in terms of tangible functionality, such as the login or registration functionality that allows users to access a given piece of software such as their Facebook account. Nonfunctional requirements are measurable qualities of a piece of software, such as usability, performance, and other quality aspects.

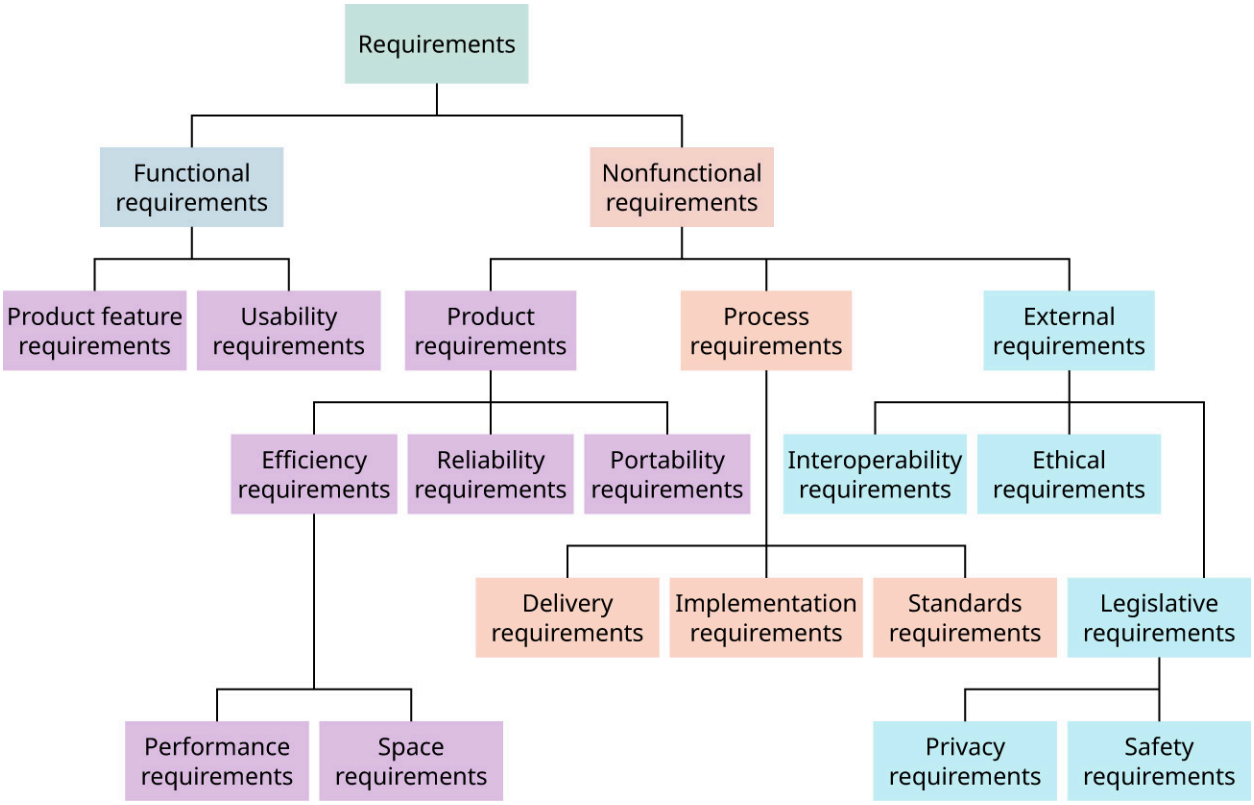


Figure 9.2 The various types of software requirements can be organized in hierarchies as functional and nonfunctional requirements based on aspects that relate to what the software does along with the qualities of the software. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

As noted earlier, a **functional requirement** relates to the services the system provides to the users. It reflects the user's expectation about the inherent characteristics of the software, (i.e., the functions it must perform). Functional requirements can include those phenomena that are generally seen or done when using a software solution, such as the information that is entered or that is generated by the software, the authenticating rules for using the software, or whether there should be an audible alert (such as a beep) when an action is selected. A **nonfunctional requirement** describes a desired quality or feature (i.e., constraint) and covers aspects of the software such as flexibility, maintainability, performance, portability, reliability, scalability, security, and more. For example, a nonfunctional requirement could specify that responses from the software must occur within a specific time frame, or that a software system must be able to handle a certain number of transactions at any given time. Another example of a constraint that a nonfunctional requirement may express is the need to use a specific programming language or follow a particular standard.

Nonfunctional features are sometimes categorized in terms of static or dynamic qualities. A **static quality** is a nonfunctional feature that is unchangeable and thus might be associated, for example, with the source code and related documentation, or with legal or project-environment specific requirements. A **dynamic quality** is a nonfunctional feature that relates to the qualitative behavior of software while it is in use, which means that is also depends on the hardware that the system runs on. Both quality types are measurable features rather than functions that are present or absent. Examples of static qualities include:

- **maintainability**, which is a measure of the amount of work required to fix bugs, improve performance, and keep the system running
- **extensibility**, which is a measure of the amount of work and cost of adding new features to software
- **flexibility**, which is a measure of the difficulty of updating the system to meet changing requirements
- **portability**, which is a measure of the amount of work and cost of migrating software solutions
- **usability**, which is a measure of how intuitive the user interface is

Examples of dynamic qualities are:

- **performance**, which is a measurement of how quickly the system responds to requests it receives
- **throughput**, which is a measure of how much input the system can process within a given time frame
- **availability**, which is a measure of how often the system will be ready and active for users
- **scalability**, which is the measurement of the work and cost to increase a system's throughput
- **security**, which is a measure of confidence that data is protected from unauthorized disclosure and that systems are protected from unauthorized access

The relative importance of the preceding characteristics depends on the solution requirements and the environment in which they are to be used. For example, in an Internet banking application, availability and usability are typically key features advertised to customers. Security would also be important, as the software solution would need to comply with regulatory requirements. In a brokerage system, performance would be an important requirement because the software system will need to display prices and process users' orders in real time.

THINK IT THROUGH

Professional Responsibility in Software Engineering

Given the professional responsibility of software engineers working on various projects across the world, what is right or wrong, in your opinion, about the following matters in a software engineering context: developing systems for the military (e.g., creating an autonomous military drone); adding a backdoor to an existing system (e.g., providing authorities with a way to unlock a locked mobile phone); or modeling patient medication needs (e.g., prescribing pain medication for each hospital patient based on historical data).

In each of these cases, the development of the system could provide real benefits. For example, an autonomous drone allows the military to act without putting its soldiers' lives at risk, and a backdoor ensures law enforcement officials can get access to information they may need to solve a crime. But there are also drawbacks to each of these systems. The backdoor risks everyone's privacy because anyone could use it to access private data on a phone, and relying on past medication data risks institutionalizing existing biases about who needs more (or less) pain medication. It is important for software engineers to consider all of the impacts that a system will have and determine whether these impacts are acceptable or not.

Software Engineer Skills

As mentioned earlier, a software engineer is involved in the creation of software. Software engineers work with others and are a part of a team. They must be able to understand and act on a vast array of requirements. To accomplish what is required, a successful software engineer must have strong analytical and problem-solving skills, must communicate well with their peers, must be a team player, must take responsibility for their commitments, must exhibit attention to detail, and must be pragmatic when adapting software engineering practices based on circumstances. In addition, software engineers must be flexible when it comes to adopting new technologies. Therefore, in terms of technical skills, software engineers must have a solid foundation in the following areas:

- Problem-solving and algorithms: Software engineers are like detectives who solve puzzles. They use algorithms, which are step-by-step instructions, to efficiently solve problems. Understanding different algorithms allows them to choose the best approach for a specific task. There are many algorithms that can be considered essential, and it can be hard to learn all of them. Understanding the importance of algorithms and being willing to learn new things is a required skill for software engineers.
- Data structures: Imagine a well-organized toolbox where each tool has its designated spot. Data structures are like these toolboxes, but for data! Knowing different data structures (such as arrays, lists, trees, stacks, and queues) and being able to assess their benefits and drawbacks helps software engineers organize information efficiently for their programs. The knowledge of fundamental data structures is essential because without efficient data structures, you cannot write efficient code.
- Programming languages: There are many programming languages and each one of them has strengths and weaknesses. Software engineers should be familiar with at least a few core languages, such as Java, Python, or C++. Most important, they should be adaptable and willing to learn new languages as needed.
- Software engineering process: Software is usually developed in a team and follows a process. This process describes how to organize a team of developers effectively as well as how to approach the creation of the software. There are many approaches to team-driven software development and software engineering processes can be either continuous or discrete. Discrete processes, such as those advocated by Agile development[2], encourage flexibility as to what activities are required and to what degree within a given project. Usually, all activities will be conducted when using continuous processes. A software engineer should be familiar with the pros and cons of various software engineering processes to be able to apply the best one to their own software development projects. Later in this chapter, both traditional and Agile software development life cycle (SDLC) processes, and various related models and frameworks, are covered in more detail.
- Testing: The purpose of testing is to ensure that a software system is free of defects and meets the functional and nonfunctional requirements. It is important to realize that testing is an ongoing activity that occurs throughout the software engineering process. For example, software engineers may conduct unit testing of a website's checkout functionality as part of their software construction activities to make sure that the software calculates the total of a customer's order correctly. They may also be involved in integration testing as components developed by other team members are being integrated.
- Tools: There are many tools that can be used in software development, and these tools go beyond just being used to create software code. They can, for example, facilitate communication within the team, help support engineering tasks, and maintain a historical record of source code versions. Each software engineer should be familiar with major tools and their underlying principles so that they can use them effectively.
- Computer technology: Software engineers need at least a high-level understanding of the computer technology that might be applicable to the software solutions they may be required to create or work with. This includes areas such as operating systems, cloud services, security, networking, communications, database technology, and UI/UX. Software engineers might also be called upon to work with computer technologies such as machine learning (ML) and its applications to artificial intelligence (AI), business intelligence (BI), and the multitude of ever-evolving applications of computer technologies that are available today. A successful software engineer is aware of current computer technology and understands how to leverage it.
- Soft skills: A software engineer should have the soft skills required to assume their role such as problem solving, communication, and time management, as shown in Figure 9.3. Soft skills tend to be more innate, but are often just as critical as the previously listed skills. For example, a software engineer could be tasked with developing a software solution within twelve months that tracks and displays relevant driving information onto a heads-up display presented on the windshield of a new model of an automobile. To deliver this solution on time, the software engineer would need to manage a schedule. As part of a team

2 The original goals of Agile development were laid out in the *Agile Manifesto*. This is at https://AgileManifesto.org.

project, they would also need to understand how to effectively communicate and interact with other project team members. Additionally, they would need to problem-solve the requirements to derive a solution using appropriate algorithms and patterns. Problem-solving includes understanding how and when to bring subject matter experts or non–software engineer resources onto a project. Being able to apply these softer skills can be just as critical to a software engineer's success as applying the right processes, methods, tools, and data structures/algorithms.

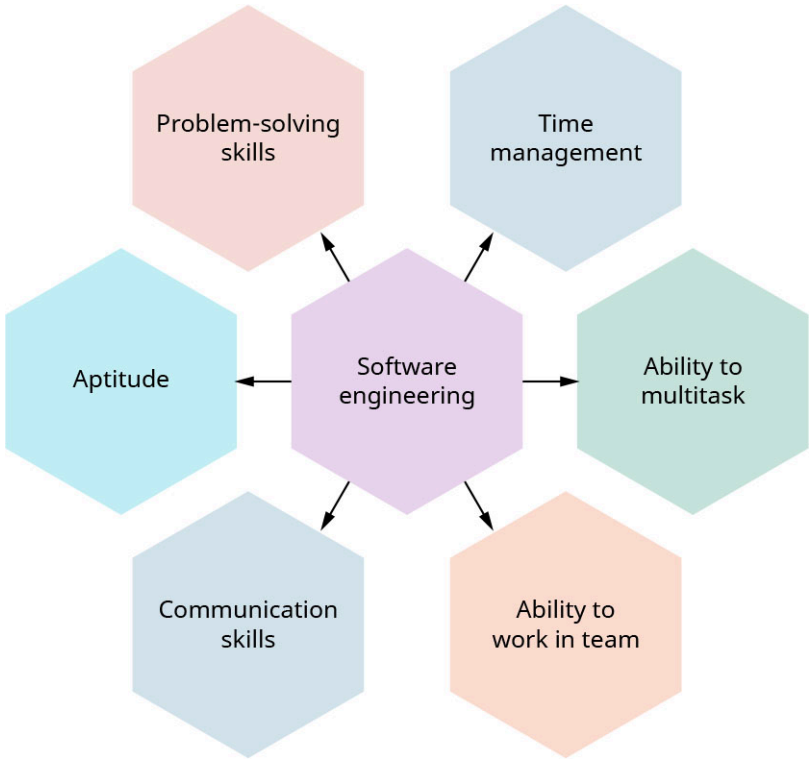


Figure 9.3 These are some of the soft skills required by software engineers. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

TECHNOLOGY IN EVERYDAY LIFE

Programming Languages: A Software Engineer's Toolkit

Imagine a toolbox with specialized tools—a hammer for carpentry, or a paintbrush for art. Software engineers also use various tools, and some of their tools are programming languages! Just as the right tool makes a job more accessible, the correct programming language helps engineers build different software applications.

Can You Guess the Language?

Identify the programming languages commonly used to build the following types of applications:

- Mobile app (like Instagram)
- Website (like Wikipedia)
- Video game

Possible answers include JavaScript, Python, C++, Java.

Why So Many Languages?

Each programming language has its strengths. Some languages are great for building an application's core

functionality (like C++ or Java), while others are better suited for adding interactivity (like JavaScript). Understanding different languages allows software engineers to choose the right tool for the job.

How Many Languages Should a Software Engineer Know?

There is no right or wrong answer to this question. It is possible, though not common, to be employed as a software engineer knowing only a single programming language. Typically, however, successful software engineers understand the different categories of programming languages and the common structure and design features of each.

It is often recommended that a software engineer learn a primary programming language that can be used for system-level programming such as C, C#, or C++. Additionally, in today's world, a software engineer should understand object-oriented programming, which is done in languages like Python, C++, and Java. Because web development often is a part of solutions, knowing a scripting language, such as JavaScript, Python, PHP, Perl, or Ruby, is also beneficial. Finally, understanding markup languages such as HTML and XML is also important.

Of course, the best languages to know are dependent on the projects that are worked on. If a software engineer is creating embedded solutions, then the programming language is likely going to be much different than if they are doing high-end gaming solutions or web-based solutions. Chapter 7 High-Level Programming Languages delves deeper into programming languages.

9.2 Software Engineering Process

Learning Objectives

By the end of this section, you will be able to:

- Describe the phases of a software development process and their purposes
- Study the popular traditional prescriptive and Agile software process models
- Suggest an effective software process

Imagine a recipe for building software. There are different ways to cook the same dish, but most recipes follow a basic structure with steps like gathering ingredients, preparing them, cooking, and serving. Software engineering processes are similar. They provide a structured approach to creating software applications. Various software engineering process models are typically used to support the software development life cycle (SDLC). After years of research and refinements, software engineering researchers and practitioners have converged on defining a generic software engineering process model, or process framework, that can be used as a template. That process framework includes a set of process elements (e.g., framework activities, software engineering actions, task sets, work products, quality assurance, and change control mechanisms) that may differ for each process model and for each project.

Traditional Process Models

One common category of process models is known as the **traditional process model**. This process framework, as you learned earlier in the chapter, encompasses four framework (i.e., generic) activities that are also known as phases: inception, elaboration, construction, and deployment:

- Inception covers planning activities where you define the project goals and identify the overall scope.
- Elaboration involves analyzing requirements and designing a detailed architecture model for the software.
- Construction is where the coding happens! The software is built based on the design created earlier.
- Deployment is the activity that focuses on releasing the software in a usable form and making it accessible to end users.

These (generic) framework activities or phases are applicable regardless of the specific software engineering

process model chosen for a project and may be elaborated differently depending on the organization and the problem area and project being developed. There are also umbrella activities that are important but tangential to framework activities. Returning to the recipe analogy, if generic framework activities represent cooking, then you can think of these activities as the things you would do alongside your cooking, like making sure you have the right pots and pans or keeping your kitchen clean. As you'll learn later in the chapter, in software development, such activities are known as umbrella activities, and they include:

- training and communication (e.g., work product preparation and production)
- risk management and planning (e.g., software project tracking and control)
- configuration management
- quality management (e.g., technical reviews, estimations, metrics/measurements, testing)
- architecture management (e.g., reusability management)
- security management

Various software engineering actions are typically performed as part of the generic framework and umbrella activities. For example, the inception phase may call for requirements engineering actions such as requirements definition and requirements management; the elaboration phase may involve high-level and detail design actions. Each type of software engineering action corresponds to a process that may be represented as a workflow or a task set, and each task results in work products that are subject to specific quality assurance and change control mechanisms. Basically, a task set (or workflow) encompasses all the tasks that are required to accomplish a specific software engineering action within a framework activity. Task sets vary depending on the characteristics of a project, and activities within a given process model usually overlap instead of being performed independently.

Software process models that adhere to the generic framework mentioned precedingly are sometimes referred to as SDLC methodologies. In general, software engineering process models are structured in this fashion to facilitate efficient development of quality software, reduce risk of failure, increase predictability, and capture best practices in software development. The software framework provides a template that allows software engineers to tailor their process model based on the specific project(s) on which they are working. The (generic) framework activities mentioned precedingly are applicable to all projects and all application domains, and they are a template for every process model. Actual process models action/methods may, however, use various approaches. Furthermore, software engineering tools may be used to (semi-)automate the various methods that perform activities.

The activities involved in developing software might vary depending on the organization and the type of software being developed. There is no single right way to create a software solution, but experience typically tells us what works well and what works poorly in a given context. Therefore, process frameworks are elaborated differently depending on the four Ps—problem, project, people, and product—they tackle.

In the past, this led to the use of various traditional prescriptive process models such as the waterfall, prototyping, spiral, and rational unified processes. A **prescriptive process model** advocates an orderly approach to software engineering that involves following a prescribed set of activities in a continuous manner. These traditional process models provide a structured approach to software development and may help with the following objectives:

- Improve efficiency: By following a clear plan, teams can work more efficiently and avoid rework.
- Reduce risk: Identifying and addressing potential issues early on can help to prevent project failures.
- Increase predictability: A structured process can help to estimate timelines and costs more accurately.
- Capture best practices: Traditional models often incorporate tried-and-true methods for software development.

These days, however, traditional prescriptive process models are perceived by some as "old-school" (i.e., ponderous, bureaucratic document-producing machines). Note that prescriptive simply means that the

process model identifies a set of process elements (e.g., framework activities, software engineering actions, tasks, work products, quality assurance, and change control mechanisms) for each project. Traditional models are generally criticized for being too rigid and inflexible. They may not be suitable for all projects, especially those with rapidly changing requirements.

In general, the various process models may have features in common with each other, and there may be some overlap among the activities conducted within each given process. In the next section, we'll explore an alternative approach called Agile software development.

Agile Process Models

The *Agile Manifesto* sets forth the Agile philosophy and emphasizes the fact that software engineering processes should not be constrained to be continuous. It advocates that it is fine to skip and accelerate framework activities to deliver a project solution faster and, therefore, it is fine for the software process to be viewed as a discrete set of meaningful activities to reduce the cost of change. In this context, agility refers to the ability to create and respond to change in order to profit in a turbulent business environment. Proponents of Agile process models question whether prescriptive process models that strive for a structured and ordered approach to software engineering are appropriate for a world that thrives on change. In general, Agile processes have very short product cycles and constantly solicit customer feedback to focus development on customers' current needs. Agile software processes promise strong productivity improvements, increased software quality, higher customer satisfaction, and reduced developer turnover. Agile development techniques empower teams to overcome time-to-market pressures and volatile requirements. However, replacing traditional process models with something less structured may make it difficult to achieve coordination and coherence in software work.

In general, the mere existence of a software process, whether it be strongly prescriptive or Agile, is no guarantee that software will be delivered on time, or meet the customer's needs, or that it will exhibit long-term quality characteristics. Everyone wants a process that can respond to change; the only debates are over how to design one and which level of discipline should be incorporated into such process models.

Agility requires that customers and developers act as collaborators within development teams. The goal should be to build software products that can be quickly adapted to meet the requirements of a rapidly changing marketplace. This is typically achieved via incremental development of operational prototypes that are improved over time. As a result, Agile software engineering is a way of working and it leverages iterative development, incremental delivery, and ongoing reassessment of products. It is based on a clear idea of the product's concept and its market. It also focuses on high-value features first and on producing tangible, working results after each iteration. Agility principles are summarized as follows:

- Ensure customer satisfaction by delivering software to customers as quickly as possible.
- Accept the fact that requirements may change and work accordingly.
- Deliver software incrementally to stakeholders as often as possible (e.g., every week rather than every month as it is in the case of traditional process models) and use their feedback to improve subsequent increments.
- Minimize the creation documentation to what is absolutely necessary and relevant.
- Build an Agile team that includes motivated participants and facilitate frequent meetings among team members to improve communication and information sharing.
- Create team processes that encourage technical excellence, good design, and simplicity while avoiding unnecessary work.
- Focus on the primary goal of delivering software that meets customer needs.
- Ensure that teamwork is not overwhelming so that team members can be effective over a long period of time.
- Consider the fact that Agile teams need to become self-organizing in order to meet the primary goal of developing solutions that are well designed and implemented to meet customers' needs.

- Instill a team culture that requires all team members to work together with one focus in mind, which is ensuring customer satisfaction.

The Agile philosophy is seductive, but it must be tempered by the demands of real systems in the real world. In general, Agile process models are not suitable for large, high-risk, or mission-critical projects. There is a spectrum of agility that one should consider that addresses these demands as illustrated in Figure 9.4.

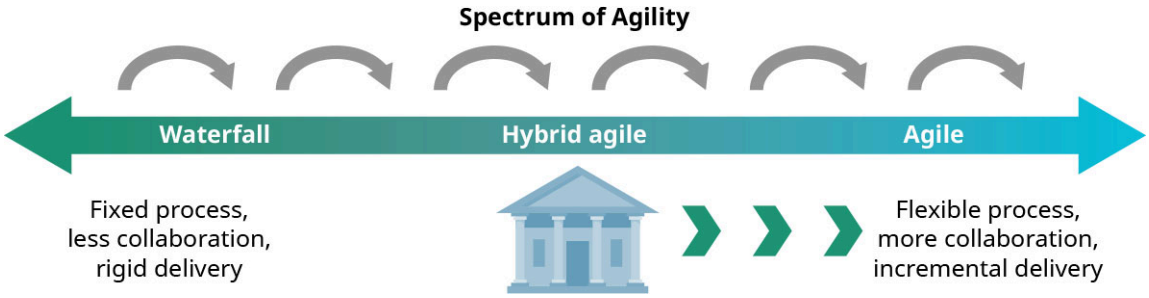


Figure 9.4 The spectrum of agility depicts traditional prescriptive process models such as waterfall on one end, and Agile processes on the other. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

When an **Agile Software Development Ecosystem (ASDE)**, which encompasses the whole category of Agile SDLC frameworks and methods, is compared with traditional (prescriptive) SDLC methodologies, the ASDE emphasizes the difficulty of predicting future needs. Thus, Agile approaches avoid creating long-term plans and fixed processes so developers can instead collaborate with customers and adjust to their current needs.

Many of the ideas related to the Agile approach are worth considering regardless of the process model a team adopts. Agile processes manage unpredictable changes that take place during software development projects. The focus of Agile processes is on the delivery of software increments in relatively short time frames and using feedback on those increments to drive development. There are trade-offs when selecting an Agile Software Development Ecosystem (ASDE). While ASDEs correctly identify the product as the most important outcome of a project, it can be difficult to scale up rapid product cycles to develop enterprise-wide software applications. In general, trade-offs are important for making things work. In particular, the potential problems caused by dysfunctional teams can be significant. The impact of human aspects of process model adaptation should be considered, and all the human factors and group dynamics of Agile teams, including collaboration and self-organizing teams, are important improvements to traditional approaches and are used repeatedly when Agile development is performed.

In conclusion, a software process, regardless of its process centricity, simply must adhere to a set of software process model criteria that are essential to ensure successful engineering of software solutions. To that extent, it is necessary to assess processes and their related activities using actual numeric measures or by applying metrics as part of analytics methods uses to monitor the performance of software process models.

Software Process Framework Activities

A good portion of a software engineer's role is spent within the various framework activities of an SDLC. As such, it is critical for a software engineer to understand the key elements of each of the various framework activities that are used to create software solutions. As you may recall, these framework activities or phases were introduced earlier as inception, elaboration, construction, and deployment. These activities provide a structured approach to creating software solutions. By understanding these framework activities and the tasks involved in each phase, you will be well equipped to contribute effectively to the software engineering process.

Inception Framework Activity

A core precondition to the creation of a solution is to know what must be developed. It is easy to say you want to add automation features to an automobile, but what does that really mean? What are the specific expectations and how do they relate to the solution that needs to be created? In order to create a solution, you have to understand what the solution is expected to do.

The **inception phase** of a project focuses on the gathering and refinement (i.e., definition) as well as the management of functional and nonfunctional requirements, which is also known as requirements engineering. In essence, the inception phase covers the planning activity that lays a project foundation. Here, you will define the project goals, identify the overall scope of the software (what features it will have), and conduct feasibility studies to assess if the project is realistic and achievable. As an example, imagine you are building a recipe app. In this phase, you would decide what features the app should have (like searching for recipes or creating grocery lists) and estimate the time and resources needed to develop it.

Defining requirements must involve stakeholders because they know what the software system should do better than others. Requirements definition involves obtaining the requirements from stakeholders and analyzing/decomposing strategic requirements until you can identify tactical actionable requirements. These will form a foundation for the creation of the analysis model. This definition process is done with either a **use case**, which describes how the software system is expected to be employed by users to accomplish a goal or requirement, or as a **user story**, which is a generic explanation aimed at the user to tell them how a software features works. Requirements management relates to handling changes in requirements and identifying the effect of such changes on the existing set of engineered requirements.

Although the software engineering actions and task sets that relate to the drawing forth of requirements may appear straightforward at first sight, it is in fact one of the trickiest parts of the SDLC. This is due to a gap that always exists between the way stakeholders and business analysts understand the requirements as compared to the way they are perceived by software engineers. This is especially true when you develop a software solution for a particular expert group that use their own terminology; often those who perform a process take some of the actions they do for granted.

The inception phase results in a specification of the system to be developed. This specification is generally incomplete and/or anomalous and is typically refined as part of subsequent process phases (or iterations of such). As a result, there is a blurred distinction between requirements specification, design, and construction.

The actual software engineering actions and task sets that are used as part of the inception phase involve gaining an understanding of the solution context and collaboratively gathering, decomposing, and tracking requirements to help elaborate a preliminary analysis model. Once the preliminary analysis model is created, requirements can then be negotiated and validated with the stakeholders via an in-person Joint Application Design (JAD) session, an approach that involves assembling stakeholders and developers, or through the use of collaborative requirements modeling tools that enable scenario-based modeling. After this, detailed requirements modeling takes place.

Agile requirements definition attempts to accelerate the gathering and analysis/decomposition of requirements. The guidelines it uses to speed this process include:

1. Use simple models such as fast sketches and user stories to allow all stakeholders to participate.
2. Adopt user, client, or expert group terminology and avoid technical jargon whenever possible.
3. Get the big picture of the project done before getting bogged down in details.
4. Refine requirements throughout the project and allow additions and revisions to occur at any time.
5. Implement the most important user stories first and only once its requirements are fully specified.
6. Make the current set of requirements available to all stakeholders so everyone can participate in selecting the features to add during the next development cycle.

Various tools may be leveraged to support the software engineering actions and task sets that pertain to the inception phase (e.g., ReqView[3]).

Requirements Modeling

The software engineering action that is part of the inception phase and focuses on the analysis/decomposition

3 https://www.reqview.com/

of software requirements is called **requirements modeling**. The goal of this action is to answer the question "*What* will the system do?" The focus is purely conceptual, and implementation details are not considered. The main purpose of this analysis is to understand the requirements at a level that makes it possible to design and implement a software system that meets the customer's needs.

As part of requirements modeling, as in business use case modeling, you typically create a domain model that captures the major concepts of the problem domain and associations between them. For example, in the domain of driving assists for an automotive solution, there could be conceptual/analysis classes such as AdaptiveCruiseControl, Car, BlindSpotDetection, and Blinker and class attributes assigned to classes as follows:

- AdaptiveCruiseControl has attributes state (on/off) and desiredSpeed (the requested speed).
- Car has attribute speed (the current speed).
- BlindSpotDetection has attribute state (on/off).
- Blinker has attribute state (on/off).

Figure 9.5 illustrates a class diagram of a partial domain model. This diagram is specified using **Unified Modeling Language (UML)**. UML is often used for modeling in projects as its visual representations provide clear, compact means of communicating among the developers. In UML class diagrams such as this, the numbers at associations are multiplicities, and the arrows specify the direction in which you are expected to read the association. For example, BlindSpotDetection monitors Car.

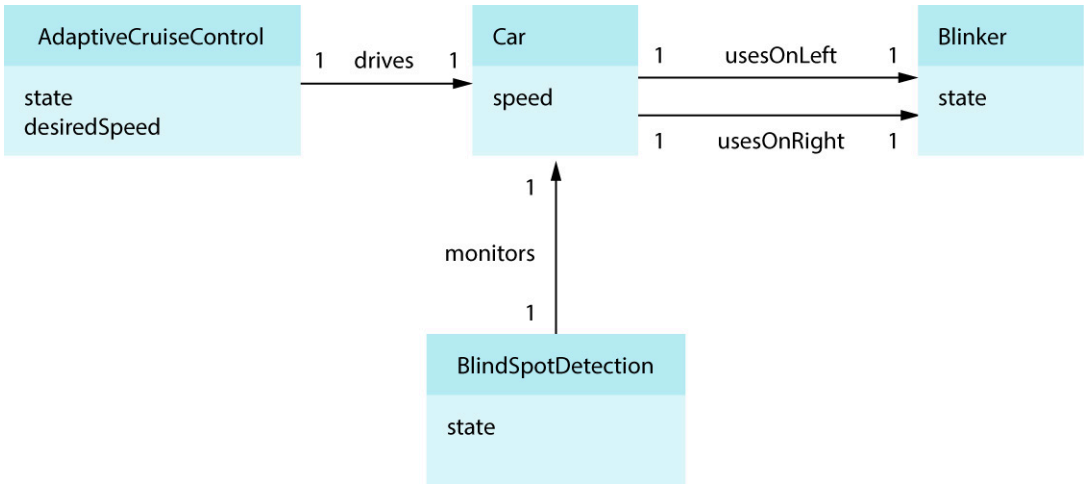


Figure 9.5 A UML class diagram, in this case of a partial domain model, is used as a communication tool among developers. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Associations are links that deserve to be stored in the system. For example, an association between AdaptiveCruiseControl and Car is required because when Adaptive Cruise Control is active, conceptual class AdaptiveCruiseControl needs access to the car. The numbers at associations are multiplicities, and they say that there is exactly one Car associated with each AdaptiveCruiseControl and that there is exactly one AdaptiveCruiseControl associated with each Car. Associations can have names, which facilitate reading and understanding the analysis model.

The UML notation may be applied to provide different perspectives as follows:

- Conceptual perspective: The diagrams describe real-world concepts or things.
- Specification (software) perspective: The diagrams describe software components.
- Implementation (software) perspective: The diagrams describe software components in a particular technology, such as Java or .NET.

We typically use all perspectives throughout the software development life cycle: conceptual perspective to capture requirements, specification perspective to describe the design, and implementation perspective to clarify implementation details.

When doing requirements analysis, you can create specific outputs/work products (also referred to as artifacts), such as use cases, scenarios, and the domain model. Use cases can be captured in plain text or via a UML Use Case diagram. A Use Case diagram consists of an actor, which represents users of the system, and the use cases that this user is expected to use. In Figure 9.6, the actor is a driver who might be seeking to turn on cruise control. Each use case is then described in detail using one or more scenarios. A **scenario** is a specific instance of operational flow within a use case that is focused on understanding a specific action. Scenarios are written either in a plain text or as a sequence of steps that describe a specific scenario instance within a use case (i.e., main scenario describing the most productive set of steps versus alternative scenarios that capture unexpected behavior).

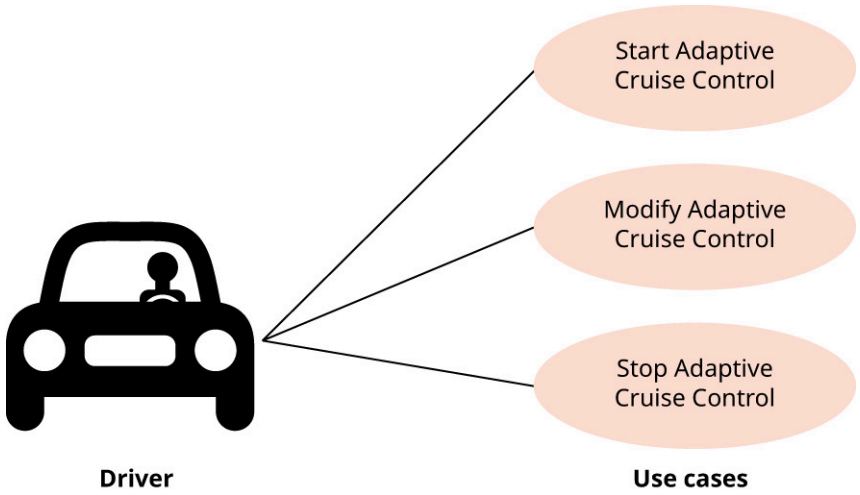


Figure 9.6 In this UML use case diagram illustrating an actor called "Driver," the three use cases might be further grouped together and categorized as Car Control system. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

As part of requirements modeling, UML diagrams are drawn whenever they bring value to help provide a conceptual perspective of what the solution is meant to accomplish. They are generally not required to be complete or perfect. The use of UML in this manner is typically referred to as "UML as sketch," and it involves informal and incomplete diagrams. Instead of drawing UML diagrams, it is possible to specify them via scripts to automate the creation of diagrams, which can save valuable time. As mentioned, the main goal of requirements analysis/decomposition in the inception phase is to understand the problem while the main goal of the elaboration phase, which comes next and involves software design, is to clarify what we are to implement.

Elaboration Framework Activity

The **elaboration phase** further analyzes the requirements to produce design models of the system to be developed. In this phase, you take a deeper dive into the specifics of the software. Requirements are refined in detail, a detail design is created that outlines the architecture of the software, and the potential risks associated with the project are identified and assessed. Design models are defined at a high-level initially to represent the various facets of the architecture of the solution that is being developed at a given level of scope. The scope could be that of a whole enterprise, a portfolio of solutions contemplated by a business unit, or a specific solution being developed by a business unit as part of a given project. Architectural facets are typically based on architectural domains specified in mainstream architecture frameworks. For example, The Open Group Architecture Framework (TOGAF) splits high-level architecture representations into four domains: business architecture, application architecture, data/information/knowledge/wisdom architecture, and infrastructure architecture. Various high-level modeling languages and associated tools may be used to facilitate the creation of high-level architecture models (e.g., TOGAF's Archimate Certified Tools[4]). The management of enterprise and solution architectures is described in more detail in Chapter 10 Enterprise and Solution Architectures Management of this book.

A detailed-level design model may then be derived from the high-level architecture model, and it is typically represented using a combination of low-level modeling languages (e.g., BPMN, UML, SysML). At this level of design, a conceptual solution that fulfills the requirements is created and seeks to answer the question "How will the system fulfill the requirements?" The conceptual solution leverages the inputs collected in the inception phase to design a software product. This information is generally organized into two types of design: logical and physical.

Logical design ignores any physical aspects. For example, a cruise control system needs to keep track of the maximum speed selected and whether the cruise control system is on or off. This information is gathered as part of the logical design and needs to be captured via a diagram that also identifies the corresponding relationships. This type of information is often captured as a set of entities (or actors) that enable the grouping of descriptive information and attributes.

A graphical representation of the method for effectively implementing what was determined in the logical design of a software solution is a **physical design**. It includes defining where information comes from and where it goes within the planned system. It includes defining how information is obtained, processed, and/or how it is presented. For example, the physical design for turning on or off the cruise control system within an automobile can include using controls made available to the driver on the steering wheel to turn on or off the cruise. There could also be controls via pressing the brake pedal to turn off the system.

Work on the logical and physical designs is generally performed first as a high-level design software engineering action followed by a detail-level design software engineering action. The focus of the **high-level design (HLD)** software engineering action is on providing a general description of the overall system design, and can include information on the overall aspects of a system including its architecture, data, systems, services, and platforms as well as the relationships among various modules and components. Its focus is to convert the requirements into a high-level representation of the solution that can then be further refined as part of detail-level design.

The focus of the **detail-level design (DLD)** software engineering action is to detail or expand upon the HLD. As part of the DLD, every element of a system is provided with detailed specifications, and the logic for each component within each module of a system solution is determined. DLD is then used to implement the actual solution as part of the construction phase of the software process.

Some of the differences between HLD and DLD are:

- HLD gives high-level descriptions of functionality, whereas DLD gives details of the functional logic within each component of a system.
- HLD is created first, with DLD created as an extension of the HLD.
- HLD is based on the requirements of the software solution, whereas DLD is based on extending the HLD. DLD should, however, still align with the requirements.
- HLD provides elements such as data and information design, whereas DLD provides the information needed to create the actual programming specification and test plan for using the data.
- A solution architect is generally involved with the HLD. Programmers and designers are generally involved with DLD.

Software Architecture Work Product

The software architecture work product acts as a blueprint of the solution being worked on. Imagine you're building a house. Before construction begins, you would create a blueprint that outlines the overall structure, major components (foundation, walls, roof), and how they fit together. This blueprint is similar to a software architecture. In software development, a **software architecture** provides a high-level overview of a software system. It describes the system's major components, their interrelationships, and how they work together. This high-level representation helps developers understand the overall design before delving into details.

4 https://www.visual-paradigm.com/features/archimate-tools/

A solution is typically represented at various levels of abstraction. Software design involves using software architectures to represent solutions at a high-level of abstraction. A software architecture constitutes a relatively small, intellectually graspable view of how a solution is structured and how its components work together. The goal of software architecture modeling is to allow the software engineer to view and evaluate the system as a whole before moving to component design. This step enables the software engineer to:

- ensure that the design model encompasses the various solution requirements
- make it possible to survey various design alternatives early on to facilitate the adoption of the best possible model
- limit the risk of building software that does not meet the requirements

When you look at a blueprint, you can see the major elements of a building and their relationships to each other. When you look at a software system at a high-level of abstraction, such as the extremely high-level shown in Figure 9.7 of a web browser, you can see its major components and the main connections between them. For example, at a high-level, a web browser connects to a web server via HTTP requests, and the web server interacts with a relational database via SQL queries to create an HTML web page that is rendered dynamically and sent back to the web browser for display.

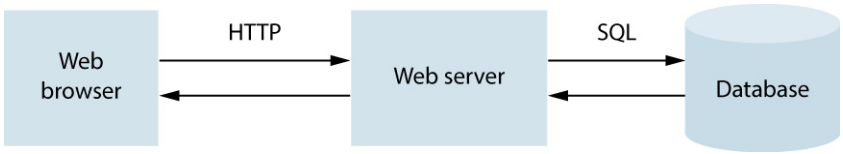


Figure 9.7 The software architecture displays a system's major components (in this case, web browser, web server, and database) and connections (HTTP requests and SQL queries) at a high level of abstraction. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Software architecture is an important part of the creation of a software solution, and it should always be designed by an experienced software engineer because changes in the software architecture typically have drastic effects on the solution implementation such as constructing solutions that do not meet the nonfunctional requirements.

When designing software architecture, you can leverage various types of software patterns that facilitate the reuse of preexisting solutions. Two examples of such patterns are an architectural style and architectural or design pattern. An architectural style is a transformation that is imposed on the design of an entire system. The intent is to establish a structure for all components of the system. Architectural or design patterns also impose a transformation on the design of an architecture, but they differ from a style because they operate at a lower level of abstraction. Patterns can be used in conjunction with an architectural style that shapes the overall structure of a system.

A software architecture is one of the work products that results from the HLD software engineering action. Software architectures are important work products because they provide high-level representations of solutions that facilitate communication among all stakeholders. They also highlight early design decisions that have a profound impact on all software engineering work that follows

When it comes to Agile software processes, the goal of creating software architecture work products as part of the HLD software engineering action is to avoid rework. In that case, user stories are leveraged to create and evolve an architectural model (sometimes referred to as a "walking skeleton") before constructing the software. The use of Agile software processes sometimes makes it difficult to manage architectural design especially when the team is developing large systems from scratch rather than adding small services piecemeal to an existing system. Agile approaches typically focus on small increments for which it may be difficult to produce an all-encompassing architecture that will be able to accommodate subsequent increments that have not been completely defined yet. This may lead to having to refactor the architecture, which could be very costly, as it may require changing a lot of the code that has already been developed. Therefore, it is recommended that teams thoroughly consider architectural design when taking on large projects that do not

build on an already defined and solid architecture. This brings up the question as to whether big design up front (BDUF) is the preferred method in this case. It is also the reason Agile teams should include software engineers with a strong background in architecture (ideally, (enterprise architecture), who can foresee the type of designs that are required to avoid costly refactoring efforts in the future.

The Scrum and Kanban Agile process models, as you'll learn later in the chapter, allow software architects to add user stories to the evolving storyboard and to work with the product owner to prioritize their architectural stories in work units called sprints. Well-run Agile projects include the delivery of software architecture documentation during each sprint. After the sprint is complete, the architect reviews the working prototype for quality before the team presents it to the stakeholders in a formal sprint review.

As mentioned earlier, the focus of the HLD software engineering action is on providing a general description of the overall system design. It can include information on the overall aspects of a system, including its architecture, data, systems, services, and platforms as well as the relationships among various modules and components. Its focus is achieved by converting the requirements into a high-level solution that can then be further refined as part of the Low-Level Design (LLD) software engineering action.

LINK TO LEARNING

The IEEE Computer Society has proposed IEEE-Std-42010:2022, *Software, Systems and Enterprise—Architecture Description* (https://openstax.org/r/76IEEEEStd) as a standard that describes the use of architecture viewpoints, architecture frameworks, and architecture description languages (ADLs) as a means of codifying the conventions and common practices for architectural description.

Software Design

The abstraction and refinement of requirements into a specification that can be used to help create a software solution is referred to as **software design**. Software design is a software engineering task set that is part of the DLD software engineering action. The focus of the DLD software engineering action is to detail or extend the HLD work products. As part of the DLD software engineering action, every element of a system is provided with detailed specifications, and the logic of each component within each module of a solution is determined.

Software design generally requires problem-solving skills as well as the ability to conceptualize framing as well as define it into a working specification that can be used to create a software solution. In contrast to the HLD software engineering action, the LLD software design task set is concerned with all the implementation details. In Agile approaches, we typically postpone many design decisions until implementation time and only design up front the parts that are tricky or that need to be solved in an unusual way.

The main concepts that drive software design are:

- Abstraction: high-level representation of components such as data (or data objects) and procedures (the sequence of instructions that usually have specific and limited function)
- Architecture: overall structure or organization of software components, ways components interact, and structure of data used by components; component-based software engineering (CBSE) can be considered as a task set as part of the DLD software engineering action to assemble solutions based on the reuse of preexisting components
- Design patterns: a design structure that solves a well-defined design problem within a specific context; pattern-based design can be considered as a task set as part of the DLD software engineering action to assemble solutions using implementation patterns and frameworks
- Separation of concerns: a technique based on the idea that any complex problem can be more easily handled if it is subdivided into pieces
- Modularity: compartmentalization of data and function

- Information hiding: controlled interfaces that define and enforce access to component procedural detail and any local data structure used by the component
- Functional independence: single-minded (high cohesion) components with aversion to excessive interaction with other components (low coupling)
- Stepwise refinement: incremental elaboration of detail for all abstractions
- Refactoring: a reorganization technique that simplifies the design without changing functionality
- Design classes: provide design detail that will enable analysis classes to be implemented

The work products generated as part of the DLD software engineering action are used to construct the actual solution. There are many other task sets that could be considered as part of the DLD, including prototype, user experience and user interface design, and specific design task sets for web, mobile, social, and gaming solutions design.

THINK IT THROUGH

Asking the Right Questions

In 1998, the Mars Climate Orbiter was launched toward Mars to study its climate. Unfortunately, the lander was never able to complete its task. The probe ended up being destroyed by the Martian atmosphere due to an error in the mathematical calculations. However, the issue was not with the calculations but the numbering system that was used for distance measurements. The software in the probe used the metric system but the team from Earth that sent the data to the probe sent their values in imperial numbers. The result was a destroyed probe.

Was this a design error? What could have been done to avoid a mistake such as using the wrong measurement system?

Construction Framework Activity

As part of the **construction phase** or framework activity, the design documents that result from the DLD software engineering action of the elaboration framework activity are used to write corresponding source code in a programming language as well as create any supporting assets such as deployable container images, databases, and controls. This is where the coding magic happens! Developers work to build the software based on the design created earlier. This phase involves writing code, testing individual components, and fixing defects/bugs.

Source code may also be automatically generated from design work products using round-trip engineering tools, although this type of functionality is still limited today. The development methodology model-driven engineering, which is compatible with Agile methods, stresses the importance of formal and executable specifications of object models and the ability to verify the correctness and completeness of the solution by executing the models. This is typically made possible when using round-trip design engineering tools and frameworks that allow for the specification of models using standard modeling notations and the creation of evolvable software from these models (e.g., jBPM[5]).

The result of the construction phase is a complete running solution that is based on the design work products and meets the expectations set forth as part of the inception phase. It should be noted that coding is a mechanistic outgrowth of procedural design, and errors can be introduced as the DLD design work products are translated into a programming language. This is particularly true if the programming language does not directly support data and control structures represented in the design. Code walk-throughs are designed to avoid this.

5 https://www.jbpm.org/

The field of DevOps, a wide-ranging collection of development and operations practices, has introduced further processes and infrastructure to automate many of the software engineering actions that are part of the construction phase. When these methodologies have been applied together, Agile methodologies and DevOps' automation have increased the speed, robustness, and scalability with which software can be constructed.

Software engineers use many tools to implement solutions. These include code editors such as Atom, Integrated Development Environments (IDEs) such as VSCode or Visual Studio, version control systems such as Git, debugging tools, testing tools, and more.

LINK TO LEARNING

Which programming language to learn often depends upon the type of programming that needs to be done or what a business currently uses. While there is no perfect way to determine which programming language is used the most, the Tiobe Index (https://openstax.org/r/76TiobeIndex) gives a general idea of which languages are most popular. This index is updated monthly and provides an indication of the changing popularity of programming languages. The rankings are based on factors including the frequency of searching on related topics, courses taught, and more. Of course, each programming language has strengths and weaknesses, so when choosing which language(s) to learn, sometimes it best to focus on the context of the solution you need to create and not just popularity.

Unit, Integration, and System Testing

Unit, integration, and system testing deal with ensuring and verifying that the software system works as expected. It typically involves activities to uncover errors that were made inadvertently during the elaboration and construction phases. Code reviews and unit, integration, and system testing are typically done as part of the construction phase by software developers responsible for writing source code. The process whereby the source code written by one developer is manually inspected by another developer is called **code review**. This is especially useful when the software team consists of developers with different levels of experience.

It is worth noting that while SDLCs often include testing task sets within specific phases, testing is an activity that should happen repeatedly throughout the software process independently from the actual phase (or iteration of such) that is currently being executed.

Deployment Framework Activity

Deployment is the phase that makes software available to users. Finally, the software is released to the users! This phase involves delivering the software to its intended audience (e.g., launching a mobile app on an app store) and providing ongoing support to address any bugs or issues that arise after deployment. In the past, **deployment** often meant installing the software on a customer's computer. Today, software is more likely to be installed on special computers or powerful online systems called cloud servers. When deploying software, some configuration might be needed, especially for complex applications. DevOps uses automation to make deployment faster and more reliable. Modern software is often updated frequently using special tools and techniques. As noted, deployment, especially if it is a nontrivial task that is not expected to be done by the customer, typically involves configuring the software solution. In a typical configuration process, the software solution is **containerized**, or packaged in such a way that it can run on different computer systems easily, and then deployed via a system, called Kubernetes clusters, that automatically manages scalability and availability.

The field of DevOps has introduced additional processes and infrastructure to automate many of the software engineering actions that are part of the deployment phase. Together, Agile methodologies and DevOps' automation have increased the speed, robustness, and scalability with which software can be deployed today. It is worth noting that modern applications' deployment techniques have evolved quite a bit as a result of

DevOps' automation. Software updates are deployed frequently today using continuous integration and deployment (CI/CD) techniques and tools.

Maintenance

Most deployed software will eventually need updating to add new requirements or fix issues that might arise. The process of updating software after it is deployed is referred to as **maintenance**. As mentioned earlier, maintenance is a software engineering action that is part of the software process deployment phase. The cost of maintenance can exceed that of development, especially if software remains in use for a long period of time.

You may think of the process of creating software as being analogous to that of building a new car. Maintaining the software is like maintaining the car while you use it. To make the analogy more precise, while software does not wear out like car hardware might, its maintenance involves updates to fix bugs and adding new features as requirements change. Building a new car may take several weeks, but car maintenance will probably last much longer because the car may be used for years. As for expenses, building a new car will cost a significant amount, but costs related to its maintenance will easily exceed the price of the car when it was new. Of course, when the costs of maintaining a car become too high, there is always the option of buying a new car, just as there is the option to build a new software product if maintenance of the legacy software gets too high.

In Agile process models, a lot of the maintenance is not limited to adding new features. Instead, it often involves the following:

- using Scrum sprints to plan the work and address customer needs without overwhelming developers
- giving priority to urgent customer requests and allowing corresponding interruptions of planned maintenance sprints to address these urgent requests
- making it possible for team members to prioritize the handling of customer requests and coordinate their processing as part of the maintenance process
- combining the use of meetings and written documentation to minimize the duration and frequency of meetings and keep them focused
- relying on informal use cases when communicating with stakeholders to supplement existing documentations and keep communication simple
- requiring that developers verify each other's work; in particular, experienced developers should review the work produced by junior developers, such as defect fixes or code added to support new features, to help them develop their knowledge

Crosscutting/Umbrella Activities

In addition to the core process framework activities, namely, inception, elaboration, construction, and deployment, there are many activities that can take place at any point during the creation of a solution and throughout the entire software process—in other words, they crosscut the process. To understand the importance of such activities, consider this analogy. In addition to completing the main stages of building a house (e.g., constructing the foundation, walls, roof), there are other important activities that happen throughout the project (e.g., choosing the colors of wall paint). These are like the umbrella you would use on a rainy day—they support the entire process but aren't part of the main building steps themselves. In software development, a **crosscutting activity**, or an **umbrella activity**, is an activity that crosscuts the entire software development process but is not part of the main building steps themselves. They can include communication and training, risk management and planning, software configuration content management, quality management, architecture management, and software security engineering.

Communication and Training

Establishing communication involves scheduling regular meetings between developers and customers and

also meeting with developers to train them on new technologies. It is necessary to communicate with stakeholders and customers at various points in the software development process. For example, collecting project requirements during the inception phase involves communication and coordination between project managers and stakeholders. Release notes serve as another form of communication, and they are written for software users to make them aware of the features that are included in a new release. More generally, the status of a project needs to be communicated with management at regular intervals to make sure that satisfactory milestones are reached according to plan.

Regular communication also happens within software development teams. For example, design reviews encourage communication to help finalize designs. Code reviews focus on communicating how the system is changing, and how to solve problems and improve code. Daily stand-ups in Agile software processes are about concise verbal communication.

Software engineering team members must undergo training on a regular basis to acquire or maintain certain skills. To support this, software development teams typically put in place organizational change transformation methodologies and frameworks (e.g., Prosci 3-Phase process) to manage their ability to successfully conduct projects given the changing nature of software engineering.

Risk Management and Planning

Risk management and planning focuses on identifying potential risks like project delays and having a plan for how to address them. Software development is a complex activity that involves many people working over a long period of time, and it turns out that not every project succeeds. Some projects are delayed, some of them overrun the budget, and some are never finished. The high percentage of unsuccessful projects creates a need for risk management and mitigation. Minimizing risk is typically the main task of a manager, but software engineers can also take on management tasks. For example, to help managers, they might provide updates that allow managers to refine their risk assessments. Ideally, these updates would address the different components of risk, including:

- Performance risk: considers whether the product will not fit its intended use
- Cost risk: determines if the budget constraints can be maintained
- Support risk: assesses how easy the product will be to maintain and update once it is completed
- Schedule risk: considers whether the project will meet expected deadlines

Risk projection attempts to associate with each risk the likelihood of its occurrence and the consequences of the resulting problems if a risk should occur. One of the software process models you will learn more about later in this chapter is called the Unified Process. In this process, risks are mitigated by selecting risky requirements for early iterations. For example, a use case that requires a new technology is typically considered risky, as well as a use case that assumes integration with legacy code. We select such use cases for an early iteration because if their implementation happens to fail, it is better to fail in the beginning of the project rather than in the end.

As the project continues, managers focus on minimizing risk across the four Ps: the *p*eople involved, the *p*roduct being developed, the *p*rocess being followed, and the *p*roject work being completed. While the Agile and traditional software processes use different approaches, the goals for each are the same: controlling risk by providing people with a well-defined product and clear processes to follow. These goals allow software engineers to estimate the work required and track the product through development. Managers compare the product completed against those estimates and use those results to make any needed adjustments.

TECHNOLOGY IN EVERYDAY LIFE

Delivering Viable Systems

The development of software often focuses on three areas: desirability, capability, and viability. In other words, the focus is on what is wanted, what is possible, and what will sell or help a business to function. To be successful, a software product must deliver something people want and is of value. It must also be possible to implement the product.

In the 1980s, the first virtual reality software was released. VR was something that many people wanted and saw a value in having; however, the technology was not capable of delivering a viable system. It is only today that software and hardware capabilities can support VR at a level that makes it possible to deliver products that people are willing to pay for.

What are some other technology areas that are viable today that were not viable ten years ago? What are some software technologies that are not viable today that could be viable within the next ten years?

Software Configuration and Content Management

Software configuration management (SCM) is a crosscutting activity that helps report, identify, and control change to items that are under managed development. These items are referred to as Software Configuration Items (SCIs). SCM also analyzes the implementation of change and provides mechanisms to publish and deploy change. One way to reinforce the importance of configuration management without a real customer is to change the project requirements sometime after the project implementation begins.

The focus of SCM tends to be on four main areas:

- Configuration identification: the identification of all components within a project, including any files, documents, source code files, directory structures, and more
- Configuration change controls: the controlling of who accesses elements of a project and the tracking of changes being made
- Configuration status accounting: the tracking of who made a change as well as when they made the change and why it was made
- Configuration auditing: the tracking of the status of a project and, more important, the tracking and confirmation that what is being created matches what is required

Many Agile teams make use of continuous integration to ensure that they always have viable prototypes ready to test and extend. The advantages of continuous integration are as follows:

- involves frequent feedback to notify developers promptly when integration testing fails so they can fix issues as quickly as possible, especially if the number of fixes required is small
- improves quality by being able to address product changes quickly; as a result, users can trust that the product meets their needs
- reduces risk by avoiding long delays between the time software is developed and its integration into the product; this ensures that design failures can be detected and addressed early on
- involves up-to-date reporting to ensure that software is correctly configured to conform, for example, to the latest code analysis metrics
- ensures that streamlined integration is used as key support technologies in organizations that use Agile software process models
- captures defects as early as possible in the software engineering process, which limits the cost of software development

Various tools may be used to support SCM, including audit management tools (e.g., ZenHub), configuration

management/automation tools (e.g., Ansible, Vagrant), continuous integration tools (e.g., Jenkins, Travis CI), dependency tracking and change management tools (e.g., Basecamp, Jira), source control management tools (e.g., GitHub), and so on.

Content management includes collection, management, and publishing subsystems. The collection subsystem facilitates the creation and acquisition of new content. It also makes it possible for humans to relate to the content and combines it as units that can be displayed more effectively on the client side. The management subsystem provides a repository for content storage capabilities, including the content database (i.e., the information structure use to organize all the content objects), the database capabilities (e.g., functions to search for content objects, store and retrieve objects, and manage the content file structure), and configuration management functions (e.g., supports content object identification, version control, change management, change auditing, and reporting). The publishing subsystem extracts content from the repository, converts it to a publishable form, and formats it so that it can be displayed in a web browser (e.g., Chrome, Safari). The publishing subsystem uses a series of templates for each type, including static elements (e.g., text, graphics, media, and scripts that require no further processing are transmitted directly to the client side), publication services (i.e., function calls to specific retrieval and formatting services that personalize content, perform data conversion, and build appropriate navigation links), and external services that provide access to external corporate information infrastructure, such as enterprise data or “back-room” applications.

Software Quality Management

Whereas testing validates that things work as expected and that there are no errors or issues, **Software Quality Management (SQM)** focuses on the development and management of the quality of the solution being developed. Tasks within SQM involve quality planning and quality control. They include, but are not limited to, activities such as:

- confirming requirements are correct, complete, and consistent
- verifying that all elements of design conform to the requirements and are of high quality
- confirming that source code follows coding standards and is written in a manner that will be maintainable going forward
- ensuring that testing checks all elements of a solution
- implementing a change management plan

Engineering quality software subsumes a deep understanding of the solution requirements and the ability to design work products that conform to these requirements. These activities must rely on the use of software engineering best practices and must be supported by adequate project management.

Assessment reviews (e.g., system engineering assessments, software project planning assessments, analysis models assessments, design models assessments, source code assessments, software testing assessments, and maintenance assessments) are an important quality assurance mechanism. Software quality assurance (SQA) is part of a broad spectrum of software quality management activities that focus on techniques for achieving and/or ensuring high-quality software.

Architecture Management

Returning to the analogy of how software architecture is like the blueprint of a house, software architecture management can then be likened to improving the blueprint as you use it to build the house. To put it another way, creating the blueprint (architecture) was just the first step. In software development, architecture management involves keeping track of the blueprint and making sure it stays up-to-date as the software is being built. This helps avoid problems later on. While HLD is a software engineering action that takes place in the software process elaboration phase, the **architecture management** umbrella activity encompasses a set of architecture management and architectural refinements techniques that can help improve the architectural design while it is under development.

Architecture management efforts may be performed at any point within the software life cycle, which explains why architecture management is a good umbrella activity; it maintains the knowledge required to qualify the "goodness" of solutions from a design standpoint, and it is handled separately from the quality management umbrella activity.

There are special tools that can help with architecture management. Similar to using software to draw up the blueprint of a house, these tools can help store and organize the architecture information and make it easier for everyone working on the project to understand it. Examples of such tools include artifact/package management tools (e.g., Docker Hub[6], JFrog Artifactory[7]), and pattern catalogs.

Software Security Engineering

Engineering software security focuses on protecting software assets against threats. Threats typically exploit software vulnerabilities to compromise the confidentiality and integrity of data. Threats may also compromise the availability of software systems by disrupting access to system services and related data.

Software architectures must be designed to address security requirements and eliminate vulnerabilities that can lead to exploits. Various design techniques can be used by software engineers to address the possibility and the effects of attacks in order to minimize related losses and costs. As an example, Microsoft's SQUARE process model provides a means of eliciting, categorizing, and prioritizing security requirements engineering for software intensive systems.

Keeping up with cybersecurity threats is proving to be difficult for businesses these days due to a lack of trained resources and increased demand for security compliance. Traditional approaches to security are no longer viable to ensure that organizations can keep operating as well as develop competitive solutions. For that reason, many businesses are combining traditional software process models or Agile process models with modern approaches, such as DevSecOps, to manage software security engineering. Using DevSecOps requires the adoption of new processes and tools as well as the training of staff members. The DevSecOps approach automates the support of security throughout the SDLC, which reduces time and costs of development and facilitates the integration of the security and development teams.

Some examples of DevOps security tools are Aqua Security and HashiCorp Vault, and examples of DevSecOps tools are SonarQube and XebiaLabs.

Popular Software Process Models

The framework activities (or phases) that have been presented as part of the process framework are general phases that get applied within software process models/SDLCs. The types of software engineering actions that get applied with each phase depend on the software development model that is used for the project at hand.

There is a multitude of SDLC models. These models have evolved over time and offer various approaches to creating software solutions. Some more traditional SDLCs are prescriptive in terms of the software engineering actions that must be conducted, while others are agile. Agility, as you know by now, has to do with the ability to skip some software engineering actions or make some of the deliverables optional in order to meet deadlines and still deliver a quality product within budget constraints. As you may recall from Figure 9.4, there is a spectrum of agility between software process models. By nature, SDLCs are incremental as it is always possible to consider a subset of requirements for a given release. In fact, there are typically as many increments as there are subsets of requirements. To accommodate changes in requirements and possibly new requirements within an increment, SDLCs can involve iterations that make it possible to add to and replan increments on an ongoing basis. This adding and replanning may introduce backlogs because, usually, the original timeline cannot be changed.

In short, SDLCs can be made agile, incremental, and iterative. Historically, traditional models were incremental

6 https://hub.docker.com/
7 https://jfrog.com/artifactory/

but not iterative or agile. The Unified Process (UP) model was the first traditional model to introduce iteration and it was quite prescriptive and, therefore, not agile, in terms of expected deliverables. Agile software process models are always incremental and iterative. That said, it does not make sense to use an Agile software model if the requirements are known and not expected to change during the increment. In that case, using out-of-the-box solutions may, with the help of some collaborative features found in Agile process guidelines set forth in agile ASDEs/SDLCs, produce better results. Traditional software process models follow a step-by-step plan, akin to building a house according to a blueprint. They are good for projects with clear requirements that don't change much. While some organizations define the software development model their software engineers are expected to use, it is almost often better for teams to pick or tailor a software process model, so it aligns with the project at hand.

Some of the popular software process models include:

- Waterfall model
- V-model
- Incremental model
- Prototyping model
- Spiral model
- Unified Process (UP) model
- Agile Process models

Waterfall

Predominantly used in the early days of software engineering, the **waterfall model** is a continuous prescriptive software process model in which phases "flow" into another the way water flows from the top of a waterfall down to the bottom. In the waterfall model, the requirements are first gathered and analyzed, then a complete software system is designed, the system is implemented, and then final testing is done before the system is finally deployed. Unfortunately, the traditional waterfall model did not make a distinction between phases and software engineering actions, and, therefore, the steps it uses correspond to specific software engineering actions or task sets with custom names that should be conducted in a prescribed sequence. Our generic process framework may still be used to represent the waterfall process, but framework activities would have to be ignored and specific software engineering actions or task sets that correspond to the waterfall phase names would have to be used. For example, some of the testing task sets from the SQA software engineering action of the Quality Management umbrella activity could be pulled together to make up the waterfall testing phase.

In the waterfall model, software engineering actions or task sets are performed in a strict order as shown in Figure 9.8, and each one of these has a required output in the form of artifacts, such as a document, diagram, or code. This software process is not Agile, so it is not possible to skip a step or drop a deliverable. For example, the output of requirements analysis is a document that describes all requirements for the system. Because this process is not iterative, it is not possible to go back to a previous step to modify this output. For example, if the final design document contains a mistake, it could not be revised during the implementation step—the next step would be testing. Note, however, that there is nothing that keeps the waterfall model from being used incrementally. The concept of incremental development was simply not understood when waterfall was first used.

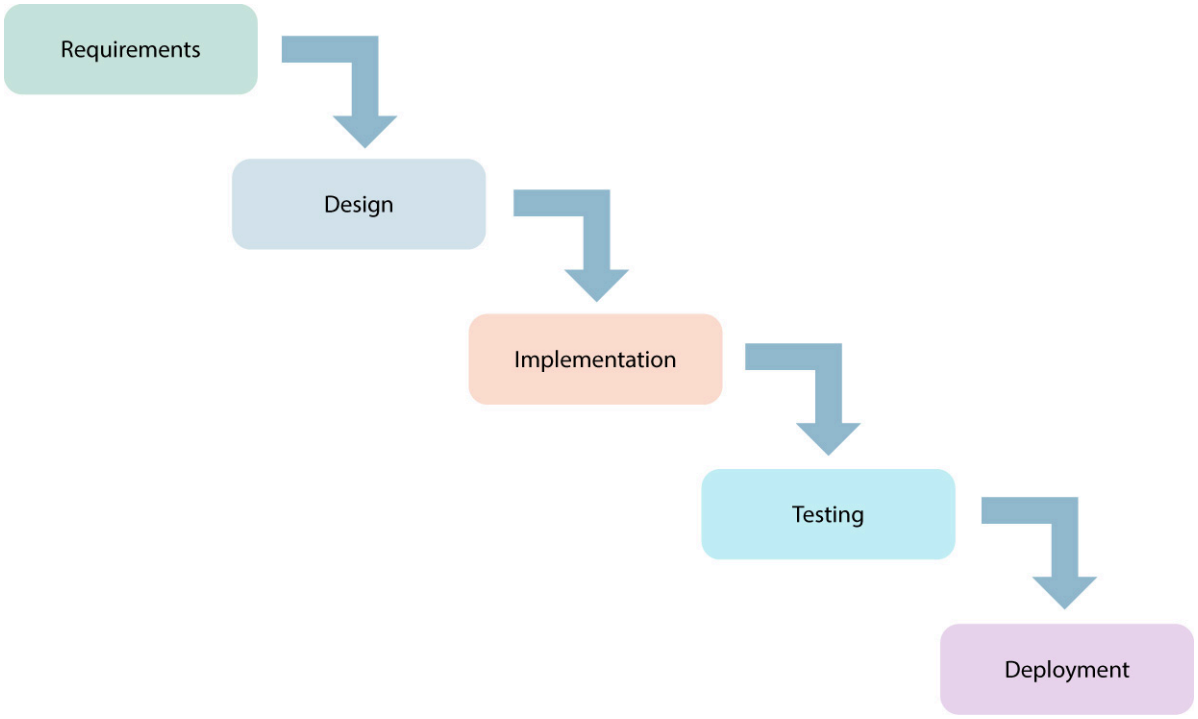


Figure 9.8 In the waterfall model, one step of the software development process "flows" into another, and each is required to produce an output in the form of a document, diagram, or code. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The major advantages of the waterfall model include:

- It is easy to understand and use.
- Steps and corresponding software engineering actions or task sets are conducted sequentially.
- The artifacts are well documented.

Although the waterfall model has some advantages, they are often outweighed by its disadvantages:

- It cannot easily accommodate changes in requirements. If there is a change in requirements, it is necessary to go back to the first step of the process model and update all the artifacts that were previously completed.
- No software product is provided until late in the life cycle. Because software is not available until the end of the implementation/construction step, it is not possible to ask the customer for feedback during the process.
- It produces many artifacts, which are not always necessary; therefore, a lot of time may be spent creating unnecessary artifacts.

Despite these disadvantages, the waterfall method is useful in situations where requirements do not change and interaction with the users of a system is limited or nonexistent during a project.

The V-Model

The **V-model** for software development is also known as the verification and validation model. The V-model is similar to the waterfall model in that it is a continuous prescriptive model that includes basic initial system creation steps starting with gathering requirements, designing the system, and coding it. Each step is prescriptive and conducted sequentially. Where the model differs is that each step of the V-model is associated with a verification or validation testing step/phase, as shown in Figure 9.9. This testing is planned in coordination with each of the design and implementation steps/phases.

Like the waterfall model, the V-model is considered easy to understand and use because it follows a specific flow when it comes to steps/phases, and each step/phase is only completed once. Also, it is best suited for

smaller projects where the requirements are easy to understand and unlikely to change after the project starts. The V-model's advantage over the waterfall method is that verification and validation testing is more integrated into the overall process.

The V-model, however, does have several disadvantages, including:

- It is not good for larger, longer projects or projects that may involve changing requirements.
- A usable software product will not be available until near the end of the software development life cycle.
- Once testing is started, it becomes more difficult to make changes to the design.

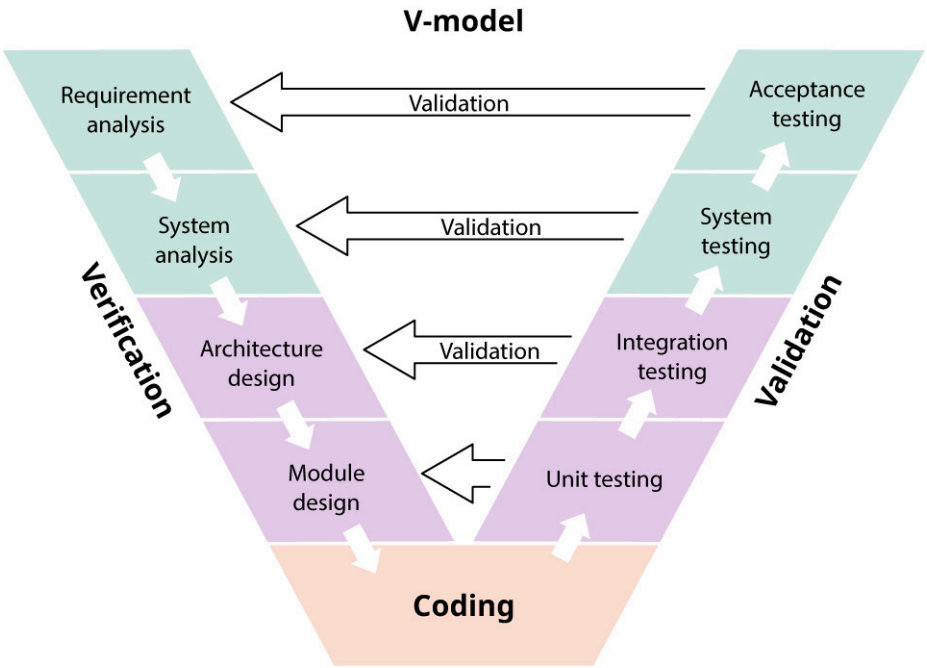


Figure 9.9 The V-model is like the waterfall model in that it is also a continuous prescriptive software process model. Despite their visual differences, in both models, each step is conducted sequentially. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Incremental Model

In the **incremental model**, the software process is divided into modules. Each module focuses on a smaller set of requirements based on an overall business plan. These smaller sets of requirements are then used to design, implement, and test that part of the software solution, as shown in Figure 9.10. Once all the modules are completed, the software solution is deployed to the users. Note that increments in this case are different from iterations. It is assumed that each increment focuses on a small subset of requirements, and it is not possible to accommodate possible changes to requirements. Projects are typically split into a number of increments meant to cover customers' needs over a period of time that is acceptable to them, while each increment is made manageable by the development team.

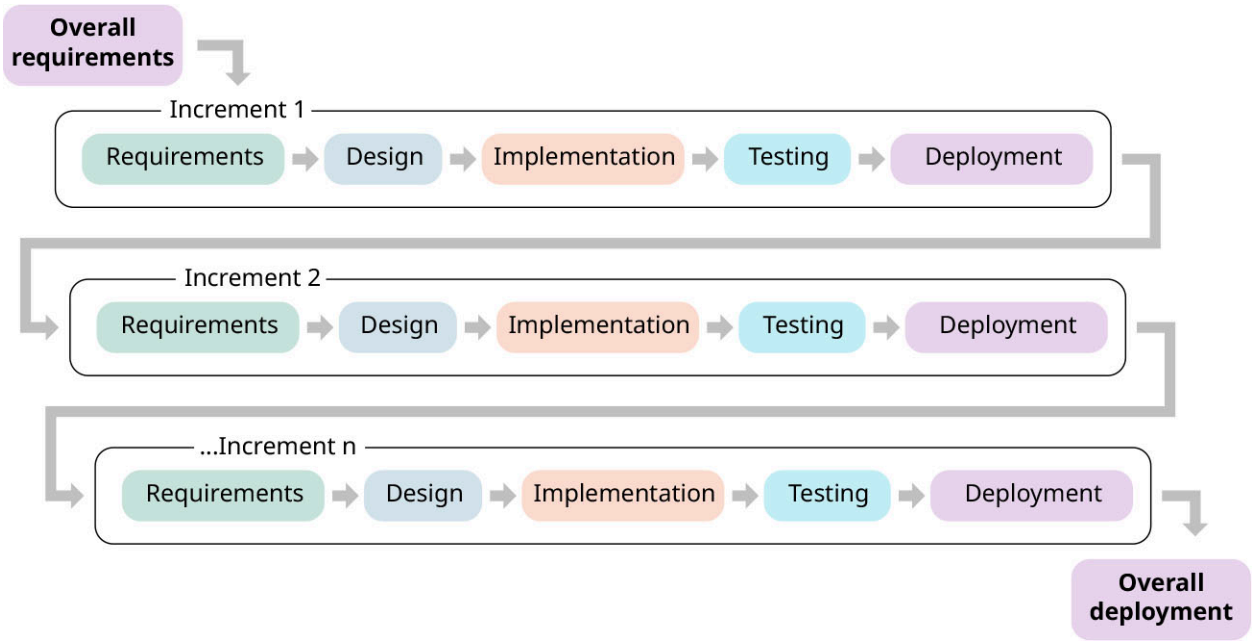


Figure 9.10 In the incremental model, the software process is divided into increments or modules, and each module focuses on a smaller set of requirements based on an overall business plan. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The incremental model is best suited when the requirements are clearly stated at the start of the project and the product needs to be released quickly. Because the increments are small, testing can be done and user feedback can be gained with each increment. This means there are opportunities to identify and fix errors or issues with the product sooner than in the waterfall or V-model methods. Once the overall requirements step is complete, each increment can focus on its specific delivery. This helps to reduce costs, especially if there is a change in the requirements. Additionally, a big advantage of the incremental model is that it is easy to know how much has been completed and what remains to be done.

Prototyping Model

The **prototyping model** requires the quick creation of a mock-up or demo of the expected final product that does what the final product is expected to do in order to be able to show end users what the system could look like and how it might function. Because the users can see what the product may look like and the basics of how it may function, they are in a better position to provide feedback that can be incorporated into the demo and then built into the final product. The prototyping model is also known as a RAD (Rapid Application Development) model because its focus is on getting a working demo created rapidly.

This model still uses gathering requirements, designing, implementing, and testing steps; however, they are all done quickly to build the prototype. The focus is on improving the prototype to get to what is required to build the final software solution.

As shown in Figure 9.11, the prototyping model can be used to build a mock-up that the user can approve, and then the mock-up can be used as an input to the standard process of designing, implementing, testing, and deploying the actual product.

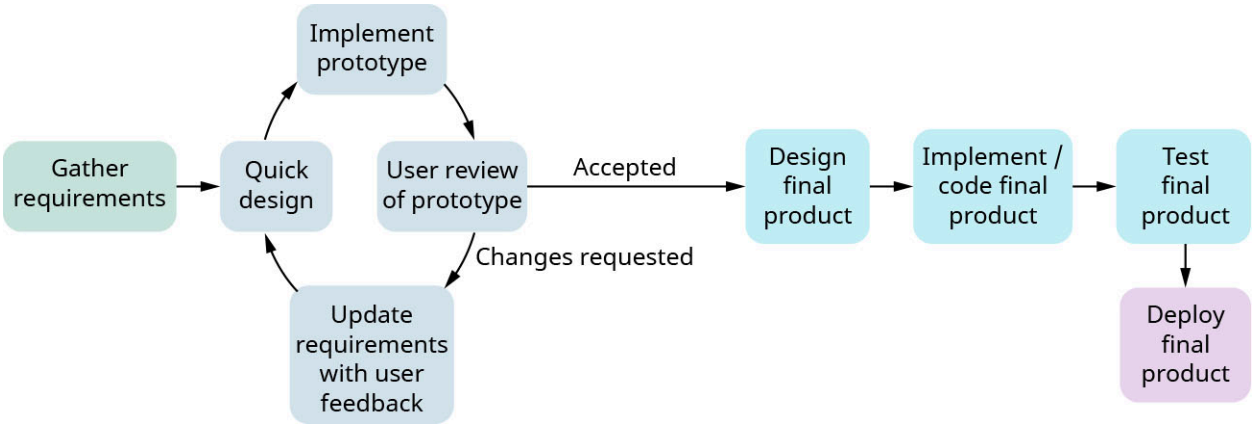


Figure 9.11 A prototyping model focuses on the quick creation of a mock-up or demo of the final product that can be shared with users to gain their feedback. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The prototyping model has some drawbacks. Because it involves getting the user involved early, the process of testing and incorporating feedback can become time consuming. Additionally, while the prototype might appear to work, generally it will lack the full internal functionality, which must still be built even after the user sees what appears to be a working system. This may require added effort to be made to manage expectations about the final product.

Spiral Model

The **spiral model** is a combination of the waterfall model with an iterative model approach and a focus on reducing risk within a project. As with the waterfall model, the spiral model starts with requirements gathering except that it starts with a small set of requirements and then cycles through planning, design, implementation, and testing for those requirements. After the initial set of requirements is addressed, the process iterates back to the beginning, where additional requirements are applied to the project, and then it continues cycling, as illustrated in Figure 9.12, until the software solution is ready for deployment. This model differs from the waterfall method in that it includes a risk analysis as part of the planning step. The risks associated with the project are also assessed during the review and testing of the system.

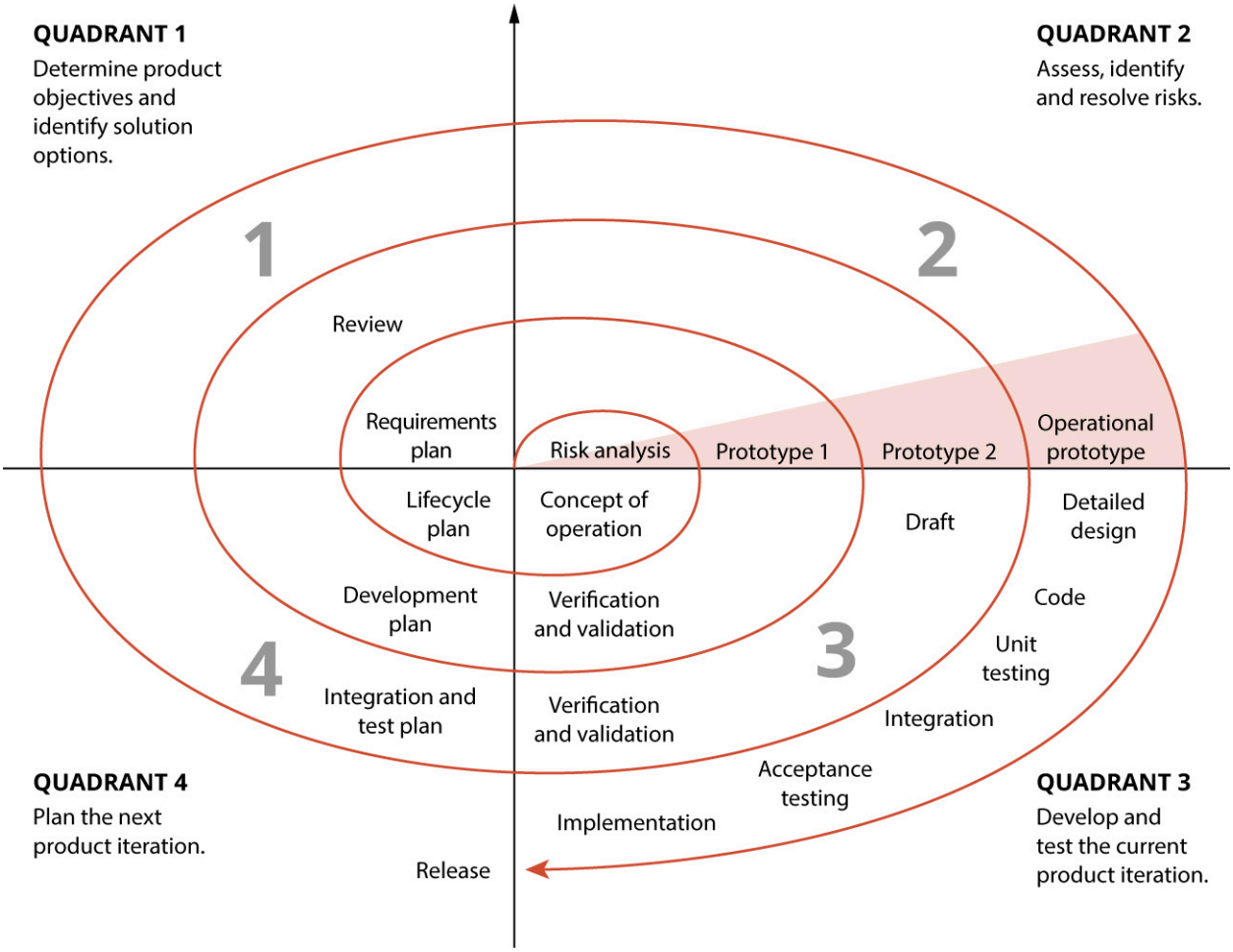


Figure 9.12 The spiral model prioritizes risk analysis by combining elements of the waterfall method with an iterative approach. Each spiral addresses a small set of requirements, but cycles through planning, design, implementation, and testing stages for those requirements. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The spiral model is often used on larger projects or when frequent releases are expected. It is also used when risks are considered high for a project and need to be monitored closely. Such risks can include cost, unclear requirements, complex requirements, or having requirements that could change. The advantage is that because of the spiraling, iterative nature of the model, changes can be added later in the project. Additionally, the model allows for better estimation of costs for individual iterations because a limited number of requirements are being addressed during each spiral. Because each iteration allows for the system to build upon itself, there is also the benefit of being able to adapt to user feedback and changing requirements.

The spiral model does have its disadvantages. Because there is an added focus on risk, it requires expertise in risk management. Additionally, the iterations not only add new features, but can build upon existing features, which can lead to added costs. If applied to smaller projects, the cost can outweigh the benefits of some of the other approaches. Because there are multiple iterations composed of several steps/phases, it is also important to follow the processes more strictly than in the case of other models, and keep good documentation to know what has been done and what is expected.

Unified Process Model

In the **Unified Process (UP) model**, the development of a software system is divided into four primary phases (inception, elaboration, construction, and transition), each of which involves multiple iterations that include the standard software development processes, as shown in Figure 9.13. Like the framework activity/phase in our generic process framework, the UP model differentiates phases from software engineering actions. The UP model still relies on a continuous prescriptive process, but it supports incremental iterative development. The

UP model uses the word *discipline* or *process* to refer to the software engineering action defined in our process framework. The names of the phases are almost the same as the names of the phases in our process framework, although the transition phase is equivalent to deployment in our process framework.

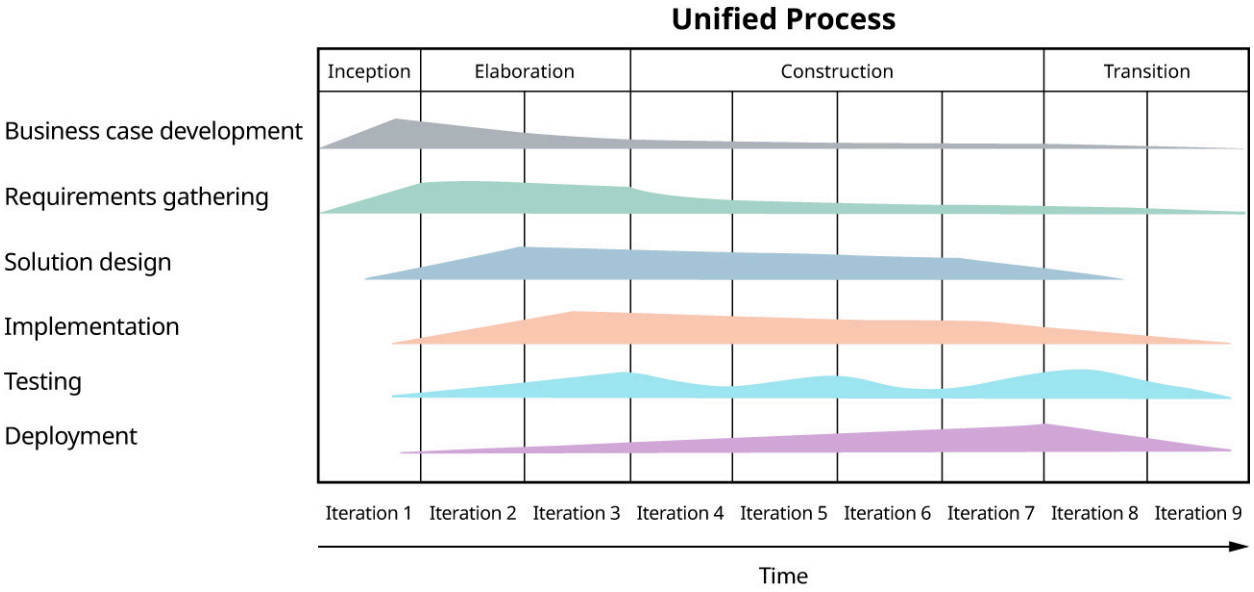


Figure 9.13 The Unified Process model is very similar to the generic process framework as it includes phases in the form of processes (or disciplines). It also features iterations, and the relative size of the colored area for a given discipline indicates how much of the discipline is performed in each iteration. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Each phase or process of the Unified Process model has its own goal. In the inception phase, the goal is to get the users to buy into the solution being presented, document the requirements, and create an initial plan for the project. The elaboration phase focuses on the solution design and firming up the project plan. The construction phase focuses on implementing and testing the software. Finally, the transition phase focuses on the deployment of the solution. As in other traditional process models and in contrast to the Agile models, each phase of the UP model results in a set of deliverables. In the inception phase, this might include documents like a scope statement, initial risk assessment, or preliminary project plan; in the elaboration phase, a software development plan, revised business case, or executable architecture baseline; in the construction phase, individual iteration plans, release description document, or user documentation; and in the transition phase, final product release and lessons learned analysis, among others.

Note that the amount of time spent in each discipline changes over time so that as the project evolves, less time is spent on requirements and design and more time is spent on testing and deploying. The expectation is that the model adjusts the extent of software engineering actions on an ongoing basis depending on what is needed to develop the software solution at a given time.

The expectation when using the UP model is that software will be delivered early and regularly throughout the process. Additionally, the model allows users to see what is coming and provide feedback, which, in turn, allows the software to be adapted to any changing needs. This requires open communication throughout the process and keeping the users as active participants in the project. Another practice with this model is to focus on reusing existing code as well as on using modeling and other tools. Tools such as UML are almost always used as part of UP. Additionally, UP lowers risk through the iterative nature of the development. Because the iterations are timeboxed, there is better control of the overall process. This helps make risk easier to manage.

The disadvantage is that the phases are combined with the disciplines and time and boxed into a set of iterations, which can result in a model that is complex to follow. To be effective, the disciplines need to be managed, and communication needs to be clear. It generally requires good management of the process.

It is worth noting that UP was the first model that distinguished phases from disciplines, which made it

possible to introduce iterations and integrate them with increments. While incremental development had existed before, the separation of phases from disciplines made it newly possible to plan iteration as part of a project increment and thus to create solutions more effectively, as software process actions were no longer associated to a specific framework activity/phase. During each iteration, partial functionality could be created, and the results could be integrated with the rest of the system.

Agile Process Models

Several popular software process models in use today align with the Agile guidelines. While the phases used within Agile process models are similar to that of other models, the difference is that some disciplines in various phases may overlap. For example, in the Agile process model, all the requirements do not need to be defined in the inception phase prior to starting the design or to coding disciplines in the elaboration and construction phases. New requirements or changes in requirements can be considered as part of subsequent phases. Additionally, an incremental, iterative approach is applied, so that it is not necessary for all the functionality in one increment to be dealt with at once. It is worth noting that the incremental model that was presented earlier may be agile for a similar reason. The benefits of Agile process models are they are flexible (i.e., allow for changes along the way) and emphasize collaboration. They also allow for continuous improvement, which is advantageous if requirements are likely to change or needs are likely to evolve. The drawbacks are that Agile models can be difficult to scale.

Agile Principles

Agile principles were formulated in the *Agile Manifesto* that was a response to the unsatisfying number of projects that were delayed, overrun their budget, and did not meet customers' expectations. Agility refers to the ability to create and respond to change in order to profit in a turbulent business environment. Some Agile principles are as follows:

- Satisfy the customer through early and continuous delivery of software.
- Welcome changing requirements, even late in development.
- Deliver working software frequently, such as each couple of weeks.
- The most effective way of communication within a development team is face-to-face (via collaboration tools).
- Working software is the primary measure of progress.
- Maintain a sustainable working pace.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity—the art of maximizing the amount of work not done—is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.

The approaches that complement Agile software development are Scrum, DevOps, and Site Reliability Engineering.

> **LINK TO LEARNING**
>
> To learn more about Agile, you can visit the Agile Alliance to learn the core principles of Agile development from the experts. The Agile Alliance is a global nonprofit organization that is focused on applying and expanding Agile values, principles, and practices. You can view its tutorial *What is Agile?* (https://openstax.org/r/76Agile) online, which provides more details on Agile and its use.

Scrum

Scrum is a type of Agile software development model. The fundamental unit of Scrum is a Scrum team, which is typically ten or fewer people. It consists of one Scrum master, one product owner, and several developers.

The Scrum master is responsible for running Scrum and for helping everyone understand its theory and practice. The product owner is responsible for product backlog management, which includes product goals and product backlog items. The product goal describes future desired features of the product, and the product backlog item defines what is required to be added to the product.

The product is developed in iterations called a **sprint**, which is a fixed-length event typically of one to four weeks. Each sprint starts with sprint planning, in which the team selects the product goals and product backlog items that will be implemented in that sprint. The selected product goals and product backlog items are moved to the sprint backlog, which is a plan for the current sprint. During the sprint, developers have a daily scrum, which is a 15-minute event for the developers that is held every day at the same time and the same place. During the daily scrum, the developers inspect the progress and, if needed, they adapt the objectives of the sprint. At the end of sprint, there is a sprint review, during which the Scrum team and stakeholders review a demonstration of what was achieved as part of the sprint increment, and the Scrum team gets feedback. After that, in the sprint retrospective, the Scrum team discusses what worked well and what worked poorly process-wise during the last sprint to produce the product increment and proposes changes to increase effectiveness. Figure 9.14 depicts the Scrum process.

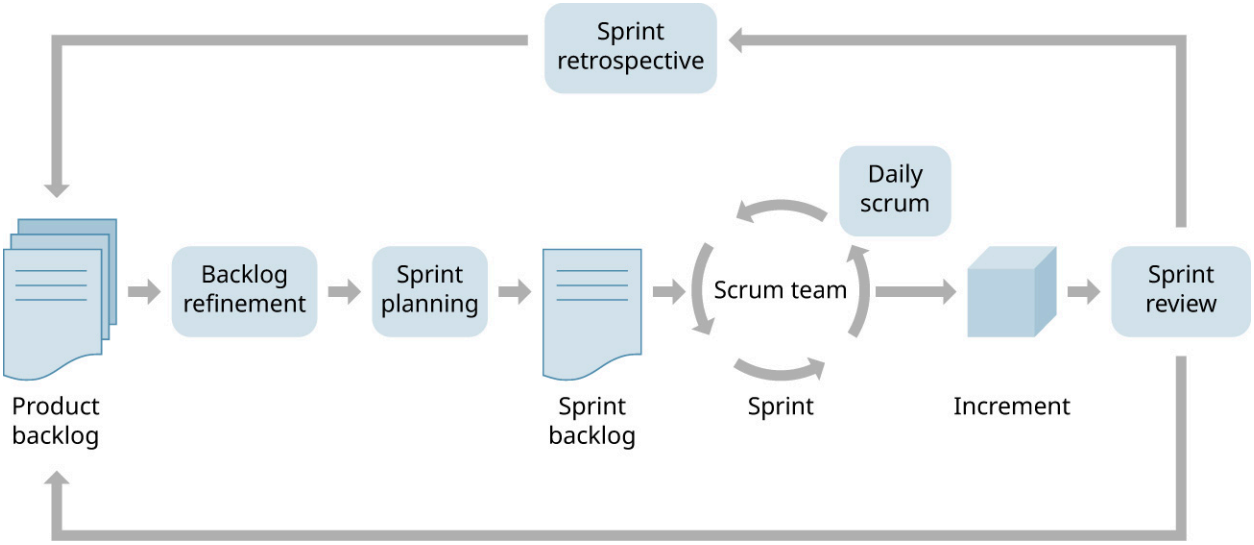


Figure 9.14 The Scrum framework is organized around sprints, workflow events that involve intensive planning, daily collaboration, review, and retrospection. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The benefits of the Scrum framework include better quality of the product, decreased time to market, higher customer satisfaction, and increased collaboration within the development team. The drawbacks are that the approach requires training; it is not suitable for large teams; and it requires daily meetings.

LINK TO LEARNING

Atlassian, a worldwide company that creates team and project related products, has a no-nonsense guide to Agile development that provides additional details on what Agile is as well as on related topics, such as Scrum, Kanban, Agile Project, the Agile Manifesto, and much more. Its site has a tutorial for Scrum as well as related articles (https://openstax.org/r/76ScrumTutorial) where you can learn more about how to use Scrum in a project.

DevOps

A **DevOps model** combines practices of software development and operations. It uses a short development life cycle and continuous delivery to achieve high-quality software products. In a DevOps model, development

and operations teams are merged into a single team, and software engineers participate in all parts of the software life cycle. As shown in Figure 9.15, this life cycle includes planning (design), development, testing, deployment, and operations in a continuous cycle.

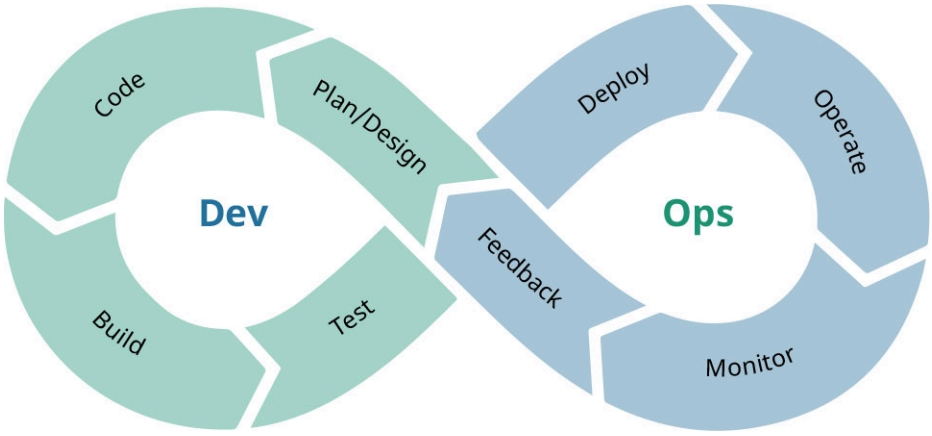


Figure 9.15 DevOps is an Agile software process model that combines the practices of software development and operations such that software engineers participate in all parts of the software life cycle. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Because the DevOps model involves a focus on both development and operations, there is less chance of errors or vulnerabilities existing in a released software product. This uniting of development and operations can also lead to faster releases or shipments of products. Because of the collaboration, there tends to be improved effectiveness, faster delivery, and an optimizing of processes.

Where the DevOps process model can struggle is with systems that are complex as well as with legacy systems. The DevOps model requires strong teamwork and collaboration or it will likely fail. Additionally, it requires that the team members have the right expertise to satisfy the expectations of the project, including the ability to do continuous integration and development.

LINK TO LEARNING

Some of the best practices associated with the fascinating field of DevOps include continuous integration, continuous delivery, microservices, infrastructure as code, monitoring and logging, communication and collaboration, among others. Read this perspective on what DevOps is (https://openstax.org/r/76DevOps) and consider researching other opinions to find other perspectives to help fully understand the concept.

Site Reliability Engineering

Site Reliability Engineering (SRE) is an approach that focuses on achieving appropriate levels of reliability when developing solutions. SRE was created to address the complexity of challenges created when software solutions get larger. It is important to make sure that software meets business needs while operating reliably. The ability to scale up must be balanced with the complexity of a solution while maintaining reliability within a system. In many ways, this is a similar goal to that of the DevOps.

Three key parts of SRE are reliability, appropriateness, and sustainability. A system needs to be reliable to serve the needs of the client. The level of reliability needs to be appropriate. Specifically, some systems don't need 100% reliability 100% of the time. For example, a feature such as cruise control only needs to be reliable when it is in use. Additionally, most cruise control features do not need to maintain an exact speed, but rather can have a difference of a few miles or kilometers when the car is going up or down hills and be okay. Regarding sustainability, a system has to be sustainable and maintainable by people.

As noted earlier, SRE is similar to DevOps in that both bring software engineering and software operations

closer. However, DevOps tends to focus on the product solution, or the "what," whereas SRE focuses more on "how," (i.e., how the solution will get done in a reliable and sustainable manner). Both focus on providing opportunities for collaboration across an organization to deliver solutions that will be successful for the client. Table 9.1 indicates areas where SRE and DevOps differ.

SRE	DevOps
Primary focus of reliability of a solution	Primary focus of effective development and delivery of a solution
Focus on regulating IT with specific measurements such as following service level indicators (SLIs) and service level objectives (SLOs)	Focus on continual integration (CI) and continual development (CD)
Prioritizes user experience by ensuring services run reliably and meet SLOs	Works to use broad ideas and does not specify how operations of services are run
Intended to be a role more than a framework, although it can be performed by those outside of the specific role	Although it can be a role, it is intended to be a philosophy adopted across a team
Works to move quickly to reduce cost of failure	Works to implement gradual change to reduce the chance of failure
Works to have specific expectations of what is acceptable regarding failure or issues with new releases	Accepts failure as a learning opportunity and prioritizes rapid recovery and continual improvement.
Uses automation and monitoring tools to standardize processes and reduce manual effort	Works to reduce organizational isolation by working closer together but does not necessarily go to the same level of using similar tools and techniques

Table 9.1 Comparison of SRE and DevOps

Suggested Process Model

Which process model is the best? There is no best process model, and no process model will work for every project or group of people. Each model has advantages and disadvantages to be considered.

The recommended approach is to use the model that can be tailored best to fit the current project and the skills of the team members. Many organizations have already made this determination and will have their own internal guidance on which model should be used. Regardless, it is often important to consider **software process improvement**, which is the process of transforming the existing approach to software development into something that is more focused, more repeatable, more reliable (in terms of the quality of the product produced and the timeliness of delivery), and more cost-effective. As shown in Figure 9.16, software process improvement typically involves four steps.

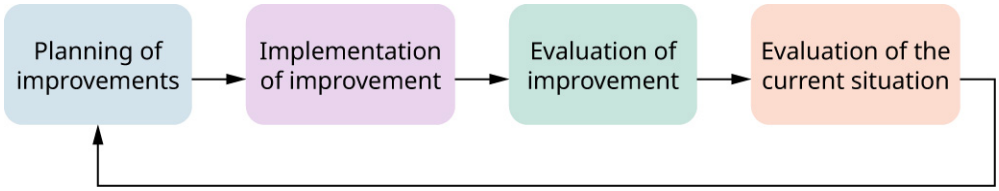


Figure 9.16 Software process improvement can be used to transform any chosen software process model into one that is more focused, repeatable, reliable, and cost-effective. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

INDUSTRY SPOTLIGHT

Applying the Right Process

The proper selection and application of a software engineering process are important in every industry today as these processes help ensure the success of the software solutions being developed. Software engineering offers many industries opportunities for improvement. For example, the New York Stock Exchange (NYSE) has engineered its software to offer stock trading capabilities to anyone anywhere so that they can trade at almost any time. As you can imagine, the NYSE software is complex, and any software issues encountered during trading hours can generate financial losses or cause reputational damages. These are some of the considerations that must have gone into the decision about which software engineering process to use to complete this upgrade.

Of all the processes you've learned about so far, which one(s) do you think might be best for a project that involves developing a software solution to serve the global markets industry? Think about the possible ramifications of software development delays when addressing a software issue on a trading floor. Imagine a software defect causes stock prices to display incorrectly during peak trading hours. How could this impact the market and investors?

9.3 Special Topics

Learning Objectives

By the end of this section, you will be able to:

- Explain the importance of testing in software engineering
- Describe the types of tools used by software engineers
- Describe ways that software reuse is made possible
- Explain the role of ethics in software engineering
- Discuss the future of software engineering

The role of a software engineer is wide-ranging, and the factors that impact software engineers are many. There are a few areas, however, that are worth delving into deeper. These areas include software testing, refactoring, design patterns, software tools used in software development, software reuse, free and open-source software (FOSS), software engineering ethics, and legal aspects.

Testing

The purpose of testing is to verify that the software being developed delivers on the requirements that were set for the project without any unintended errors or side effects. In simple terms, it is to make sure software does what it was expected to do. Through this process of confirming that the software meets expectations, any issues that are found can be resolved to ensure the integrity of the product

Testing should happen within nearly every phase of a project, with each phase having a specific testing focus. One axiom of testing is to test early and test often. As can be seen in Figure 9.17, the earlier issues can be found when developing software, the less costly they are to fix or resolve. Issues are generally found as a

result of testing.

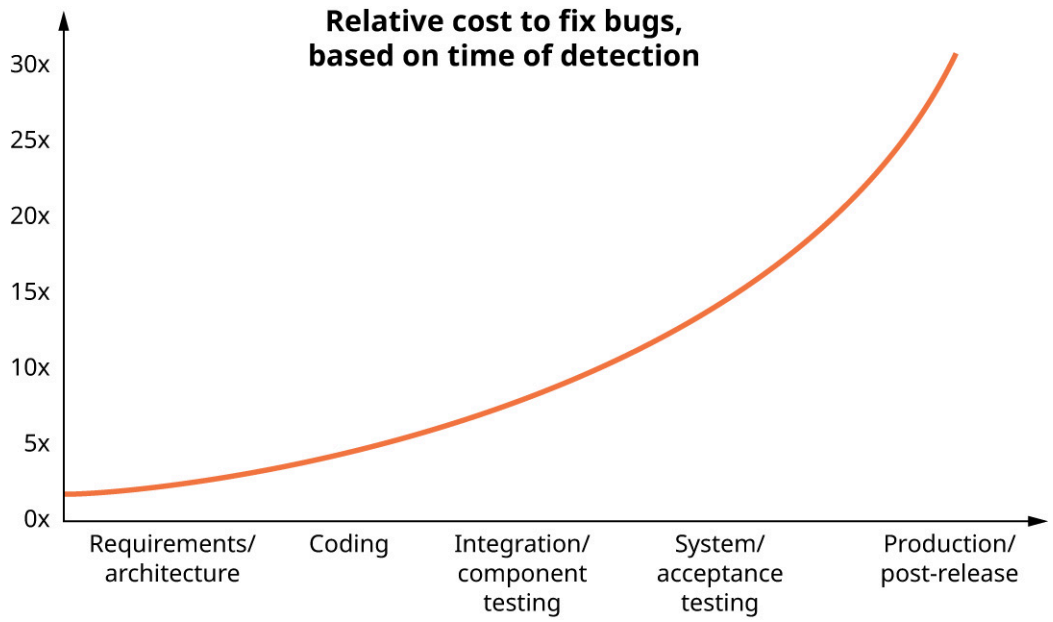


Figure 9.17 The cost of finding issues rises through the life of a software development project; for example, the later a bug is found, the more it costs to fix. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Levels of Testing

Software goes through various levels of testing to ensure quality:

- Unit testing: Individual software components (functions or methods) are tested in isolation to verify they work as intended.
- Integration testing: Focuses on interfaces between components, ensuring they work together seamlessly.
- System testing: Tests the entire software system as a whole, verifying it meets overall requirements.

In addition to the various levels of testing, there are other crucial testing approaches that are used to ensure that software delivers to expectations. These approaches include:

- Acceptance testing: verifies that the software meets the expectations and requirements defined by the client or end users
- Usability testing: evaluates how easy and intuitive the software is for users to interact with
- Stress testing: assesses how the software performs under heavy loads or unfavorable conditions
- Performance testing: evaluates the speed, responsiveness, and scalability of the software under various conditions
- Security testing: identifies and addresses security vulnerabilities to protect against unauthorized access or data breaches

Some testing is done by software developers, including software engineers who write code, but a majority of testing is done by quality engineers whose primary focus is testing. End users can also participate in acceptance testing or user testing scenarios.

CONCEPTS IN PRACTICE

How Does Microsoft Test Windows Before Release?

Microsoft uses millions of testers before deploying a new update of Windows. These testers are a part of the Windows Insider program and have access to the prerelease versions of Windows. Anyone can sign up

for this program; they just need to agree to test the new version and provide Microsoft with feedback. Because testing is done on a prerelease version of the software, the testers know that the software may be unstable and will likely have issues. If you are curious about testing—or are interested in becoming a tester yourself—visit Microsoft Windows Insider (https://openstax.org/r/76WdwsInsider) or Apple's Beta Software Program (https://openstax.org/r/76BetaSoftware) for more information.

Methodologies of Testing

There are three primary testing methodologies, and they are categorized by the knowledge the tester has (or needs to have) to conduct to the tests. The three methodologies are:

- The manner of testing where the tester is aware of the code, so they can test that the internal structure of the item being tested works properly, is called **white box testing**. Other names for white box testing include glass box, clear box, and structural testing.
- Tests based on requirements and the functionality of what is being tested without the need to focus on the code itself is called **black box testing**. The tester does not need to have any knowledge of the code. Other names for black box testing include input-output testing, specification-based testing, and behavioral testing.
- The hybrid of both white box and black box testing is called **gray box testing**. The person who designs the test has a partial knowledge of code structure and understands the intended design of the software.

When testing at the testing system level, we distinguish between verification and validation. Testing that the software solution functions without errors is called **verification**. In organizations that have large development teams, verification is often done by a Quality Assurance (QA) team. Testing that the software solution conforms to the requirements and does what the user wants it to do is called **validation**. Both verification and validation require code execution and can happen in all levels of testing, with verification generally taking place before validation.

Testing is also distinguished by purpose into functional and nonfunctional testing. With functional testing, the functionality is tested. With nonfunctional testing, qualities such as performance, scalability, and usability are tested.

The attributes of a good test are as follows:

- A good test has a high probability of finding an error or issue.
- A good test is not redundant. (It does not test the same thing as another test.)
- A good test should be neither too simple nor too complex.

Unit Testing

A crucial role in software development, the process of **unit testing** involves testing individual units of code, such as methods and functions, and it is usually done by developers during the development of the software or when updates are made. Software developers write scripts or code that test the functionality of a specific piece of code. Unit tests are typically added to a regression test suite, so they can be run again after each change to the source code to verify that the change did not break any existing functionality. Imagine a function that calculates the area of a rectangle. A unit test would verify that the function returns the correct area for different width and height values.

Unit testing offers several benefits:

- Early bug detection: Unit tests help identify bugs early in the development process, when they are easier and less expensive to fix.
- Improved code quality: The process of writing unit tests encourages developers to write clean, modular, and well-documented code.

- Maintainability: Unit tests serve as living documentation, clarifying the intended behavior of code components and making future modifications easier.

CONCEPTS IN PRACTICE

Staying in Their Lane

In the automobile industry, most new cars now include a lane detection system (LDS). This system includes code that verifies video images that are received by the automobile. The software checks to see if the vehicle crosses one of the lines painted on the road. If so, it activates a warning system to let the driver know. Units test for such a system include the following:

- An appropriate line is recognized.
- A warning occurs if a line appears to be crossed.
- The warning ends after the designated period of time.
- When the LDS system is turned off, the warning is not triggered.

Software engineers as well as testers perform these tests when the code for the LDS system is created as well as any time any updates are made to the system.

Code Coverage

Proper unit testing improves the quality of a software system and helps discover bugs early. In simple terms, a software **bug** is an issue or error with software programming. To find bugs, it is important to review and run the code.

The range of a unit test is typically measured in terms of **code coverage**, which is the measurement of the percentage of code that is activated or reviewed by the test. Common types of code coverage are statement coverage, line coverage, and path coverage. Measurement of the percentage of statements that are activated at least once when you run all tests is **statement coverage**. Measurement of the percentage of activated lines of source code that are tested is **line coverage**. Measurement of the percentage of paths through source code that you go through is **path coverage**. As shown in Figure 9.18, with path coverage, if you have two conditional statements, one after the other, there are typically four paths through this code:

1. Both conditions are true.
2. The first condition is true and the second is false.
3. The first condition is false and the second is true.
4. Both conditions are false.

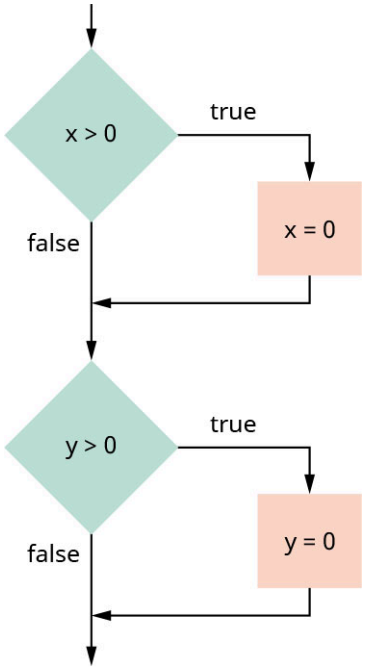


Figure 9.18 This activity diagram illustrates how path coverage works with four possible paths: (i) *x* and *y* positive, (ii) *x* positive, *y* negative or zero, (iii) *x* negative or zero, *y* positive, and (iv) *x* and *y* negative or zero. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Path coverage is more demanding and typically requires more tests than statement or line coverage, but tests are better. You need a minimum of four tests to achieve 100% path coverage, but a minimum of two tests to achieve the same level of statement coverage.

Support for Unit Testing

Unit testing is so common that major programming languages provide some support for automated unit testing. An example is JUnit, a unit testing framework for the Java programming language, which is shown in Figure 9.19. JUnit facilitates the writing and managing of tests, and it can be run in some development environments quickly, typically via a single click. Many of the other major programming languages have similar frameworks or tools. There are also third-party software products that can be used to help with unit testing.

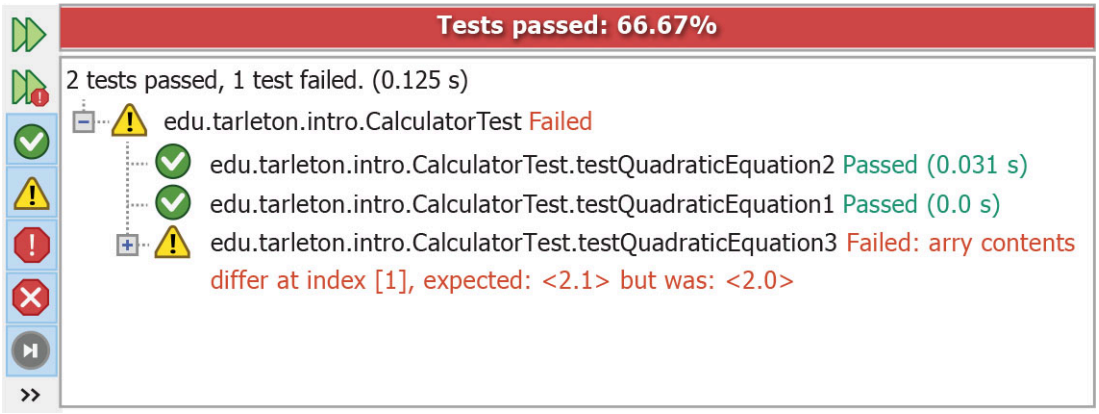


Figure 9.19 A sample JUnit test window in NetBeans integrated development environment (IDE) is shown. You can run all tests by clicking on the green arrows. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: NetBeans by Sun Microsystems)

Test-Driven Development

Many organizations use **test-driven development (TDD)**, which is a process where developers write tests before they write the code. This might seem counterintuitive, but it offers several benefits:

- Clarifies requirements: Writing tests first forces developers to think critically about the problem they are trying to solve and the expected behavior of the code. For example, if you are to write code that finds real roots of a quadratic equation and you start with unit tests, you will probably identify three cases that must be solved separately: the case when the equation has two roots, the case when the equation has a single root, and the case when the equation has no root.
- Facilitates early bug detection: Writing tests first helps identify bugs early in the development process, when they are easier and less expensive to fix.
- Results in improved design: The test-driven approach can help identify edge cases and scenarios that might be overlooked during traditional development, leading to a more robust design.
- Results in improved code quality: The focus on writing clear, testable code leads to overall higher code quality.

As illustrated in Figure 9.20, in test-driven development, you write unit tests first. You then run the tests, which should fail because there is not a fully implemented function to test against. You will then run (and rerun) the tests as you develop the functions. Development continues until all of the unit tests pass.

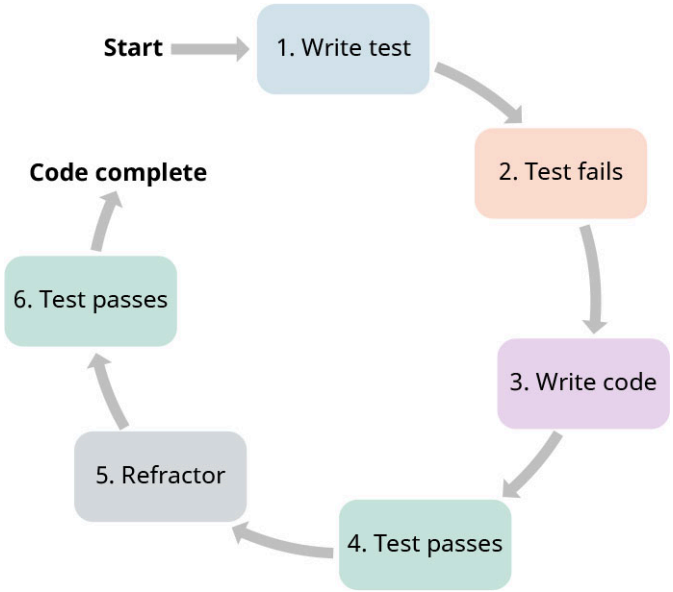


Figure 9.20 In test-driven development, the developer writes the test before writing the code. Thus, when the test is first run, it should fail. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Here's a breakdown of the steps of the test-driven development process:

1. Write a test: The process starts with the developer or development team writing a unit test that defines the expected behavior of a specific function or code unit.
2. Run the test (and see it fail): Run the test. Initially, the test will likely fail because the function it tests doesn't exist yet or isn't implemented completely.
3. Write just enough code to make the test pass: Focus on writing the minimum amount of code necessary to make it pass the test.
4. Run the test (and see it pass): When the code passes the test, this ensures it is fulfilling the intended functionality defined in the test.
5. Refactor (improve code structure): Once the test passes, take a step back and refactor the code to improve readability, maintainability, and overall code structure without changing functionality. The process of restructuring source code without changing its functionality is called **refactoring**. We typically use refactoring to make code more readable. This can be as simple as renaming a variable or as complex as breaking the function up and splitting it across modules. Once the refactoring is complete, the unit tests should be executed again to make sure the code continues to pass.
6. Run tests again: After refactoring, run all tests again to verify the code continues to pass and ensure no

unintended side effects were introduced during refactoring.

Once you're done with all the steps, start the cycle over: Repeat this cycle (write test, run test, write code, refactor, run test) for each unit of functionality you develop.

System Testing

Whereas unit testing focuses on the various units or pieces within a software solution, **system testing** focuses on the complete and fully integrated software product. The overall goal of system testing is to make sure the complete software solution works on the whole as expected. This is generally a black box style of testing that focuses on making sure the software meets all the requirements that were determined and checks that any possible scenarios that could be applied to the software function play out as expected. System testing is generally done by testers rather than the end users of software, and it involves reviewing both functional and nonfunctional requirements.

System tests can be performed by testers, or they can be automated. The form of system testing where a person must run the software system, provide any input, and manually check all output is called **manual testing**. Repeating tests requires these efforts to be repeated by the tester. The form of testing where a program or code is executed that tests the functionality of a software system or some of its parts is called **automated testing**. It allows for repeating tests without requiring repeated effort on the part of the tester. An example of such a program is Selenium. Selenium is a suite of free tools that automates the testing of web applications. The easiest way to use Selenium is to have it capture the interaction between your browser and web application. The interaction is saved as a script. You can then edit the script, if needed, and replay it to check if the web application still works. Selenium also provides libraries for many popular programming languages, including Java, C#, and Python, that allow users to write programs that perform this testing.

Acceptance Testing

Also called user acceptance testing, **acceptance testing** is the process used to determine whether the software solution fulfills the customer's expectations. This level of testing generally occurs after system testing and is done by the customer, client, or other end user of the software.

Acceptance tests are formalized testing that can be specified by the customer, and it tests both functional and nonfunctional requirements. To test functional requirements, acceptance tests often follow scenarios created in requirements analysis. For example, in a library information system, we can use scenario "Add new book" to create an acceptance test. During the acceptance test, the tester will fill out an input form, submit it, and check that the system contains a new book.

In the automotive industry, acceptance testing could be done by the department that requested the new features. It could also be done by dealerships or customers who own or have purchased previous versions of a given automobile. A driver could be asked to use the new features and then be surveyed to see what issues occurred.

Usability Testing

The user-centered testing method that evaluates how easy and intuitive a software system is to use is called **usability testing**. It goes beyond just ensuring the software functions correctly and focuses on the **user interface/user experience (UI/UX)** the part of computer programming that concerns how information is presented to the user and how the user can interact with a program. Usability testing is essential because software that is difficult to use can frustrate users and hinder adoption. By identifying usability issues early in the development process, developers can make improvements that lead to a more user-friendly and successful product. Usability testing typically involves recruiting a small group of users representative of the target audience. These users are asked to complete a series of tasks while a usability tester observes their actions and records their feedback. The tester pays close attention to areas such as:

- Task completion: Can users complete the intended tasks efficiently and without errors?

- Ease of learning: How easy is it for users to learn how to use the software?
- User satisfaction: Are users satisfied with the overall user experience?

While acceptance testing generally happens near the end of a project, usability testing can and should happen much earlier. The focus of usability testing is to enhance user experience, and it typically emphasizes the following areas:

- Interactivity: Are interaction mechanisms (e.g., menus, buttons) clear and consistent? Do they provide appropriate feedback to user actions?
- Layout: Is information organized in a logical and easy-to-find way? Can users navigate the system intuitively?
- Readability: Is text easy to read and understand? Are error messages clear and actionable?
- Aesthetics: Is the overall look and feel of the software pleasing and consistent? Do colors, fonts, and images contribute to usability?
- Display usage: Does the software solution use the best possible screen size and display resolution?
- Timing: Are important features and other pieces of functionality quickly accessible?
- Feedback support: Is useful feedback provided to end users, and can they easily go back to their work once the feedback has been read?
- Personalization: Is the application addressing various categories of users and does it support inclusion?
- Help: Is it easy for users to access help and other support options?
- Accessibility: Can the software be used by people with disabilities? This includes features such as screen readers and keyboard navigation. Note that **accessibility testing** is a subset of usability testing that focuses specifically on the needs of users with disabilities. Here are some examples of accessibility considerations:
  - Screen reader compatibility: Can screen readers interpret the content and functionality of the software accurately?
  - Color contrast: Does the software use sufficient color contrast to be usable by people with visual impairments?
  - Keyboard navigation: Can all features of the software be accessed using only the keyboard?
  - Content features: Is blinking, scrolling, or auto content updating avoided to accommodate users with reading difficulties?

INDUSTRY SPOTLIGHT

Usability and Acceptance Testing by Car Manufacturers

Usability and acceptance testing are quite different. Let's consider again the software for the lane departure systems (LDS) that are incorporated into automobiles these days. Acceptance testing would confirm that the lane departure system works as expected: Did the warning happen when a vehicle crossed a lane line? Did the software recognize when the vehicle departed the lane? Did the system warning start and end in an amount of time that was appropriate to the user? When the system is disabled, did the software ignore lane departures and not trigger the warnings?

With usability testing, the focus is more on whether the system met the expectations of the users interacting with it: Was the system usable? Did the user use the feature as intended? Did the user react appropriately to the warning signal? Was the signal so startling it caused the user to jerk and risk an accident? Did the user simply turn off the system because they found it annoying? The focus of usability testing is on how the solution is being used and whether it was created in the most intuitive manner

The Insurance Institute for Highway Safety found that 49% of users turned the LDS off during their normal commuting. More important, it was found that 54% turned it on if it used tactile warnings compared to only 46% turning it on if it used an audible alert. From a usability testing standpoint, this shows that it is better

to focus on a tactile warning over an audible one if the goal is to get the most usage. Because LDS has been shown to decrease the chance of fatal accidents, even with low usage, it can still help save lives.

Software Engineering Tools

Software developers use many tools that facilitate development; some of them are used daily while others only occasionally, but they all play an important role in software engineering process.

Compiler

A compiler is a program that converts source code into a syntax or format that a computer can execute. Programming languages such as C and C++ compile into machine code, which consists of instructions for a specific processor. Other programming languages such as Java compile into machine independent code. For example, Java program code is generally compiled into Java bytecode. This machine independent code is then executed on a Java Virtual Machine.

Debugger

A **debugger** is a program that assists in detecting and correcting of bugs. It typically helps in finding issues in code by enabling features such as the ability to halt a program while it's running so that variable values can be displayed and even changed. Debuggers can also run a program line by line to see what each line of code is doing. More advanced features involve halting a program on a specified condition and checking for deadlock and memory leaks. Figure 9.21 shows the NetBeans debugger stopped on line 6 of a program. In the lower part of the screen, the software developer can see the values of the variables currently used by the program.

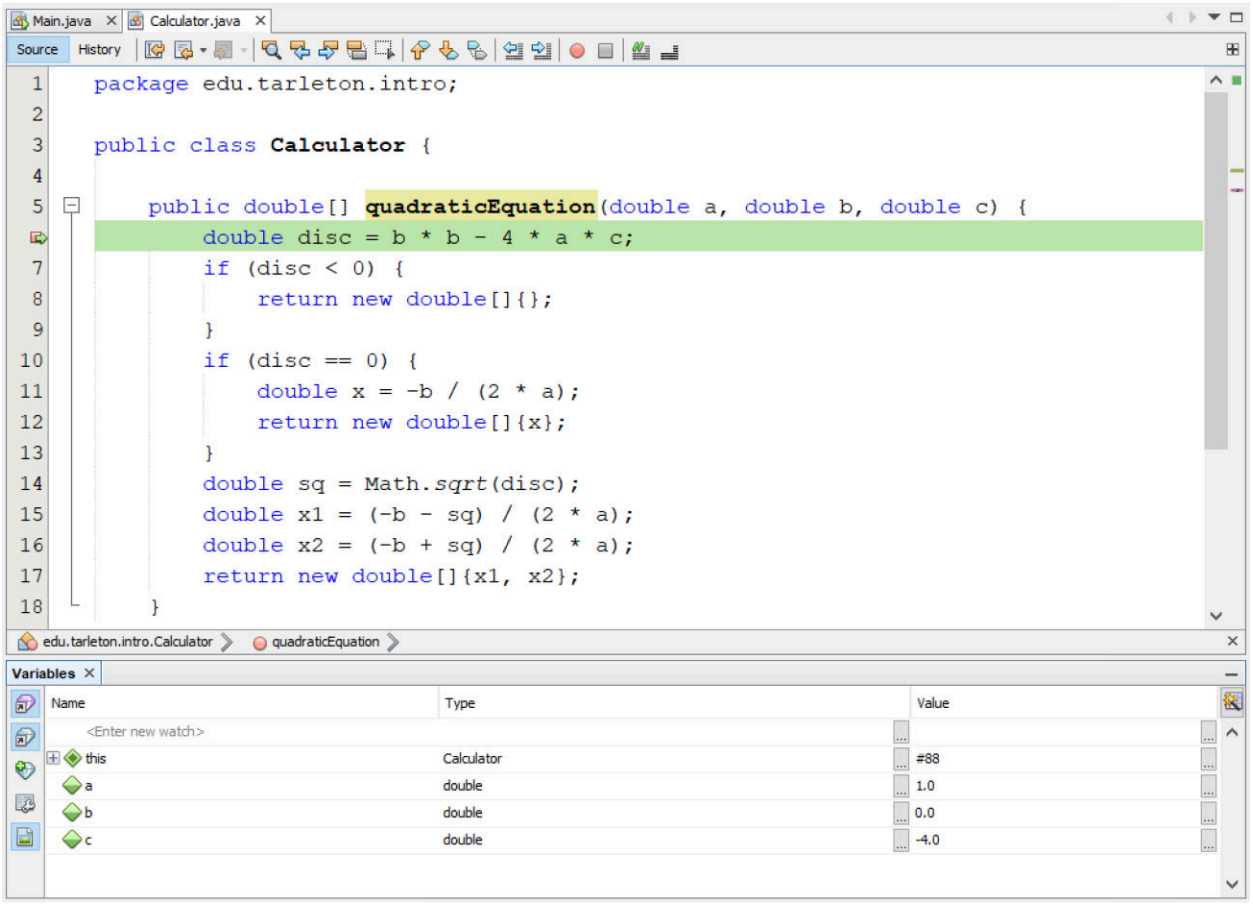


Figure 9.21 In this debugger tool from NetBeans, the program stopped at line 6, creating an opportunity for the developer to choose what to do next. The lower window displays the values of variables currently used by the program. (rendered in NetBeans by Sun

Microsystems; attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Profiler

A **profiler** is a program that performs dynamic program analysis that can be used to optimize or otherwise streamline code. It collects data such as frequency and duration of method calls and presents it to the software developer who can use it to propose code optimizations. The process of data gathering is called profiling, and it is typically performed when the performance of the software system does not meet specified criteria.

Integrated Development Environment (IDE)

Software engineers typically write source code in an IDE such as IntelliJ, Microsoft Visual Studio, or VS Code. An IDE is a software application that facilitates software development by combining several tools that developers use such as an editor, compiler, debugger, and profiler. Some IDEs only support a specific programming language, such as IDLE for Python, and some support multiple programming languages, such as Microsoft Visual Studio, which supports C++, C#, Visual Basic, and J#.

The best IDEs provide features that facilitate writing and maintaining source code. These features can span from simple features such as syntax highlighting to sophisticated ones such as static analysis of source code. Syntax highlighting displays different parts of the source code in different colors, which facilitates reading. Figure 9.22 shows an IDE highlighting Java code.

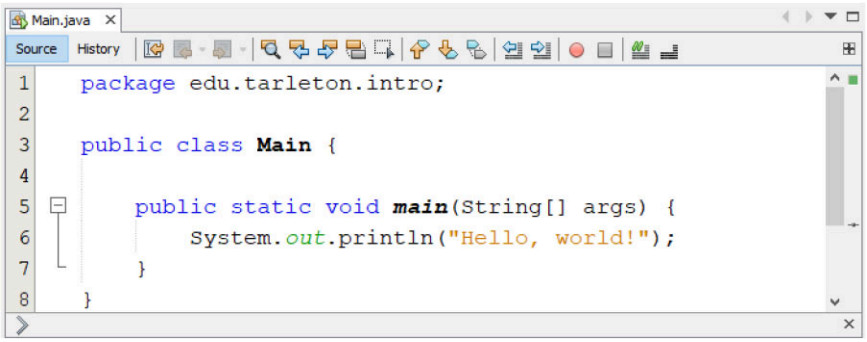


Figure 9.22 Syntax highlighting is a feature of IDEs that helps developers read code. Here, Java source code is being highlighted by the NetBeans IDE. (rendered in NetBeans by Sun Microsystems; attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The cost of IDEs can range from free to thousands of dollars. IDEs such as Eclipse, NetBeans, and Microsoft Visual Studio Code are free. Microsoft Visual Studio and IntelliJ IDEA have a fee associated, although there are free editions of each available. Generally, the cost of an IDE is offset by the increase in productivity that a developer gains from its use.

Version Control System

A quite common tool used in software development is a **version control system**, which stores the history of changes to source code and facilitates collaboration of multiple developers. The most popular version control system today is Git, which is often used with a common hub, such as GitHub or Bitbucket.

Git stores the project source code written by a developer in a repository that is typically on the developer's machine. Developers can push their code to a project repository for integration purposes. The main features of Git are as follows:

- It tracks changes in your source code.
- It enables reverting the changes.
- It facilitates collaborative development.

GitHub facilitates code sharing among developers by allowing developers to modify source code on their own machine, commit changes to a Git repository on their machine, and then push changes from their machine to

GitHub so that other developers can access them.

Many open-source projects are hosted on GitHub that allow people to download their source code and contribute. Figure 9.23 shows a standard GitHub page, in this case from the JavaScript library called the jQuery project.

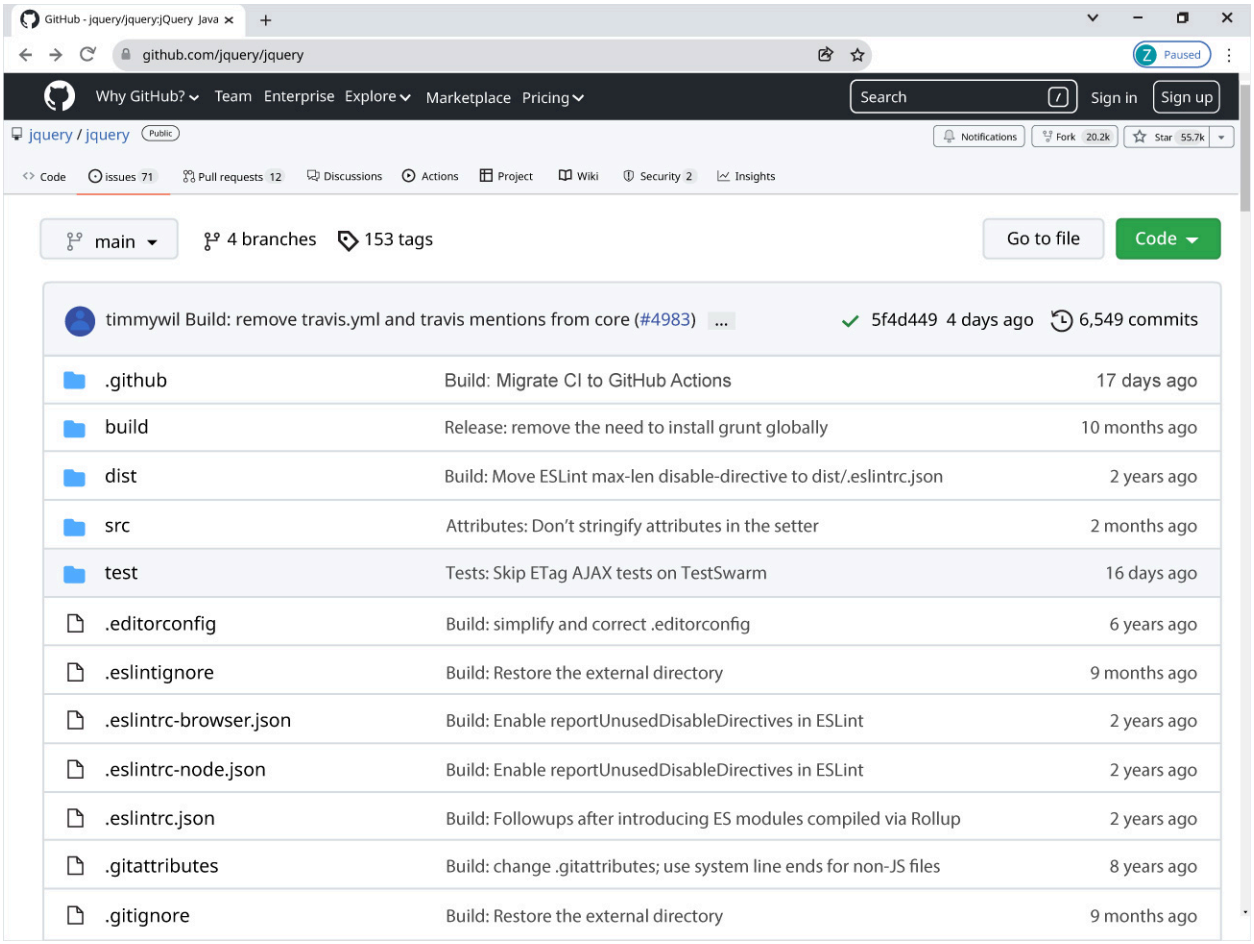


Figure 9.23 GitHub is a version control system that facilitates code sharing among developers. (credit: GitHub, CC0 1.0)

Bug Tracking System

Another common software development tool is a **bug tracking system**, which aids in the tracking and resolution of fixing issues with software. Bug tracking software stores information about issues that have been reported and their resolution.

A common workflow with the bug tracking system is as follows: when a customer or a tester reports a bug, a new record in the bug tracking system is created. Then, the project manager or any other team member assigns the bug to a developer. The bug report must provide instructions as to how the bug manifests and may be fixed so that the assigned developer can reproduce it and decide how to proceed. Here's a general breakdown of how bug tracking works:

- If the bug cannot be reproduced or if it is not a bug but a request for a new feature, it will be rejected.
- If the report describes a bug that was already reported, it will be marked as a duplicate.
- If the fix is scheduled for a next release, it will be marked as deferred.

When the developer fixes the bug, the code should be retested to confirm that the bug has truly been resolved. Figure 9.24 illustrates a common bug reporting and tracking flow.

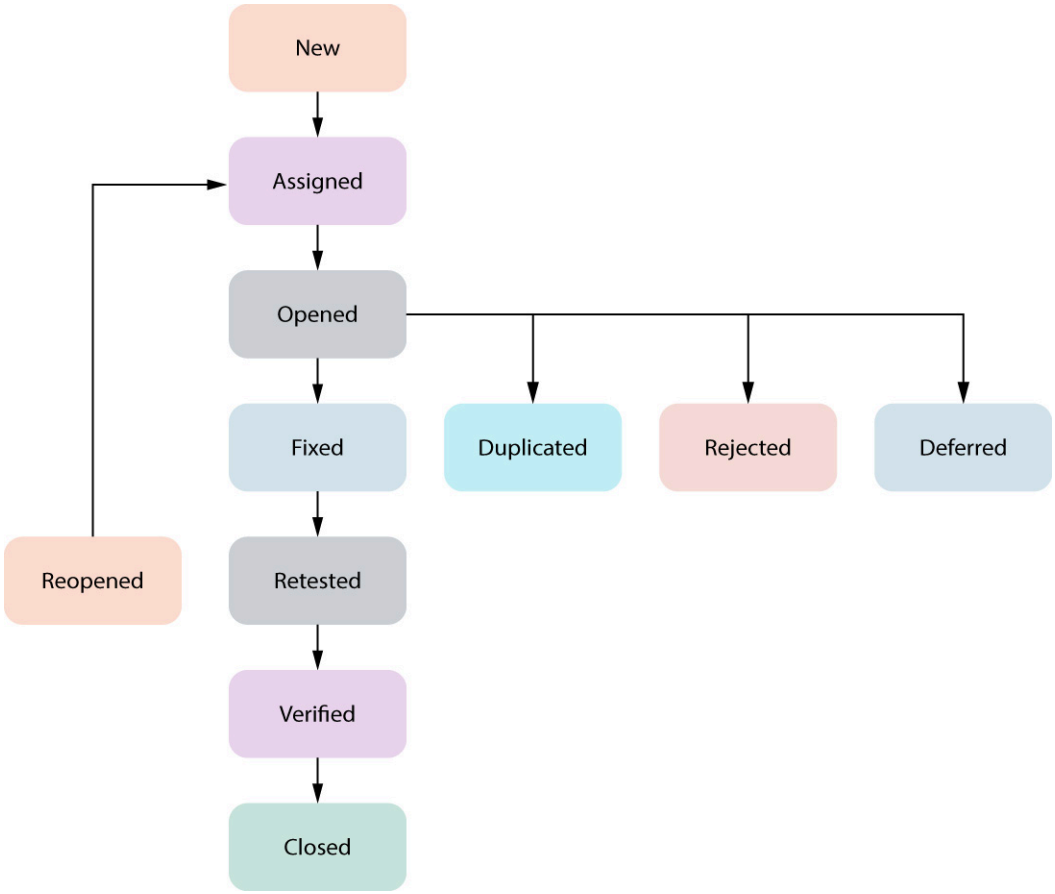


Figure 9.24 The Jira issue tracking system shows a common workflow of bug tracking systems. When a bug is detected, a new record is created. After the bug has gone through the process of being assigned and fixed, and the fix has been verified, the record closes. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Figure 9.25 shows a bug tracking tool—the jQuery bug tracking tool—that is available on GitHub.

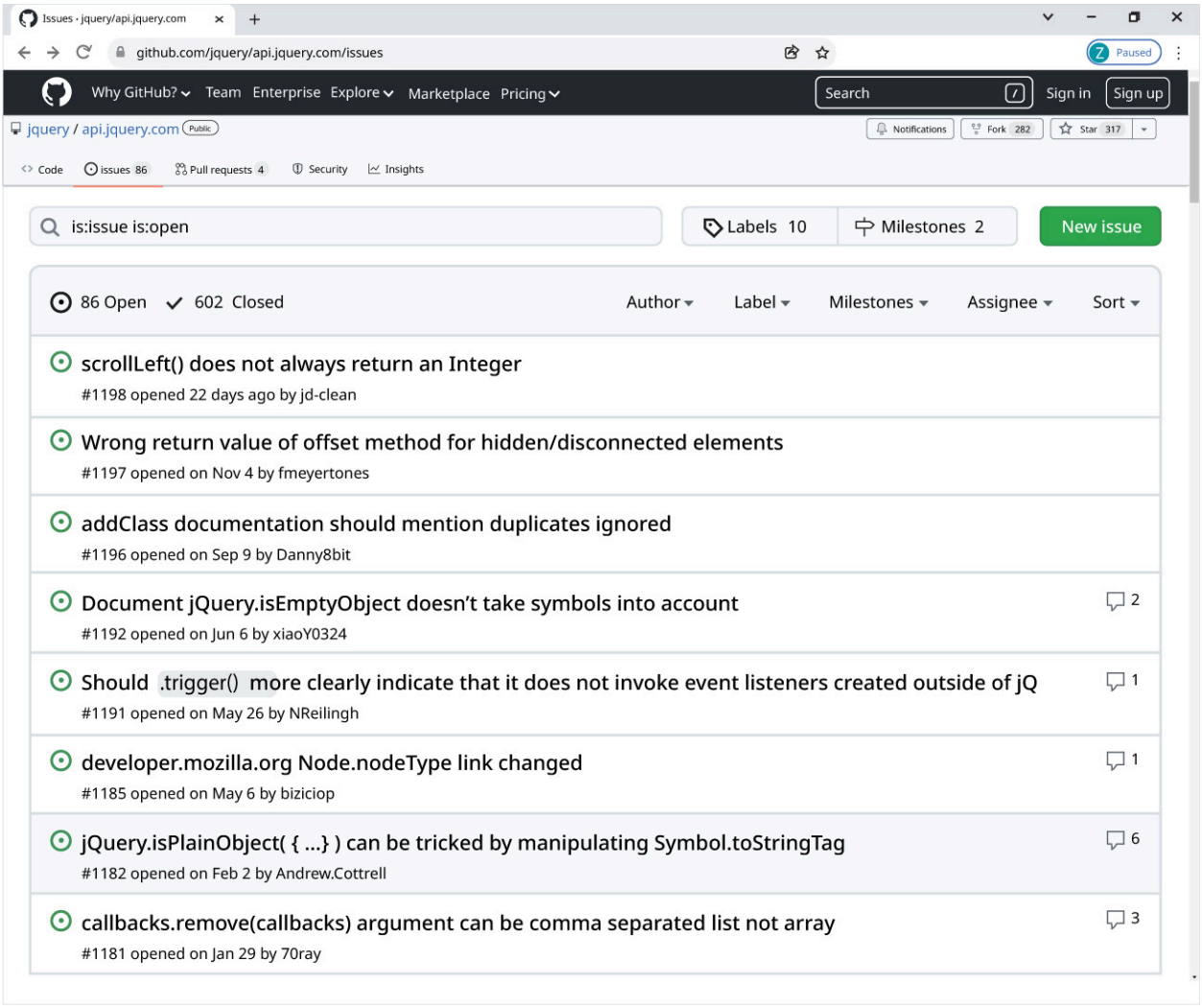


Figure 9.25 The jQuery bug tracker on GitHub keeps track of how many issues are pending ("86 Open") and which have been resolved ("602 Closed"). (credit: GitHub, CC0 1.0)

INDUSTRY SPOTLIGHT

Bugs Not Caught

In January of 2022, the San Francisco Unified School District rolled out a new employee information system, which included the payroll for over 10,000 employees. Unfortunately, the development team did not do a thorough job of testing this system, and it contained a number of bugs when it was launched. As a result, hundreds of teachers and staff members did not, even after several months, receive paychecks from the system.

Because the system was unable to process payroll, hundreds of employees were unable to pay their rent, medical bills, and other expenses. In this case, the delivery of a flawed system harmed people.

This is an example where a system's product did not adhere to the highest standards and failed to be tested to a level that would avoid an issue that directly harmed many of the employees who relied on the system. While the Chief Technology Officer issued apologies and assigned 100% of the project staff to get the issues resolved, the reality is—and ethically speaking—more time should have been allotted to test the system prior to it replacing the old system.

We typically think about all of the ways that technology and systems can help people. It is also important to acknowledge there are risks also. Bugs in software can have a significant impact on the people affected. This reinforces the importance of testing and validating a system before we release it.

Software Reuse

When constructing software, developers commonly use libraries and frameworks, which consist of code that was written by other developers. Libraries typically provide some functions, which you can call in your code, and frameworks typically provide application skeletons, which can call your code. For example, you can have a library that provides encryption functionality and a framework that facilitates the development of web applications. Both libraries and frameworks are common in software development, and it is common to combine them. Libraries can be static, which are linked to your code at compile time, or dynamic, which are linked to your code when it executes. Because static libraries are linked at compile time, you must recompile the program when the library is modified. On the other hand, a dynamic library is in a separate file, and thus it can evolve independently of your code.

Some programming languages, such as Java, provide ready-to-use code as part of a language library. That code may be invoked via an application programming interface (API) and it involves implementation of common data structures and operations on them. Software applications may also leverage external APIs that allow developers to access functionality or data. For example, the United States Postal Service (USPS) provides an API that can be used to get shipping rates, track packages, schedule package pickups, and more. There is no need for a developer to write new code to do this, but rather they can tap into the USPS API instead.

It might seem that we do not need to write code anymore because we can assemble applications from existing components, but, unfortunately, it does not work this way. One of the reasons is that we usually want to customize functionality, and another reason is that even if an existing component is a good fit for a project, we are often not allowed to use it due to legal reasons. Each component is published under legal terms called licenses, and if we want to use it, we must follow these terms. For example, components published under the GNU General Public License require software that uses them to be published under the same license, which is usually not acceptable in commercial environments.

TECHNOLOGY IN EVERYDAY LIFE

Libraries on Your Computer

Code reuse and the use of libraries is extremely common. In fact, if you look at the files on a Microsoft Windows computer, you will find that a plethora of libraries are used by the Windows operating system itself. The files that have an extension of .DLL are library files where DLL stands for dynamic-link library.

On a computer running Microsoft Windows, do a search in Windows File Explorer using "*.dll" to view a list of dynamic-link libraries. Notice how many there are and the dates they were updated. Many of the Windows operating system files are DLL files that get updated at a different frequency as compared to the operating system. The names of some of the files will give you an idea of what the code within them likely does. You can, of course, attempt a similar search for a computer running on Apple's operating system.

Patterns

A **pattern** is a high-level concept that supports the idea of reuse and provides reusable solutions to problems often encountered when building software. Patterns are not backed up by any theory; they relate to solutions that were observed in practice at various levels of abstraction and proved successful. In general, patterns are discussed in terms of their meaning, their intent, and the benefits of using them across many software

engineering areas. Patterns are typically organized hierarchically in pattern catalogs using descriptive pattern templates. Pattern languages may be used to describe the compatibility between various patterns and help weave them together whenever applicable.

Software engineers are likely to encounter two broad categories of patterns depending on whether they are designing a software architecture model or mapping it to a corresponding implementation architecture tailored to the platform or environment the solution is meant to be deployed onto. Patterns used to model software architectures include architectural style, architectural patterns, and design patterns, an architectural model, or an implementation of such.

As you learned earlier in the chapter, an architectural style is a transformation that is imposed on the design of an entire system. The intent is to establish a structure for all components of a system under development. Examples of architectural styles for distributed systems include the object management architecture (OMA), service-oriented architecture (SOA), multitier architecture, and peer-to-peer decentralized architecture. Note that multiple architectural styles may be combined to create a hybrid architecture style. An architectural pattern is part of a category of patterns that focus on the architecture of software. They tend to be more abstract and focused on improving issues such as deployment, availability, maintainability, performance, scalability, security, and testing. A **design pattern** is a reusable solution to a design problem that software engineers repeatedly encounter while architecting and designing software systems. For example, the design pattern Singleton suggests how to restrict the number of class instances to one. This problem commonly occurs in software systems whenever a class represents a thing or concept that has a single instance. A more complex design pattern is Builder, which allows a software engineer to create an object in steps and is typically used when creation is a complex process.

Furthermore, a design pattern imposes a transformation on the design of an architecture of a given architectural pattern. Therefore, architectural and design patterns may be used in conjunction with an architectural style to shape the overall structure of a system. For example, Model-View-Controller (MVC) is a typical architectural pattern that is used to design the software architecture of a multitier system. If the multitier system provides a scoreboard, that feature will leverage a Singleton design pattern to avoid confusion and ensure the existence of a single scoreboard.

There are many design patterns described in software development literature. The first book collecting design patterns, *Design Patterns* by Gamma, Helm, Johnson, and Vlissides, was written in 1995 and continues to be available today. More patterns have been identified and shared since that time. You can find many listings of commonly used design patterns online such as the one available on the tutorialspoint (https://openstax.org/r/76tutorialspt) website.

Each design pattern provides a solution to a problem in a particular context. Thus, design patterns are not universal solutions as they almost always come with disadvantages, and it depends on the context whether the advantages outweigh the disadvantages, which is why it is important to not only be familiar with design patterns, but also to understand their advantages and disadvantages so that you can assess their benefits and drawbacks in a specific context. Design patterns can be applied when you design a software system, or they can be introduced later on in the process of refactoring. They are rarely applied in their pure form as they are described in literature. The more common practice is to tailor them according to the context in which they will be used. It is also common that the code base of a software system may implement several design patterns, which contributes to the superior quality of the system.

Refactoring

As has been stated earlier, refactoring is the process of restructuring source code without changing its functionality. It is not specific to reuse, though refactoring can involve reuse. Typically, refactoring consists of making a series of elementary changes called micro-refactorings, such as:

- *Rename an attribute*, which changes the name of an attribute to a new one that better corresponds to its

purpose.
- *Rename a method*, which changes the name of a method to a new one that better corresponds to its purpose.
- *Extract a class*, which moves a part of an existing class into a new class.
- *Extract a method*, which moves a fragment of code into a new method.
- *Inline a method*, which replaces a method call by its body. This is typically used only for very short methods.
- *Move a method*, which moves a method to a more appropriate class.
- *Move a field*, which moves a field to a more appropriate class.
- *Remove dead code,* which removes code that never executes.
- *Replace a literal with a symbolic constant,* which replaces a magic value with a named constant.
- *Change method parameters,* which adds or removes method parameters.
- *Substitute an algorithm,* which replaces one algorithm with another. This is typically done to improve performance of the software system.

You can apply these micro-refactorings manually, but it is more common to perform them in an IDE. Modern IDEs provide support for some refactoring, and you can often apply a selected refactoring by activating an appropriate menu item, as shown in Figure 9.26.

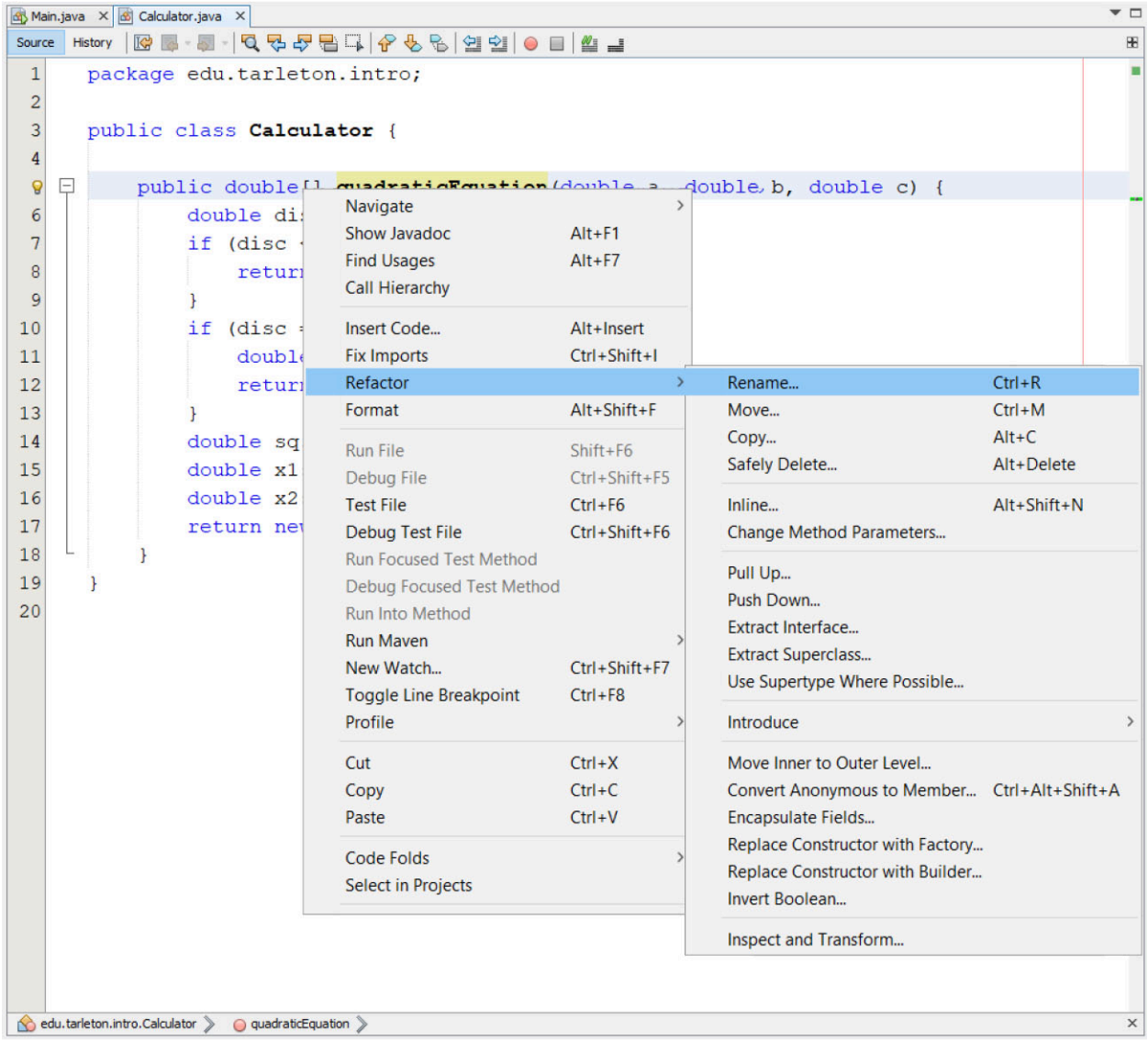


Figure 9.26 In the NetBeans IDE, the action of renaming a method falls under the Refactor menu item. (rendered in NetBeans by Sun Microsystems; attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Returning to the concept of reuse, one of the benefits of refactoring is that functions, classes, and other chunks of code can be extracted from other parts of the code. These can then be reused by placing them in libraries, by incorporating them into a framework to be used by other developers, or by constructing an interface to access them.

Software Licensing

When software engineers design and develop software for an organization, that software and all its code generally belong to the organization. This means the source code and functionality cannot be used by others without permission. This permission is typically controlled by a **software license**. The type of license can be determined by those who created the software or commissioned its creation. Two of the most common categories for defining software reuse are closed source and open source. Proprietary, or **closed-source** software, is a form of licensing that prohibits access to underlying source code. This type of software is typically developed for organizations or as software to be sold. It can be used to protect software that might contain proprietary information or intellectual property within the code. Free and open-source (FOSS), or **open-source** software, is a form of software licensing that provides access to the underlying code and generally allows the code to be reused and modified

It is important to read the license terms of any software code used. It is also important to make sure that, if software is to be shared, it is clear as to what license will be applied to both the use of the software and its code. If a license is not provided with code, then the code is generally protected under copyright rules.

Free and Open-Source Software (FOSS)

Free software (sometimes called libre software) is software distributed under the terms that provide users with the following four freedoms:

1. the freedom to run the software for any purpose
2. the freedom to study how the software works and modify it
3. the freedom to redistribute the software
4. the freedom to improve the software and release the improvements to the public

The freedom to study how the software works and modify it and the freedom to improve the software can hardly be fulfilled without source code, and so these freedoms more or less imply that source code of the software must be available.

Free here refers to liberty, not price (you may be required to pay for free software). Software that is distributed gratis is referred to as freeware, and it rarely comes with source code. Open-source software is software distributed under a license that grants users the right to use, study, modify, and distribute the software and its source code. It does not mean that software must be gratis, although in many cases it is the case. Today, it is common that open-source projects provide their source code via a public repository on the Internet that allows anybody to contribute.

The benefits of FOSS are as follows:

- Improved quality and reliability: Community involvement in open-source projects can lead to better quality and more reliable software due to a wider range of users and developers identifying and fixing bugs.
- Lower development costs: Open-source software can benefit from community contributions, reducing development costs for companies and organizations.
- Innovation: Openness and collaboration can accelerate innovation as developers can build upon existing open-source code.

Some projects that were developed as free and/or open-source software can be categorized as follows:

- Operating systems
  - Linux: An open-source operating system, which is distributed in so-called distributions. The distributions are referred to by name such as Debian, Fedora, Ubuntu, RedHat, and SUSE, and they can be commercial (paid) or community-driven (gratis). The Linux kernel is licensed under GNU General Public License, version 2, which makes it a free and open-source software.
  - Android: A free and open-source operating system developed by a consortium sponsored by Google. It is based on the Linux kernel and other open-source software, and it aims primarily to touchscreen mobile devices, such as smartphones and tablets. Android is the most common operating system (not only on mobile phones).
- Development tools
  - NetBeans IDE: An open-source IDE that started as a student project with the goal of creating an IDE for the Java programming language. Now it belongs to Apache Software Foundation and also supports other programming languages, such as PHP, C, C++, and JavaScript.
  - Eclipse IDE: An open-source integrated development environment that started at IBM and is now managed by the Eclipse Foundation.
  - GNU Compiler Collection (GCC): A free and open-source compiler (or more precisely a suite of compilers) that supports various programming languages, hardware platforms, and operating systems. It is the standard compiler included in many Linux distributions.

- Web browsers
  - Google Chrome: A no-cost web browser developed by Google. Although most source code of Google Chrome comes from free and open-source software, it is licensed as proprietary freeware. It is available on Windows, Linux, Android, macOS, and iOS.
  - Mozilla Firefox: A free and open-source web browser developed by Mozilla Foundation. It is available on Windows, Linux, macOS, and Android.
- Programming languages
  - OpenJDK: A free and open-source implementation of the Java Platform, Standard Edition. Many other distributions of the Java Platform, such as Amazon Corretto, are based on OpenJDK.
  - C#: A modern cross-platform programming language initially developed by Microsoft. Both the C# language and the current (as of 2014) Microsoft C# compiler are open source.
  - Python: A modern programming language is that is completely open source, freely usable and distributable, even for commercial use, making it extremely popular with businesses around the world.
- Application servers
  - Payara Server: An open-source application server, with paid support options.

While some open-source software is entirely free, FOSS projects can have successful business models. It may seem that open-source software development does not bring any benefits to software publishing companies, but community involvement leads to better quality and higher reliability of the software and lower cost of development. Many software companies that moved to open-source development adequately modified their business model and instead of selling software, they charge the customer for providing additional services. For example, Payara Server is an open-source application server for Jakarta Enterprise Edition applications. The server is gratis, but companies that use Payara Server for mission-critical applications can pay for support, which allows them to consult a team of software engineers at any time. In addition, they receive monthly releases with bug fixes and security alerts. Many companies that embrace open-source development focus on providing value-added services, such as technical support, training, or custom development, around the core open-source product.

It is important for companies and developers to pay attention to the following:

- Software licensing considerations: Understanding software licenses is crucial for both software users and developers. While many people accept End User Licensing Agreements (EULAs) when installing software on their personal computers and devices, it is important that software engineers understand that these are legal agreements. As such, when using software to build solutions, it is important to understand what the licensing allows.
- Using open-source software: Software developers should be aware of the license terms associated with any FOSS they use to ensure compliance.
- Distributing software: Additionally, when distributing software, it is important to make sure that an included license defines how you or the organization that owns the software wants it used and protected. Choose a license that aligns with your project's goals and how you want your software to be used and distributed.

FOSS is a powerful development model that fosters collaboration, innovation, and cost-effective software creation. While security considerations should also be factored in, open-source software plays a vital role in the tech industry.

Software Engineering Ethics and Legal Aspects

Software engineers often spend a lot of time creating code and solutions to solve problems. While they might be the ones writing the code, that does not mean the code or the knowledge that they gained while writing the code belongs to them.

For example, if a software engineer worked for an auto manufacturer, they might be tasked with designing and creating a futuristic product that detects the eye movement of a driver to verify they are looking forward while driving. The software might present an alert if the driver's attention wanders from the road for longer than a given period. Generally, it would be considered unethical for this same developer to use what they learned while working for the auto company to develop their own software product that uses the same code to detect eye movement. While the software engineer might have written the code, they wrote it for the auto company, so it would be unethical to use the same code in an independent product without the auto manufacturer's permission.

Software engineers should abide by a code of ethics that guides the work that they do and the products that they produce. At a personal level, ethics for software engineers involve the following guiding principles:

- Do not use other people's data or code for financial gain.
- Do not leverage other people's proprietary information as part of a commercial project.
- Do not use or hide the use of other people's data or programs as part of your own projects.
- Do not violate the privacy of others.
- Do not gain wrongful access to a system for financial gain.
- Do not create or propagate computer viruses or worms.
- Do not create or use programs that promote discrimination or harassment of any kind.

The Association for Computing Machinery (ACM) and IEEE-CS established the Committee on Professional Ethics (COPE), which published a Software Engineering Code of Ethics and Professional Practice. This code of ethics states that software engineers shall commit themselves to making the analysis, specification, design, development, testing, and maintenance of software a beneficial and respected profession in accordance with their commitment to the health, safety, and welfare of the public. The Code goes on to define eight principles that all software engineers are required to follow to possibly avoid legal consequences:

1. Responsibility should be taken for the work done.
2. Actions should be in the best interest of a client or employer.
3. Products and updates should be created in a manner that adheres to the highest professional standards.
4. Integrity should be applied in actions taken.
5. An ethical approach to managing software development and updates should be maintained and promoted.
6. The integrity of what it means to be a software engineer should be maintained.
7. Software engineers should be supportive of peers and colleagues in a manner that is fair.
8. Learning should be a continual endeavor to the betterment of the software engineer's own profession.

LINK TO LEARNING

You can find the full code of ethics and professional practice on the ACM website. This includes an overall preamble (https://openstax.org/r/76ACMethics) as well as details for each of eight guiding principles.

THINK IT THROUGH

Applying the Concepts of Ethics

As a software engineer, you will deal with the constant change of technology and the constant need to understand a variety of business scenarios to build solutions. Clearly, you cannot be expected to know everything.

Consider two hypothetical scenarios: In the first, suppose that a developer, when faced with a task they did not know how to do, spent days avoiding the rest of the team as they researched and read books to try to learn the new technology. They did not tell anyone they did not know the technology.

In the second, imagine a developer was faced with the issue of needing to code for a widget for the team to use. The issue was shared online in developer forums such as StackOverlow and CodeProject and the developer asked others to provide code to do what the widget was expected to do. While the developer did not understand what the code did, he copied what he was given and presented it to the team as his own without saying where it came from.

In both situations, how did the concepts of ethics apply?

Road Ahead for Software Engineering

People have vastly different opinions as to what software engineering will be like twenty or thirty years into the future. One day, revolutionary discovery or invention may change our society in a radical way—or not. Nobody can predict when or what specific scientific breakthroughs will happen, and discoveries can shape our future in ways that are hard to imagine today. Based on our experience of the past twenty or more years, however, we can make some predictions about where software engineering may be in 2040.

Evolution of Programming

We can, for example, predict that the evolution of current programming languages will be driven by changes in hardware and that the tools developers use will improve. But with the evolution of AI and machine learning, will software developers become obsolete? Under current circumstances, the answer is no. AI can, however, help with some developer's tasks, such as testing, debugging, and refactoring. While we can expect that intelligent bots will be members of development teams, they will not fully replace software developers.

Future of Software Development

Now let's discuss three questions about the future of software development:

1. What programming languages will we use? We can expect that current programming languages will evolve, perhaps even become obsolete, by 2040, but it is hard to imagine that languages like Java, Python, C, C#, and C++ will disappear entirely because there is so much code written in these programming languages.
2. Can we get to the point where all software is written, and we do not need any software development? Absolutely not in the near future. We are not at the end nor in the middle of a technology revolution—we are at the beginning. The importance of software in society will grow, and software systems will become more sophisticated. For example, with the Internet of Things (IoT), we will connect thousands and thousands of smart devices to the Internet, and they all will need software.
3. Does it make sense to learn programming today? Absolutely. With the increased use of smart, connected devices, the integration of technology is getting more rather than less prevalent in almost all facets of life, and technology continues to offer an advantage to businesses to operate quicker, better, and more effectively than the competition. Given this, software development will become more important.

Additionally, electronics (including computer hardware) and the way computer programming is applied continue to evolve. In the early 2000s, mobile computing and Wi-Fi connectivity changed the way computing happened. In the past decade, connectivity continued to evolve, and many new types of smart devices needed new code. Over the next few decades, changes in the core of computers, such as the creation of quantum coprocessors, have the potential to revolutionize software development yet again.

In 2021, the U.S. Bureau of Labor Statistics predicted that over the next decade, the number software engineering jobs in the United States will increase by 25%[8]. This is, in part, because technology continues to infiltrate our daily lives. For example, many grocery chains are updating their shopping carts to include sensors that record all of the items you add to the basket area. As this automation continues and expands, it will require a lot of software developers. The current push for autonomous vehicles will be accomplished as a result of sensors and a large number of software engineers creating hundreds of thousands of lines of code. It will take years of testing, debugging, and refactoring before autonomous vehicles are as commonly accepted as a mobile phone is today. It will be software engineers who make that happen.

8 Bureau of Labor Statistics, U.S. Department of Labor, *Occupational Outlook Handbook*, Software Developers, Quality Assurance Analysts, and Testers, at https://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm (visited *June 04, 2023*).

Chapter Review

Key Terms

acceptance testing process used to determine whether the software solution fulfills the customer's expectations, also called user acceptance testing

accessibility testing subset of usability testing that focuses specifically on the needs of users with disabilities

Agile Software Development Ecosystem (ASDE) Agile SDLC frameworks and methods that emphasize the difficulty of predicting future needs and thus avoid creating long-term plans and fixed processes so developers can instead collaborate with customers and adjust to their current needs

application software software that enables you to fulfill common tasks such as creating a text document, drawing a picture, or playing music

architecture management helps improve the architectural design while it is under development

automated testing form of testing that uses software tools to check functionality of a software system or some of its parts; this allows for repeating tests without requiring repeated effort on the part of the tester

availability measure of how busy a system is when a user attempts to use it

black box testing manner of testing where the tests are based on the requirements and the functionality of what is being tested without the need to focus on the code itself; input-output testing, specification-based testing, and behavioral testing

bug issue or error with software programming

bug tracking system aids in the tracking and resolution of fixing issues with software

closed-source form of software licensing that prohibits access to underlying source code; this type of software is typically developed for organizations or for software that is meant to be sold

code coverage measurement of the percentage of code that is activated or reviewed by unit tests

code review process whereby the source code written by one developer is manually inspected by another developer or a team of reviewers

construction phase framework activity in which the design documents are used to write corresponding source code in a programming language and to create any supporting assets

containerized software packaged in such a way that it can run on different computer systems easily

crosscutting activity (also: **umbrella activity**) activity that crosscuts the entire software development process but is not part of the main building steps themselves

debugger program that assists in detecting and correcting bugs; it does this by allowing the user to pause program execution at any statement, view the state of the program at that moment, before allowing the computer to continue until the next statement at which they want to pause

deployment framework activity that involves making the developed software solution available to users

design pattern reusable solution to a design problem that software engineers repeatedly encounter while architecting and designing systems

detail-level design (DLD) focuses on detailing or expanding upon the HLD; as part of the DLD, every element of a system is provided with detailed specifications, and the logic for each component within each module of a system solution is determined

DevOps model Agile software process model that combines practices of software development and operations and uses a short development life cycle and continuous delivery to achieve high-quality software products

dynamic quality nonfunctional feature that relates to the qualitative behavior of software while it is in use, which means that it also depends on the hardware that the system runs on

elaboration phase framework activity that involves further analyzing the requirements to produce design models of the system to be developed

embedded software software that is integrated with hardware and can include both application and system software features

extensibility measure of the amount of work and cost required to add new features to software
flexibility measure of the amount of work and cost required to make changes to a system when requirements change
functional requirement based on expectations of the user for the inherent characteristics of the software
gray box testing manner of testing in which the person who designs the test has a partial knowledge of code structure and understands the intended design of the software; it is a hybrid of white box and black box testing
high-level design (HLD) focuses on providing a general description of the overall system design, and can include information on the overall aspects of a system, including its architecture, data, systems, services, and platforms as well as the relationships among various modules and components
inception phase framework activity that involves focusing on the gathering and refinement (i.e., definition) as well as the management of functional and nonfunctional requirements, which is also known as requirements engineering
incremental model software process model in which software development is divided into modules, and each module focuses on a smaller set of requirements based on an overall business plan
legacy software software that has been written in the past, relies on obsolete technology, and is still in use today
line coverage type of code coverage that measures the percentage of activated lines of source code that are tested in a unit test
maintainability measure of the amount of work required to make changes to a software system
maintenance process of updating software after it is deployed
manual testing form of testing where a person must run the software system, provide any input, and manually check all output; repeating tests requires these efforts to be repeated by the tester
nonfunctional requirement describes a desired quality feature and covers aspects such as flexibility, maintainability, performance, portability, reliability, scalability, security, and more
open-source form of software licensing that provides access to the underlying code and generally allows the code to be reused and modified; also known as free and open-source (FOSS)
operating system software that controls and provides access to the computer's basic functionality
path coverage type of code coverage that measures the percentage of paths through source code that you go through in a unit test
pattern high-level concept that supports the idea of reuse and provides reusable solutions to problems often encountered when building software
performance measure of response time as seen by the user
physical design graphical representation of the method for effectively implementing what was determined in the logical design of a software solution
portability measure of the amount of work and cost required to migrate software solutions to a new platform, such as a new operating system
prescriptive process model advocates an orderly approach to software engineering that involves following a prescribed set of activities in a continuous manner; in contrast to Agile development
profiler program that performs dynamic program analysis that can be used to optimize or otherwise streamline code
prototyping model software process model that requires the quick creation of a mock-up (or demo) of the expected final product doing what it is expected to do so that end users may provide feedback; also known as RAD (Rapid Application Development)
refactoring process of restructuring source code without changing its functionality
requirements modeling software engineering action that is part of the inception phase and focuses on the analysis/decomposition of software requirements
scalability measure of the amount of work and cost required to modify a system to provide higher throughput
scenario specific instance of operational flow within a use case that is focused on understanding a specific

action

Scrum type of Agile software development model

security measure of confidence that data is protected from unauthorized disclosure and that systems are protected from unauthorized access

software architecture description of the overall structure of a software system, its major components, and their interrelationships

software design engineering design task set in which the abstraction and refinement of requirements are formed into a specification that can be used in the creation of a software solution

software development life cycle (SDLC) structured set of framework activities required to develop a software solution based on a set of requirements

software license documentation that determines whether an organization's source code and functionality can be used by others without permission

software process improvement process of transforming the existing approach to software development into something that is more focused, more repeatable, more reliable (in terms of the quality of the product produced and the timeliness of delivery), and more cost-effective

Software Quality Management (SQM) focuses on the development and management of the quality of the solution being developed

spiral model software process model that is a combination of the waterfall model with an iterative model approach and that focuses on reducing risk within a project

sprint fixed-length workflow event that is part of Agile software process development and typically runs one to four weeks

statement coverage type of code coverage that measures the percentage of statements that are activated at least once when you run all unit tests

static quality nonfunctional feature that is unchangeable and thus might be associated with the source code or with legal or project-environment specific requirements

system software software that enables you to control hardware and provides an environment in which other software can run

system testing focuses on the complete and fully integrated software product to make sure the complete software solution works on the whole as expected

test-driven development (TDD) process where developers write tests before they write code

throughput measure of the total amount of input data that may flow through a system; it is different from performance, which measures how fast a system can perform its functions

traditional process model process framework that encompasses four framework (i.e., generic) activities that are also known as phases: inception, elaboration, construction, and deployment

umbrella activity (also: **crosscutting activity**) activity that crosscuts the entire software development process but is not part of the main building steps themselves

Unified Modeling Language (UML) visual modeling language that can be used to capture the result of software analysis and design

Unified Process (UP) model software process model in which the development of a software system is divided into four primary phases (inception, elaboration, construction, and transition), each of which involves multiple iterations

unit testing tests individual units of code, such as methods and functions, and is usually done by developers during the development of the software or when updates are made

usability measure of how intuitive the user interface is

usability testing confirms that the software being developed not only meets the requirements that were set by the user, but is also easy and intuitive for the user to use

use case describes how software system is expected to be employed by users to accomplish a goal or requirement

user interface/user experience (UI/UX) part of computer programming that is concerned with how information is presented to the user and how the user can interact with a program

user story generic explanation aimed at the user to tell them how a software feature works

V-model software development process model that is similar to the waterfall model in that it is a continuous prescriptive model, but it is associated with a verification or validation testing step/phase, and thus also known as the verification and validation model

validation tests that the software solution conforms to the requirements and, therefore, does what the user wants it to do

verification tests that the software solution functions without errors

version control system tool used to store the history of changes to source code and facilitates collaboration of multiple developers

waterfall model continuous prescriptive software process model in which phases "flow" into another the way water flows from the top of a waterfall down to the bottom

white box testing manner of testing where the tester uses the source code so they can develop tests that verify that the internal structure of the item being tested works properly; also known as glass box, clear box, and structural testing

Summary

9.1 Software Engineering Fundamentals

- Software engineering is concerned with how to effectively develop software.
- Software engineering is about solving practical issues that arise in software development, while computer science is more about theoretical principles used in software.
- A good software engineer should have a solid background in computer science and must have good practical skills in areas such as computer programming.
- Software is usually developed in a team in which people have different roles and responsibilities.
- Developing software involves addressing both the described desired functional and nonfunctional features.
- There are a number of categories of software, including application, system, and embedded.
- Software engineers should understand software engineering processes, specific activities such as testing, and software tools.
- Soft skills such as communication and problem-solving skills are important to being successful in the role of a software engineer.

9.2 Software Engineering Process

- A software engineering process framework is used to define the software process model or SDLC model that will be used to create a software solution.
- Most SDLC models use phases or standard framework activities. The four common phases are inception, elaboration, construction, and deployment.
- SDLC models differ in terms of the rigidity and number of software engineering actions they specify need to be completed as part of the software development process.
- One major category of process models is the traditional process models, which are usually prescriptive, sequential, and may not be suitable for projects with rapidly changing requirements.
- A second major category of process models is the Agile process models, which are less rigid, allowing for activities to skipped or accelerated to deliver a project solution faster and integrating user input early in the process.
- In software development, developers use software architecture to view and evaluate the system as a whole before moving to component design.
- In addition to the generic framework activities, there are other activities that crosscut the entire software development process but aren't part of the main building steps themselves, such as communication and training, risk management and planning, software configuration and content management, software quality management (SQM), architecture management, and software security engineering.
- Software engineers should understand the various models and how they differ. Some of the popular

software process models include the waterfall model, V-model, incremental model, prototyping model, spiral model, Unified Process model, and Agile Process models (such as Scrum and DevOps, among others).

9.3 Special Topics

- Testing is a critical part of building successful software solutions.
- To ensure quality, software goes through various levels of testing, such as unit testing, integration testing, and system testing, and it is subject to different testing approaches, such as acceptance testing, usability testing, stress testing, performance testing, and security testing. Each test serves an important function.
- Test-driven development in a process in which developers write unit tests before they write code. While counterintuitive, this can help clarify requirements, facilitate early bug detection, and even improve design and code quality.
- Whereas unit testing focuses on the various units or pieces within a software solution, system testing focuses on the complete and fully integrated software product. Systems can be manual or automated.
- Acceptance testing is used to determine that the software system works as was specified in the requirements. Taking place after system testing, this is typically done by the customer, client, or other end user of the software.
- Usability testing tests the user interface by gaining feedback from a small group of users who are representative of the target audience.
- When developing software, developers use tools such as compilers, debuggers, profilers, integrated development environment (IDEs), version control systems, and bug tracking systems.
- To resolve problems that commonly occur in the development of a software system, developers use reusable solutions called architectural patterns and design patterns.
- Refactoring is a technique for restructuring an existing piece of code without changing its functionality.
- Open-source software is provided with source code. Examples of open-source projects are Linux, Android, NetBeans IDE, Eclipse IDE, and the C# programming language.
- Whether you simply use software or create software, it is important to understand licensing so that you know what you can or cannot do with the software.
- Software engineers should abide by a code of ethics that guides the work that they do and the solutions that they produce.
- The ACM/IEEE-CS Joint Task Force produced a Software Engineering Code of Ethics and Professional Practices that states that software engineers shall commit themselves to making the analysis, specification, design, development, testing, and maintenance of software a beneficial and respected profession.
- Software engineering and the development of software solutions is evolving. As technology changes, so do some of the ways software is coded.



Review Questions

1. List some of the best practices related to software engineering.
2. What is a true statement about the relationship between software engineering and computer science?
   a. Software engineering and computer science are unrelated.
   b. Software engineering is a subset of computer science.
   c. Computer science is a subset of software engineering.
   d. Computer science and software engineering are the same thing.
3. List five types of software that a software engineer is likely to work on.
4. List five soft skills software engineers should have.
5. Software engineers must have a solid knowledge of several technical areas. List six areas other than soft skills that were mentioned in this chapter.

6. What are common roles in a software team?

7. What does SDLC stand for?
- **a.** system design life cycle
- **b.** software development life cycle
- **c.** system development life cycle
- **d.** system design life cycle
- **e.** software delivery life cycle

8. Name the four common phases used in the SDLC.

9. List four SDLC models.

10. What is a stated part of the software process improvement?

11. What are the disciplines of Unified Process?

12. What are the phases of Unified Process?

13. What type of testing focuses on making sure the individual pieces of code that were written are working correctly?
- **a.** acceptance testing
- **b.** performance testing
- **c.** stress testing
- **d.** unit testing
- **e.** usability testing

14. What type of tests would generally be performed by a software engineer?

15. What is the primary purpose of verification?
- **a.** to run the code to verify it works correctly
- **b.** to make sure the software solution conforms to the requirements that were specified for the project
- **c.** to make sure the software solution does what the user wants it to
- **d.** to make sure all the project owners agree to move forward with each phase of the software development life cycle

16. What is the purpose of validation?
- **a.** to run the code to validate that it works correctly
- **b.** to make sure the software solution does what the user wants it to
- **c.** to make sure all the project owners agree to move forward with each phase of the software development life cycle
- **d.** to make sure all lines of code will be executed

17. What is the key defining element of test-driven development?
- **a.** Each line of code is given its own test.
- **b.** Tests are written in a manner that requires multiple-choice solutions.
- **c.** The testing team is responsible for all tests that are to be written.
- **d.** Tests are written before the code.

18. What is a debugger?

19. What is an integrated development environment?

20. What is a version control system?

21. Can open-source or FOSS software be used at no cost?

Conceptual Questions

1. Explain what is wrong with the notion that computer software does not need to evolve over time.
2. Explain why nonfunctional requirements are an important part of the requirements for software products.
3. Explain why a software engineer needs to have solid knowledge of programming languages.
4. What is an iteration in iterative development?
5. What is an increment in iterative development?
6. What are the key issues addressed by an Agile approach in software engineering?
7. What are the drawbacks of the waterfall model?
8. What are the benefits of DevOps?
9. What is the overall purpose of testing in software engineering?
10. What is path coverage?
11. What are three ways code coverage can be measured and what does each do?
12. What are design patterns? Provide an example of a design pattern.
13. What is the benefit of licensing software?
14. What type of license could you use if you were okay with sharing your application's source code and allowing others to use or modify it?
15. Can we get to a point in the next twenty years where we will no longer need software engineers or to write code?
16. Ethically, what are five things listed in this chapter a software engineer should never do?

Practice Exercises

1. List five computer technology areas that a software engineer should be familiar with. Consider how each could impact or be applied to a software solution.
2. List five soft skills that are important for software engineers
3. Categorize the following software programs as system software, application software, or embedded software: Linux Debian, Windows 11, Microsoft Word, a website, a Samsung refrigerator door sensor with a small CPU chip, and a sensor on a street lamp with a small CPU chip.
4. You've been assigned to lead a project and your first task is to determine the software development life cycle you should use. What are things you should consider that will impact which SDLC model you should pick?
5. You are the engineering manager for a software development team. A new client comes to you and offers to fund a new project for your team. The client has a poorly developed set of requirements upfront but has an urgent need to get working software. How could you satisfy your client's needs?
6. Research DevSecOps and summarize how it is different from DevOps.
7. Analyze processes such as RE, SBSE, and TDD, and explain how they improve traditional software engineering processes from an SPI standpoint.

Problem Set A

1. Determine at least six functional requirements for a navigational software program that could be used

within an automobile.

2. If you were asked to define functional (software) requirements to build a digital pencil (e.g., stylus), what are four possible requirements? Note that while a stylus includes a mix of hardware and software requirements, the same level of thought and detail goes into the process of defining requirements for the software portion.
3. What would be some of the nonfunctional requirements applicable to a virtual reality (VR) application?
4. You've been asked to help determine the requirements for a new software product. This product is a simple calculator. The project is expected to use an iterative approach with at least three iterations. Consider the following requirements and determine who you might prioritize and group them. Explain the reasoning behind your prioritization, considering factors such as user value and project complexity.
    a. Program should accept whole and rational numbers.
    b. Program should allow for addition, subtraction, multiplication, and division.
    c. Program should allow for determining the square root of a number.
    d. Program should display the results of the operation only after the equal button is selected.
    e. Program should allow for grouping of actions by using parentheses. For example, the calculator should allow the following to be entered: 2 + (10 / 2) = and should receive the result of 7, not 6 or any other value.
    f. Program should allow for numbers to be in decimal or octal format.
    g. Program should allow the user to clear the last value entered or all values entered.
5. Research Scaled Agile Framework (SAFe) and summarize how it can be useful for projects of a very large size and how organizations can use it to effectively manage multiple teams in an Agile development environment.
6. Often Agile development is used to create products that are based on emerging technologies. Why do you think this is?
7. Why does part of the Agile development model involve minimizing documentation?
8. Identify specific business or personal situations related to ethics in software engineering, and explain how you would react to them.
9. Research and compare pros and cons of open-source software and paid software licenses.
10. Research various Integrated Development Environments and summarize why some IDEs are aimed at different types of development efforts and programming languages.

Problem Set B

1. Choose a software program you use frequently. This could be a game, social media platform, productivity tool, or anything else. Analyze its functionalities and identify six areas for improvement. For each improvement, describe the benefit it would bring to the user experience.
2. Imagine you're designing a software system for a library. List at least six functional requirements this system should meet.
3. Besides functionality, what are at least three nonfunctional requirements that are important for a library information system? Explain why each requirement is important.
4. What are the questions that should be answered by each team member at the daily Scrum meeting?
5. Safety critical applications are key components of a system that would severely impact the safety and well-being of the users and bystanders. Provide an example of a safety critical application that, if the software of that component would fail, could cause catastrophic consequences to the life and safety of users and

bystanders.

6. Research the Boeing 737 Max 8 Maneuvering Characteristics Augmentation System (MCAS) software and sensor problem. What went wrong and how could Boeing and the Federal Aviation Administration (FAA) have done a better job of developing and testing a safety critical application?
7. As a software developer, you wrote a routine that converts degrees Fahrenheit to degrees Celsius and you want to test it. How would you suggest performing unit testing of this functionality?
8. Explain how you could create an open world machine learning bot that could trade on a financial exchange.
9. Identify new examples of software as a differentiator for products and services.



Thought Provokers

1. Consider our start-up company that is 100% committed to leveraging innovative technologies as a business growth facilitator. Describe how it can best use software security engineering to create products or services that can generate business. Give precise examples and explain how the start-up would be able to scale the resulting business (i.e., keep sustaining the cost of doing business while increasing its number of customers).
2. In the health-care industry, the Health Insurance Portability and Accountability Act (HIPAA) mandates that electronic billing and the sharing of client information cannot be done without proper software support. Software plays a crucial role here because it enables electronic communication, and it lowers expenses in comparison to information exchange in traditional paper form. Other than health care, what are other industry examples that have changed drastically in the past decade as a result of applying software as a part of the solutions they offer? Note that the automotive industry was already mentioned in the chapter.
3. Virtual reality is a technology that has gotten a lot of attention recently. With companies like Microsoft promoting Mixed Reality and Meta (the company previously known as Facebook) promoting their Metaverse, VR has big companies behind it. Virtual reality, however, has been promoted in the past by game companies such as Nintendo but ended up fading away. Will virtual reality fade away again? What will make virtual reality a success? What impact do software engineers have on its success?
4. As the project leader for the software engineering of the new features of next year's car model, you have been tasked to deliver the lane departure system, additional navigational features on the dashboard computer screen, and a heads-up display system that shows the speed and any warnings displayed onto the windshield. When the product was demoed, the user of the system indicates that they also want to include navigational aids from the navigational system onto the windshield heads-up display. When the navigation system is providing directions, the heads-display on the windshield should show an arrow when getting close to a turn. This functionality was not a part of the original requirements, but rather is a change to the scope of the project. What do you do? What impact will this have on the process and SDLC models being used for the project? Further imagine the new functionality is critical for safety and could potentially prevent accidents. Would your approach change?
5. Consider our start-up company that is 100% committed to leveraging innovative technologies as a business growth facilitator. Describe how it can best use software process model selection and refinement to create products or services that can generate business. Give precise examples and explain how the start-up would be able to scale the resulting business (i.e., keep sustaining the cost of doing business while increasing its number of customers)?
6. Consider our start-up company that is 100% committed to leveraging innovative technologies as a business growth facilitator. Describe how it can best use the open world software approach to create products or services that can generate business. Give precise examples and explain how the start-up

would be able to scale the resulting business (i.e., keep sustaining the cost of doing business while increasing its number of customers).

7. It is common for software engineers to help determine requirements for a project. What are potential issues that can arise from having software engineers make decisions on requirements?

8. What are the ramifications of releasing software that has issues? How could this impact customers? How could it impact the software engineers who worked on the project? Consider the features discussed at the beginning of this chapter for software in the automotive industry. What would the impact be of errors in software for features such as lane departure system, cruise control, navigation, or augmented heads-up displays?

Labs

1. Researching traditional software process models: Several traditional prescriptive process models were not covered in this section (e.g., Phased model, Aspect-Oriented Software Development (AOSD) model), and related alternative techniques (e.g., Formal Transformation Process model, Reuse-Based Process model, Rapid Application Development, Personal Software Process model/Team Software Process model, CMM, SEI's IDEAL Model, Component Based Software Engineering or CBSE software model, Hybrid Process models). Find reliable sources such as articles or websites that explain each framework. Answer the W5 questions (Who, What, When, Where, Why) for each framework:
   Who: Who created or popularized the framework? Is it used by any specific industries?
   What: What are the core principles and practices of the framework?
   When: When might this framework be a good choice for a project?
   Where: Are there any examples of companies that use this framework successfully?
   Why: Why might a team choose this framework over others?

   Identify a real-world application: Imagine you're working on a software project to develop a mobile game with a small team. The requirements are well defined and changes are not expected. Based on your research, which traditional software process model would you recommend for this project? Why?

2. Researching Agile process models: Several Agile frameworks or methods were not discussed in this section (e.g., Adaptive Software Development, Dynamic Systems Development Method, Crystal, Feature Driven Development, Incremental Funding Method IFM, Kanban, XP, Incremental Prototyping Model, Agile Pattern-Driven Approach, Design Thinking Method, Lean Startup Method). Find reliable sources such as articles or websites that explain each framework. Answer the W5 questions (Who, What, When, Where, Why) for each framework:
   Who: Who created or popularized the framework? Is it used by any specific industries?
   What: What are the core principles and practices of the framework?
   When: When might this framework be a good choice for a project?
   Where: Are there any examples of companies that use this framework successfully?
   Why: Why might a team choose this framework over others such as Scrum?

   Identify a real-world application: Imagine you're working on a software project to develop a mobile game with a small team. The requirements are well defined but there is a need to be flexible and adapt to user feedback quickly. Based on your research, which Agile framework (Scrum, Kanban, or one you researched) would you recommend for this project? Why?

3. There are a number of software development tools that are freely available or that can be accessed and run from the web. One such online integrated development environment is Replit, which can be found at Replit.com. This IDE allows you to enter programs using a variety of programming languages, such as C, JavaScript, Python, C++, Java, and more. Create an account on Replit.com. Once your account is created, go to to access tutorials (https://openstax.org/r/76Replit) on using Replit and programming. Pick an Introductory tutorial from the site on a programming language such as JavaScript and complete it.

4. Explore Visual Studio Code (https://openstax.org/r/76VisStudioCode). Describe what it does and how it relates to software development.
5. Read excerpts from Kurzweil's *The Singularity Is Near* and his more recent book, *How to Create a Mind* as needed to justify his prediction regarding digital utopia and immortality as human capabilities (and lifespans) are amplified using technology. Also, analyze Bill Joy's prediction of digital dystopia and human extinction as machines become more and more capable. Express your point of view regarding these two diametrically opposed visions and how you believe it could influence the way software engineers approach the definition of a long view for the human race.



10

Enterprise and Solution Architectures Management

Figure 10.1 Metaphoric architecture incorporates symbolic elements and patterns into the design to create narratives in a space, such as the windows in this government building that represent transparency. Computer science also uses patterns for solving problems and creating narratives. (credit: modification of “Keep 'em coming” by Dylan/Flickr, CC BY 2.0)

Chapter Outline

10.1 Patterns Management
10.2 Enterprise Architecture Management Frameworks
10.3 Solution Architecture Management

Introduction

TechWorks is facing many technical problems such as difficulties in transferring the data between the divisions, sharing resources, and end user training. The CIO suggested hiring a solutions architect manager who will be responsible for building teams, establishing relationships, setting strategy, and proposing solutions for the current problems as well as finding any available opportunity.

The first mission for the new solutions architect manager, Mr. John, is studying the current system design by setting up a meeting with the enterprise architects and the solution architects. Mr. John has asked the solution architects to finalize the design and implementation of the solution and the enterprise architects to verify that the information technology strategy is aligned with the enterprise mission by analyzing the business strategy. Altogether as a team, they created a plan for the solution by dividing the problem into subproblems using a divide and conquer approach.[1] They reviewed the enterprise levels to verify proper bookkeeping of architectural knowledge, techniques, and artifacts and their impact on architectural designs at various levels of scope. In the middle of the management process, Mr. John discovers that a data analyst and several software engineers need to join the team. The data analyst will define the required information for each level. The software engineers will leverage prior architectural knowledge at all levels of scope to create next-generation, secure, super-smart society, intelligent, and autonomous solutions. The team created many patterns to help address the problems, supported the creation of solutions, and put in place a framework ensures that enterprise solutions are in alignment with the evolving vision and strategy of the organizations that uses these solutions to operate and conduct day-to-day business. The main recommendation from the team to implement super-smart society, intelligent, autonomous solutions is to adopt the Microsoft Azure Cloud Platform as a Service and a target solution implementation framework to support customer-facing and internal business functions.

10.1 Patterns Management

Learning Objectives

By the end of this section, you will be able to:

- Understand how to apply patterns at various levels of scope
- Organize patterns in hierarchies and leverage pattern catalogs and pattern languages
- Relate to the current state of patterns management

A pattern documents a recurring problem and provides a reusable template in the form of a problem-solution pair within a given context. A pattern is not only a pairing of problem and solution, it provides the rationale that binds the problem and the solution together. Each pattern deals with a specific recurring problem in the design or implementation of a business solution. Patterns can be used to construct architectures at various levels of scope to guarantee specific properties. Patterns also capture existing, well-proven experience in solution development and help promote good design practice.

Pattern Hierarchy

Patterns range from abstract to concrete. Abstract patterns represent concepts without physical references (i.e., are not available to the senses). Concrete refers to objects that are available to the senses. For example, freedom is abstract, but the book (on freedom) is concrete.

Design-centric patterns are organized in a **pattern hierarchy** that includes architectural styles, architectural patterns, and design patterns. An abstract pattern located at the top of the pattern hierarchy is called an **architectural style**; it captures a set of characteristics and features that make a structure identifiable. Styles are designed to capture knowledge of effective design for achieving specified goals within a particular context. An architectural pattern conforms to specific architectural styles and embodies architecture design decisions that are applicable to recurring design problems. They provide parameterized templates that can be adapted to provide solutions that are suitable to different development context. Design patterns conform to specific architectural patterns and provide granular design components to articulate them.

1 Divide the problem into subproblems and then combine the subproblem solutions to a final solution.

Once a technology stack that picks specific technologies that should be used to implement the solution has been selected, design-centric patterns can be implemented. Each technology stack has a corresponding implementation-centric pattern hierarchy[2] that includes implementation styles, implementation patterns, and idioms. An **idiom** is a phrase or expression whose meaning cannot be inferred from the literal definitions of its individual words, but instead is understood through common usage within a language.

Solution architects create several complex processes to successfully implement a business solution. These steps are:

1. Study the elements of technology that can be applied to solve a specific problem.
2. Propose a combination of building blocks for the best possible fix to the problem.
3. Design a solution and manage the implementation.

After the solution architect finalizes the design and implementation of the solution, the enterprise architects verify that the information technology strategy is aligned with the enterprise mission by analyzing the business properties. Enterprise and solution architects use pattern hierarchies to create best practices for business and technical architectures that guide the implementation of solutions to business problems. Generalization and information hiding are used to keep more abstract patterns at the top of the hierarchy. The technology that allows the management of architecture results from decades of research in patterns management and frameworks used to plan and deliver solutions that meet business and technology strategies.

TECHNOLOGY IN EVERYDAY LIFE

Patterns in Everyday Life

Patterns help us to organize our life activities. For example, using patterns, we can establish our daily activities using process patterns such as “start my day” and decompose this pattern into more specific process patterns such as waking up, eating breakfast, and getting ready for work. It is never the case that specific actions that correspond to these activity patterns occur the same way every day.

How does the knowledge of patterns help people in everyday life? Provide a couple of illustrative scenarios to explain your opinion. Your scenarios should not be limited to integrating software patterns but rather describe scenarios where people can refer to patterns to analyze real-life situations.

Analysis and Design Model Patterns

A blueprint is a high-level plan used in the development stage, and an **enterprise architecture (EA)** is a conceptual blueprint that defines the structure and operation of organizations. Design modeling provides a variety of different views of the system like architecture plans for the enterprise. Examples of a design-centric architectural style are microservices. An example of a corresponding implementation-centric implementation style is REST Services. When the problem appears, an architectural pattern embodies architectural design decisions that are applicable to a recurring design problem parameterized to account for different solution development contexts.

A design pattern relates to common design structures and practices that enable the creation of reusable software. An example of a design pattern is a *singleton*. A **singleton** restricts the instantiation of a class and ensures that only one instance of the class exists (e.g., a scoreboard object in a game should be derived from a singleton class to ensure that there is only one scoreboard object and one set of scores for the players at any given time). The enterprise architecture scope includes:

- Enterprise scope: higher-level patterns that can be applied to the overall structure of the enterprise

2 There may be several implementation-centric patterns that realize a given design-centric pattern.

- Solution scope: the pattern that can be applied to a single solution or system
- Domain scope: the pattern that can be applied to a specific domain

A pattern hierarchy applies at all levels of scope, but variants of the higher-level patterns add additional embedded patterns that may be documented as a given enterprise architecture (EA) model, which gets described more specifically at the portfolio or system levels. This is similar to the map of the city being part of the map of a region within the map of the world, as shown in Figure 10.2.

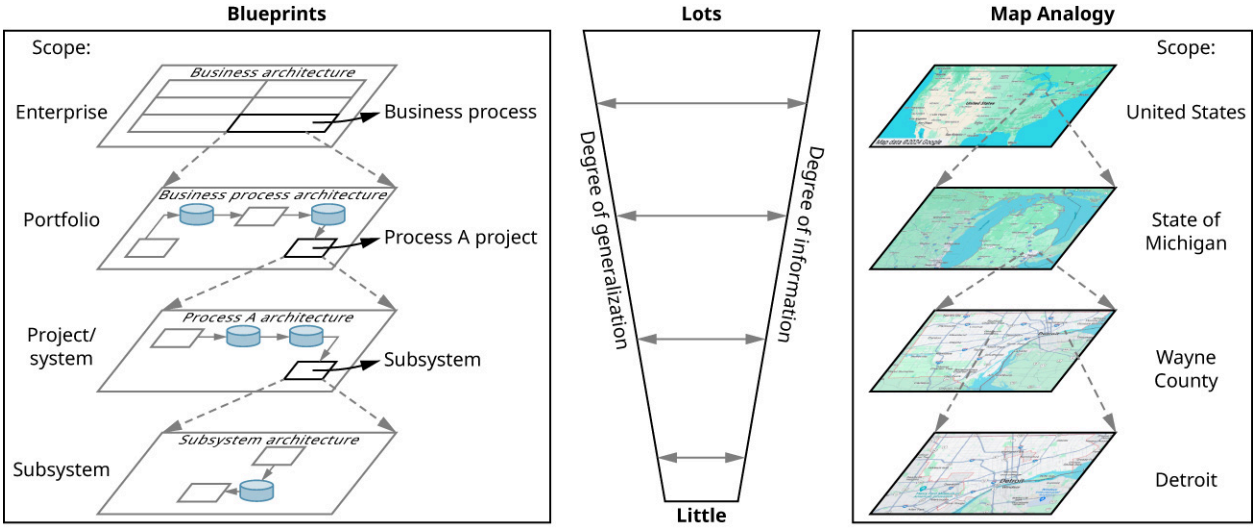


Figure 10.2 A blueprint shows top-down enterprise levels: enterprise, portfolio, system, and subsystem. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Implementation Patterns

In object-oriented programming languages such as Java, a singleton class can have only one object at a time. The purpose of a singleton class is to restrict the number of object creations to only one, which ensures access control to resources. A Java singleton class (Figure 10.3) is a corresponding implementation-centric example of an idiom that implements a singleton design pattern and allows one instance of a Java class to exist, as in Figure 10.4.

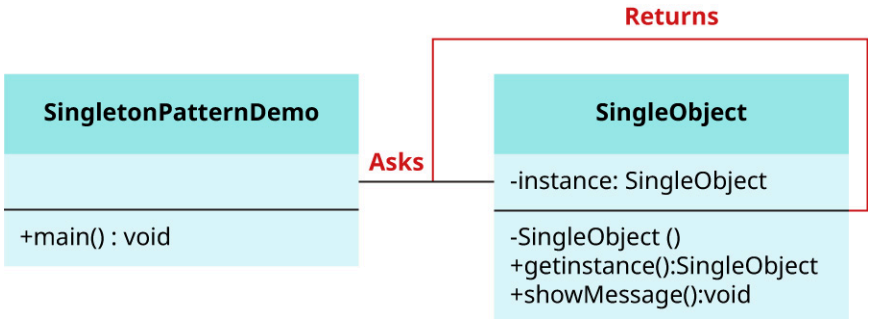


Figure 10.3 The SingletonPatternDemo class accesses the Java singleton class SingleObject using the getInstance() method to return the object. showMessage will print the message. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

```
class SingletonExample {

  private static SingletonExample singleObject;
  private SingletonExample()
  {
  }

  public static SingletonExample getInstance() {
   // write code that allows us to create only one object
  }
}
```

Figure 10.4 A Java singleton class is a corresponding implementation-centric example of an idiom that implements a singleton design pattern and allows one instance of a Java class to exist. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

GLOBAL ISSUES IN TECHNOLOGY

Patterns of Solutions

Problems and the need for solutions vary depending on the region of the world where people are located. For example, economically developing regions like Africa or parts of the Middle East do not have fiber-optic networks as sophisticated as the ones deployed in Europe, where extensive investments in infrastructure have been made. This disparity impacts not only access to the Internet but also the potential for growth in education, technology, and health-care systems. As a result, solutions to these problems are tailored to address the unique needs in these regions. Addressing these needs might involve developing alternative solutions like satellite Internet or mobile broadband networks, which can provide connectivity. However, these solutions need to be cost-effective and scalable to ensure they meet the population's needs.

Pattern Catalogs

From a top-down standpoint (e.g., enterprise, portfolio, or project), architecture may appear in different levels of focus. The architecture areas span the whole enterprise and include various domains of architecture, such as business, information, application, and technology domains. The enterprise domains in the three standpoints are illustrated in Figure 10.5.

Scope	Domain			
	Business	Data	Application	Technical
Enterprise				
Portfolio				
Project				

Abstraction level
Presentation
Conceptual
Logical
Physical

Figure 10.5 The architecture areas span across the whole enterprise and include various domains of architecture, such as business, information, application, and technology domains. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Patterns may be described using a pattern taxonomy and stored in a pattern repository or catalog. A **pattern catalog** is a collection of patterns that are organized according to specific characteristics and according to how the relationships between them are defined. Figure 10.6 illustrates the general structure of a pattern catalog. A pattern catalog allows the bookkeeping of clusters of patterns at various levels of specialization (i.e., analysis and design model patterns versus implementation patterns).

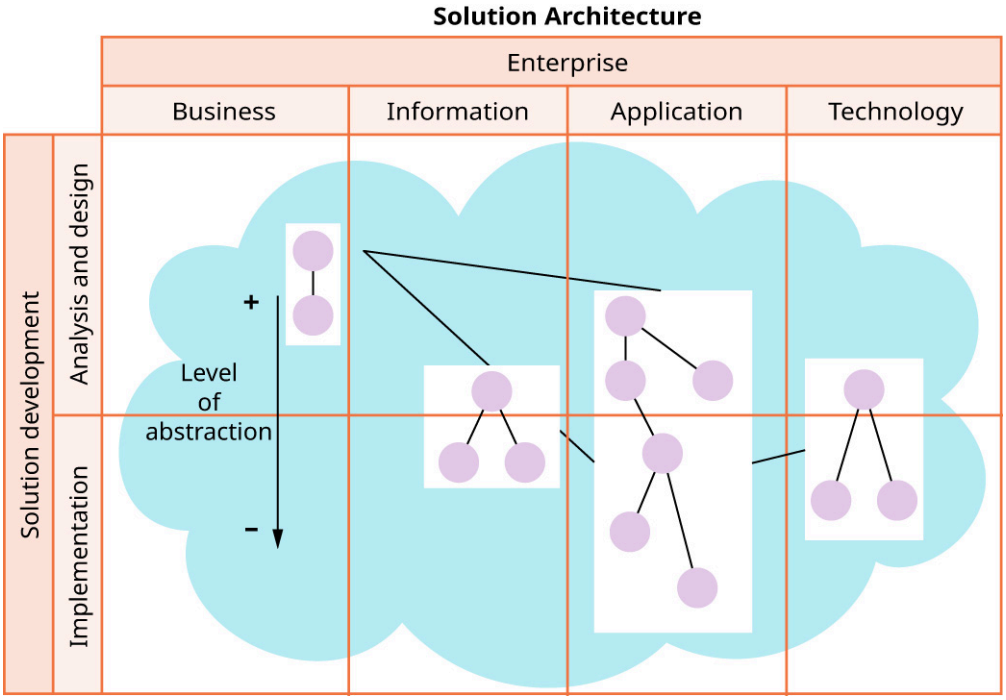


Figure 10.6 The general structure of a pattern catalog allows the bookkeeping of clusters. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

One example of a business architecture pattern that includes a generalization of business service composition and a pattern used to describe the sequencing of business services is **business service orchestration**. If the data does not include data presentation, then business service orchestration provides additional logic to process this kind of data. Another example of a business architecture pattern that focuses on the observed sequence of messages exchanged by peer services when performing a unit of work is called **business service choreography**. Figure 10.7 illustrates how various business architecture design patterns (e.g., exchange, shipment, order, and payment) may be combined via inheritance and composition to create a “buyer-seller” design pattern.

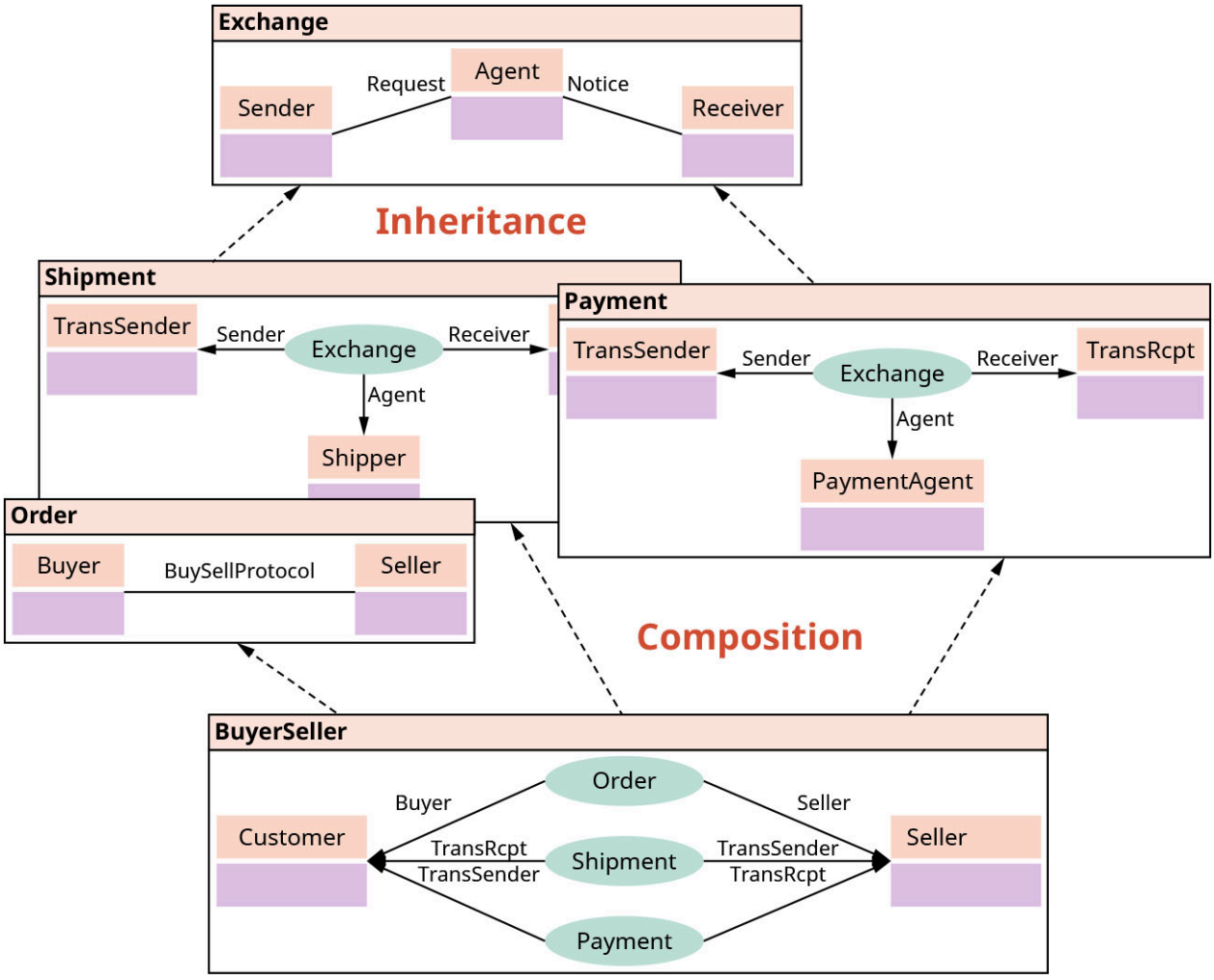


Figure 10.7 Various business architecture design patterns, such as exchange, shipment, order, and payment, are combined via inheritance and composition to create a buyer-seller design pattern. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

LINK TO LEARNING

Read this article comparing orchestration and choreography (https://openstax.org/r/76OrchChoreo) to learn more about the advantages and disadvantages of both of them.

Implementation patterns are typically specific to technology stacks that are selected as part of the specialization of a solution design (i.e., a transition from analysis and design model to implementation architecture). Figure 10.8 illustrates how a pattern catalog is used to select the patterns and eventually create and deploy software engineering products at different project levels.

		Enterprise perspectives			
		Business	Information	Application	Technology
Architecture and solution engineering views	Analysis/ design				
	Implementation				
	Product				
	Deployment				

Figure 10.8 A pattern catalog can be used to select the pattern based on the architecture and solution view for the enterprise perspectives. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Pattern Languages

As we learned in Chapter 9 Software Engineering, a **pattern language** provides a connected view of how to apply one pattern in the presence of another.

CONCEPTS IN PRACTICE

Are All Pattern Problems the Same?

Classifying software patterns is a similar problem to classifying patterns for buildings. For example, you would have to consider each room in a building and describe its role (e.g., kitchen, bedroom) along with specific related patterns, such as appliances, furniture, doors, and locks. As you may recall, design patterns are described using pattern templates. Enterprise pattern catalogs are more complex as they need to consider the degree of specialization of patterns; that is, whether they apply to the enterprise, portfolio, or project level. Therefore, there are many characteristics that must be considered when creating an enterprise pattern catalog, including the domain of applicability, scope, and level of abstraction.

Current State of Patterns Management

A series of "patterns" that create an organization chart for developing software is called **patterns management**. In **software engineering**, the detailed study of engineering to design, develop, and maintain software, Christopher Alexander, a well-known building architect, is regarded as the father of the Pattern Language movement. Eric Gamma, Martin Fowler, and several other contributors are all part of the large community that has been focusing on collecting and documenting software patterns over the past forty years. As a side note, the original wiki, the technology behind Wikipedia, resulted directly from Alexander's work, according to its creator, Ward Cunningham. Christopher Alexander's work has also influenced the development of **Agile software development**, which is an interactive approach to software development in order to deliver value to customers, and Scrum, which is a management framework that helps the development team organize the work to reach a specific goal. A **framework** is an implementation-specific skeletal subsystem for design work. A **subsystem** is a set of collaborating components that perform a given task included in software systems, and it is a separate entity within a software architecture.

Enterprise architects use four architecture domains and architecture views to handle hybrid domains (e.g., an information systems architecture is as an architectural view that spans across the application and data architectures and handles both control and data flows). Therefore, the applicability of patterns may be qualified according to the (combination of) domain(s) of architecture they apply to business architecture patterns or information systems patterns.

INDUSTRY SPOTLIGHT

Patterns and the Health-Care Industry

Architectural styles and related pattern hierarchies are important in every industry today. For example, there are different styles of software architectures in health care that help address various types of business needs. These include mobile health architectures, emotion control management architectures, and remote-surgery architectures. Mobile health architectures enable remote patient monitoring and ensure secure data exchange. Emotion control management systems track emotional patterns in patients. This allows for personalized mental health interventions. Real-time data and robotics allow surgeons to operate on patients who are located far away through remote-surgery architecture. These architectural styles each play a critical role in enhancing health-care delivery and operational efficacy.

A more general view promotes the bookkeeping of architectures at various levels of abstraction as part of an **architecture continuum**, representing a structure composed of building blocks to reuse architecture assets that conform to a pattern language. The corresponding specializations are then organized in a solution (architecture) continuum. In this case, architectures used at the enterprise level in the industry and specific organizations are derived from foundation and common systems architectures, as shown in Figure 10.9, and then realized practically as solution architectures.

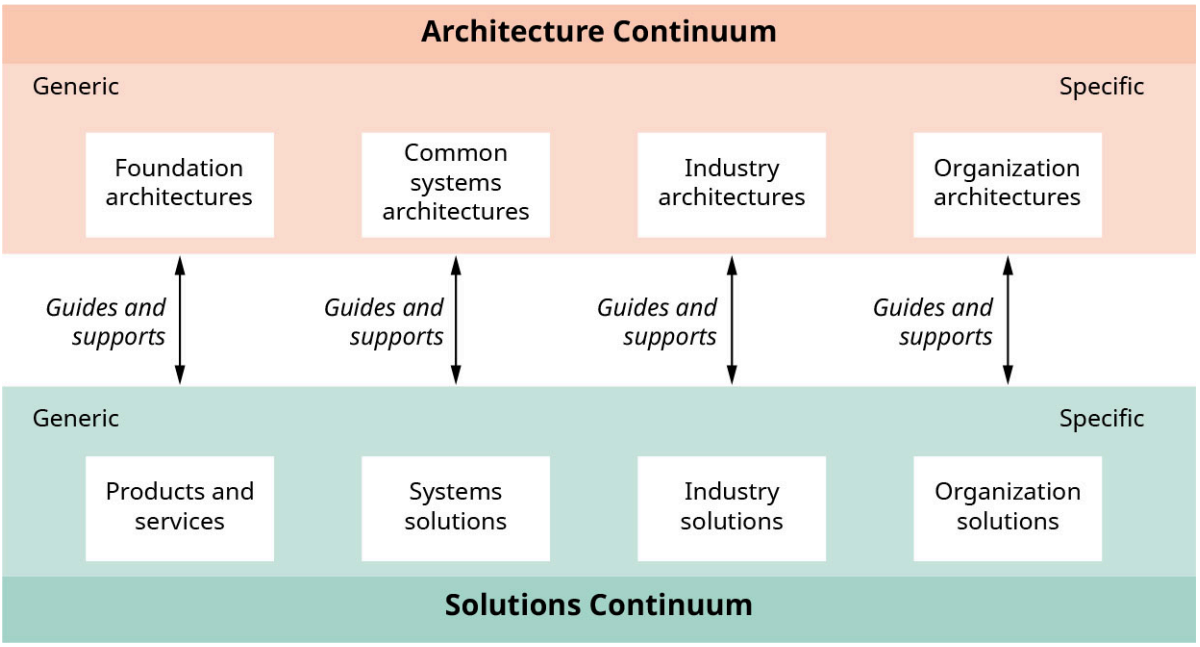


Figure 10.9 The Open Group Architecture Framework (TOGAF) foundation architecture contacts common systems architectures, which contacts industry and organization architectures. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

10.2 Enterprise Architecture Management Frameworks

Learning Objectives

By the end of this section, you will be able to:

- Understand how enterprise architecture management helps align business and technology strategies
- Explain what enterprise architecture frameworks are used for
- Relate to the specifics of the TOGAF enterprise architecture framework
- Build a strategic adoption road map
- Apply process frameworks and blueprinting templates in EAM

Enterprise architecture (EA) is a comprehensive, well-defined approach to a business plan that utilizes information technology to meet the objectives of the business vision by aligning business and technology strategies. Information technology includes software, hardware, communication, data, and people. Managing EA will involve using **enterprise architecture management (EAM)**, which helps guide the adoption of technology stacks and corresponding implementation frameworks. EAM is the practice of managing an enterprise's business and IT architecture. It involves planning, designing, implementing, and governing the enterprise architecture to support the organization's goals.

Enterprise Architecture

There are many activities/business processes associated with an enterprise; however, its focus is on activities that allow it to meet current and future objectives. A **business process** is a series of steps a group of stakeholders performs to achieve an enterprise concrete goal or set of objectives. Current objectives reflect an enterprise's mission and short-term strategies, while future objectives reflect their vision and long-term strategies. To ensure successful development and execution of these strategies in either the short or long term, an enterprise will rely on EA as shown in Figure 10.10.



Figure 10.10 The goal of EAM is to provide enterprise capabilities to keep business solutions aligned with the enterprise strategy. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

As illustrated in Figure 10.11, EA follows the requirements of a preestablished operating model, helps establish meaningful strategic boundaries, supports the management of core business capabilities that enable the development of a business platform for the execution of routine business tasks, and allows people to concentrate on improving the business rather than just purely running it. A foundation for execution is a combination of IT infrastructure (i.e., hardware, software, communication, and data) and digitized processes that automate the company's core business capabilities. The goal of the foundation for execution is to develop alignment between business and IT strategy. Creating and maintaining a foundation for execution relies on the company's operating model, strategy adoption road map, and a view of its reference architecture. A **road map** is used to guide an organization with planning and achieving business goals over time through technology.

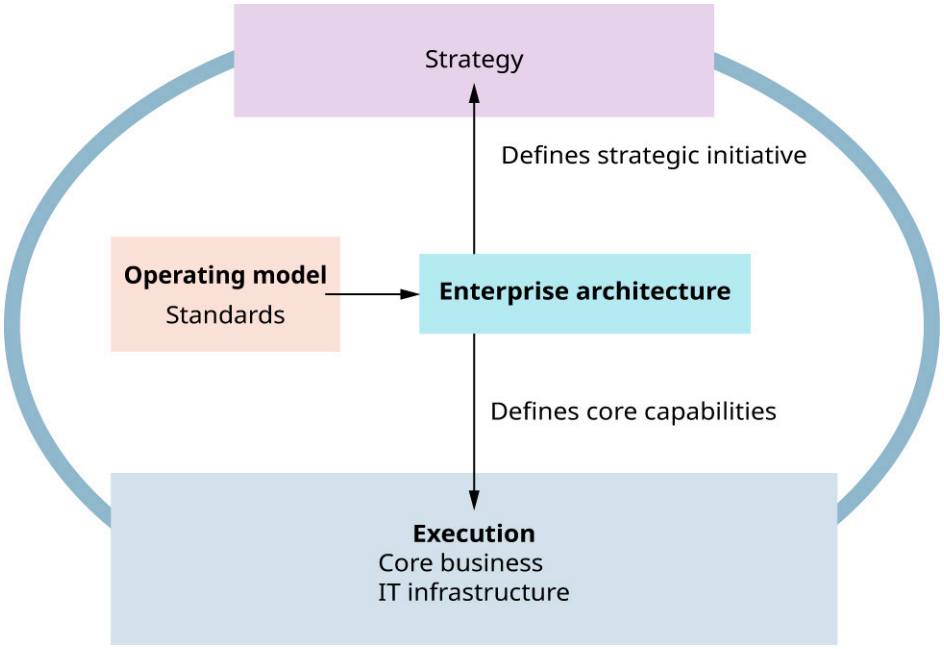


Figure 10.11 This figure depicts the EA foundation design for execution. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Figure 10.12 illustrates various operating models known as diversification, coordination, replication, and unification. These models correspond to variations in the way business processes are standardized and integrated.

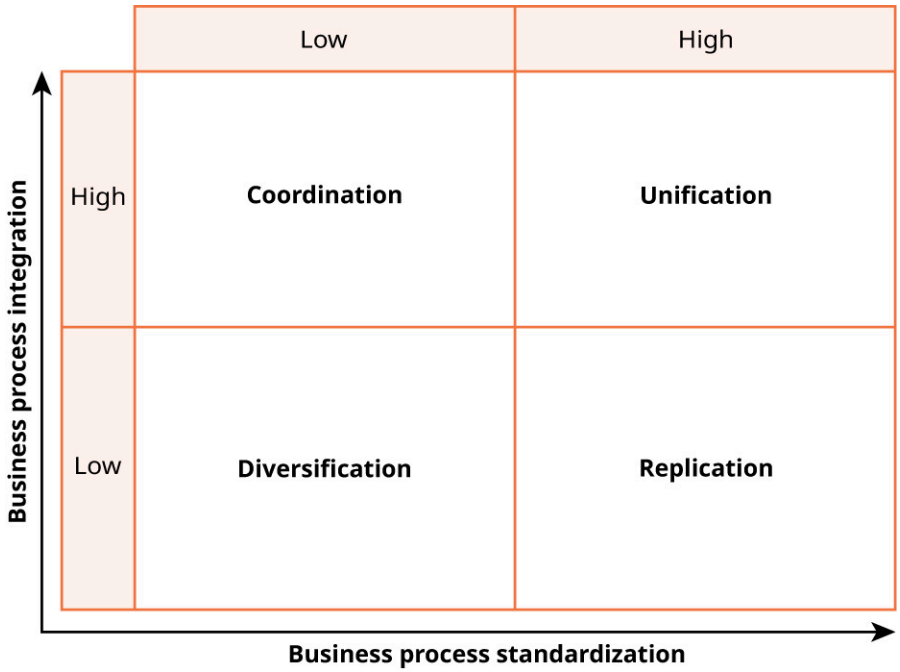


Figure 10.12 Operating models known as diversification, coordination, replication, and unification correspond to the way business processes are standardized and integrated. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The enterprise IT strategy is based on a collective set of principles. These principles form a consistent framework for technology decision-making and reflect a level of consensus among key stakeholder technology groups. The technology groups include senior leadership, the architecture council, and leading architects. Figure 10.13 shows the principles are intended to guide decision-making at all levels and are organized as a business, information, application, and infrastructure.

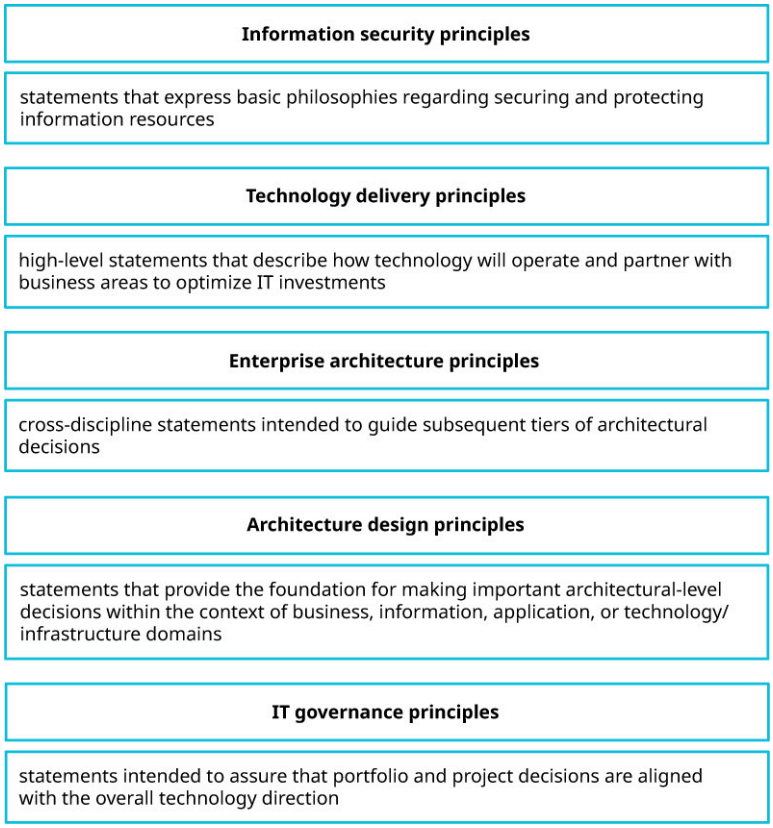


Figure 10.13 The organizing framework includes information security principles, technology delivery principles, enterprise architecture principles, architectural design principles, and IT governance principles. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Figure 10.14 summarizes typical enterprise capabilities within various business units such as manufacturing, corporate, and technology.

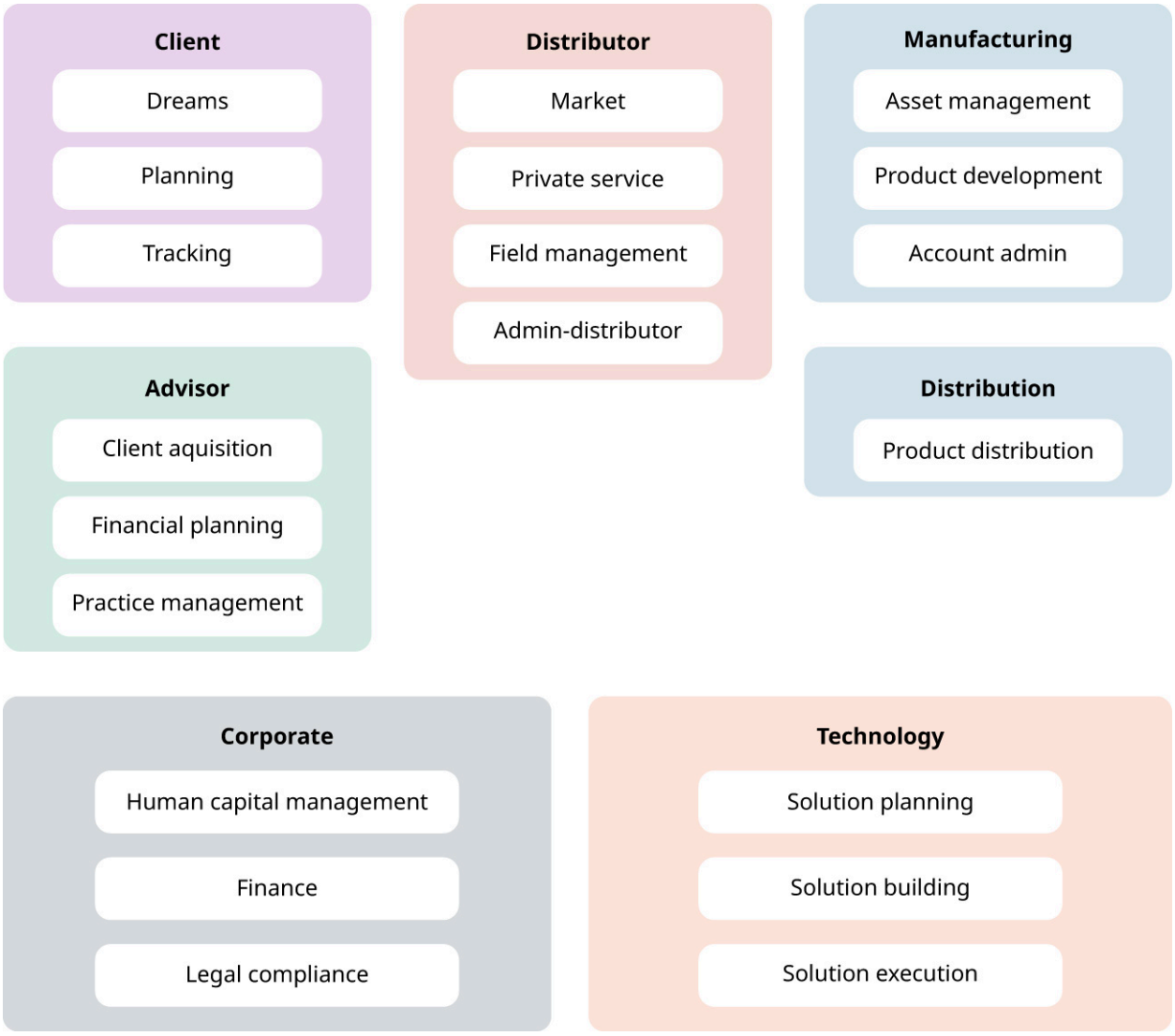


Figure 10.14 Typical enterprise capabilities within various business units such as manufacturing, corporate, and technology are shown. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Investment in technology may have possible impacts, such as goals, desired expectations, and metrics. In the goals, we estimate many factors, such as the required time, the deliverables information, the cost of quality, and the outputs. In the desired expectation, we evaluate the project (e.g., right project, right cost, right timing, and right resources). In the metrics, we measure the investment demand, employee satisfaction, and quality of service (i.e., availability and cost). Figure 10.15 illustrates the possible impact of investment initiatives in technology capabilities as they are arranged in the plan and solution section, built into the solutions section, and deployed into products delivered to customers. The process will start with planning solutions and investments, then building solutions, and, finally, running solutions.

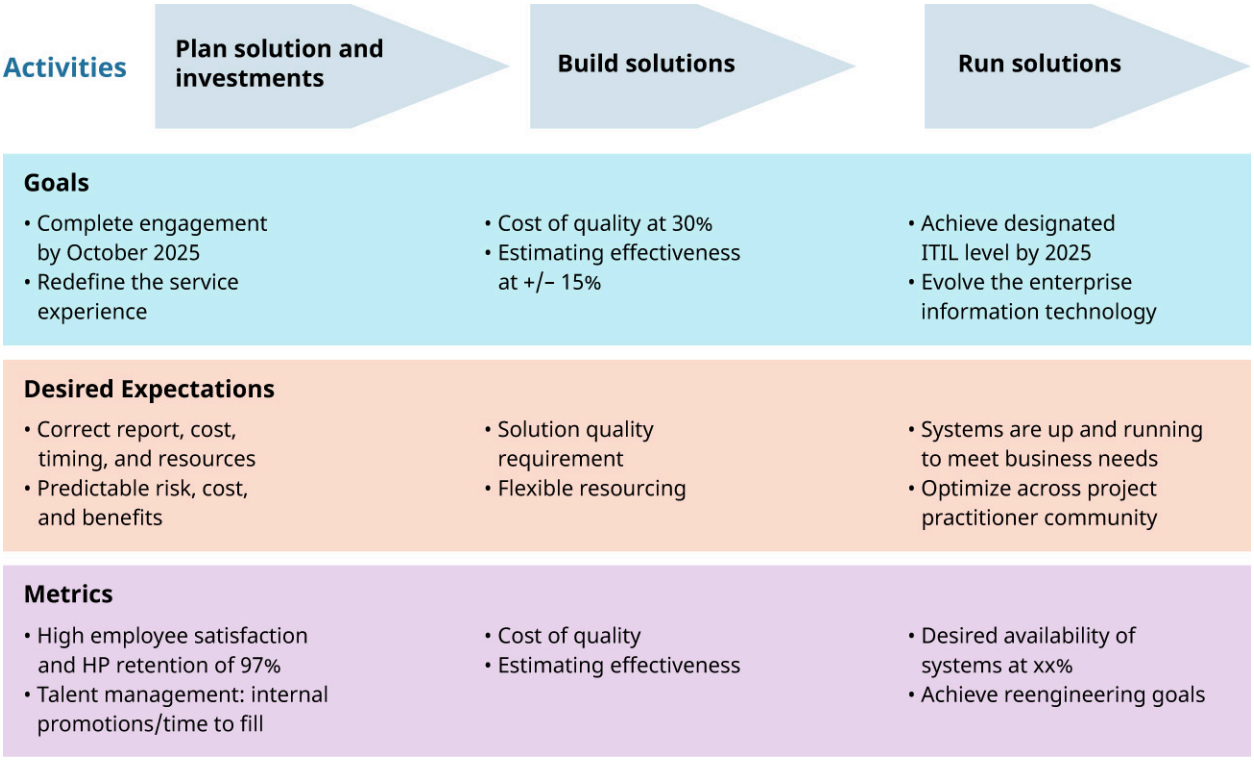


Figure 10.15 There are many possible impacts of investment initiatives in technology capabilities. Each section lists the assigned capabilities and/or impacts. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Business and technology executives are responsible for ensuring that an IT project achieves division-wide and company-wide objectives by engaging with the enterprise architects. EA is an evolving practice that takes time, investment, and patience. It should provide a strategic framework for organizations to handle technological disruptions by aligning IT systems with business goals while helping decision-makers implement changes that drive innovation, resilience, and efficiency to ensure short-term initiatives and long-term projects can contribute to the success of the company.

A leading health-care organization aimed to reduce operational costs while streamlining patient care. They used EAs to identify the need for a unified data system. The system needed to integrate patient records, billing, and telemedicine services. The EA team presented recommendations, including adopting cloud-based platforms and automated workflows. This helped the organization handle business disruptions, minimize data silos, and improve service delivery. The project aligned with division-wide objectives, showcasing how EA practices can drive strategic transformation in response to industry challenges.

Aligning Business and Technology Strategies

Aligning business and technology strategies is typically difficult on an ongoing basis because of the lack of alignment with enterprise-driven initiatives. Enterprise architecture helps align business and technology strategies, as illustrated in Figure 10.16. This alignment is a continuous process and is achieved through the business-IT alignment life cycle.

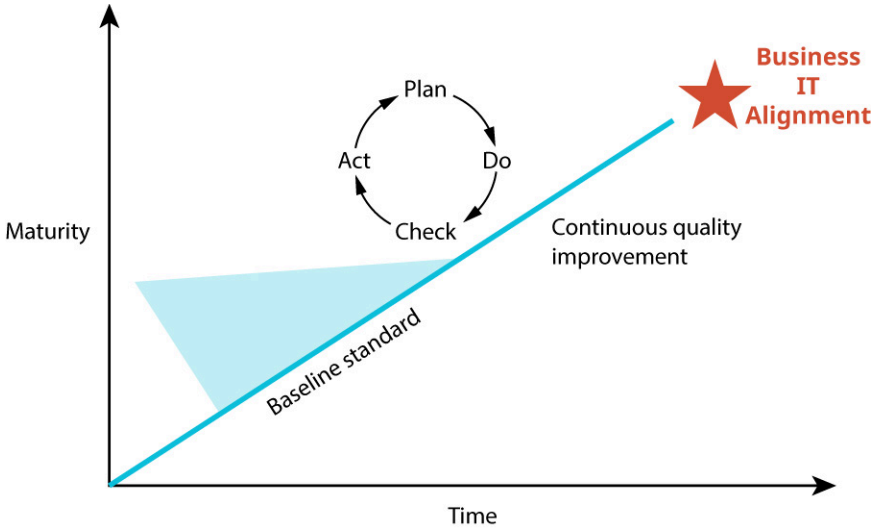


Figure 10.16 The business-IT alignment life cycle is a continuous improvement process that helps align business and technology strategies. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The business IT alignment life cycle helps organizations align their business goals with IT strategies through different stages. The first stage is initiation, which involves checking the problem facing the enterprise or available opportunities. The second stage is strategy, which involves designing solutions. The third stage is planning, which involves planning how to implement the solution. The fourth stage is execution, and it involves implementing the solution. The last stage is assessment, which entails evaluating the solution.

The enterprise team should be careful when developing business IT alignment because enterprise business solutions may lack alignment with the enterprise-driven initiative. In other words, business solutions should take into account the architectural guidelines that are set forth at the enterprise level and not just reinvent the wheel. Figure 10.17 shows the four symptoms that may lead to the lack of alignment.

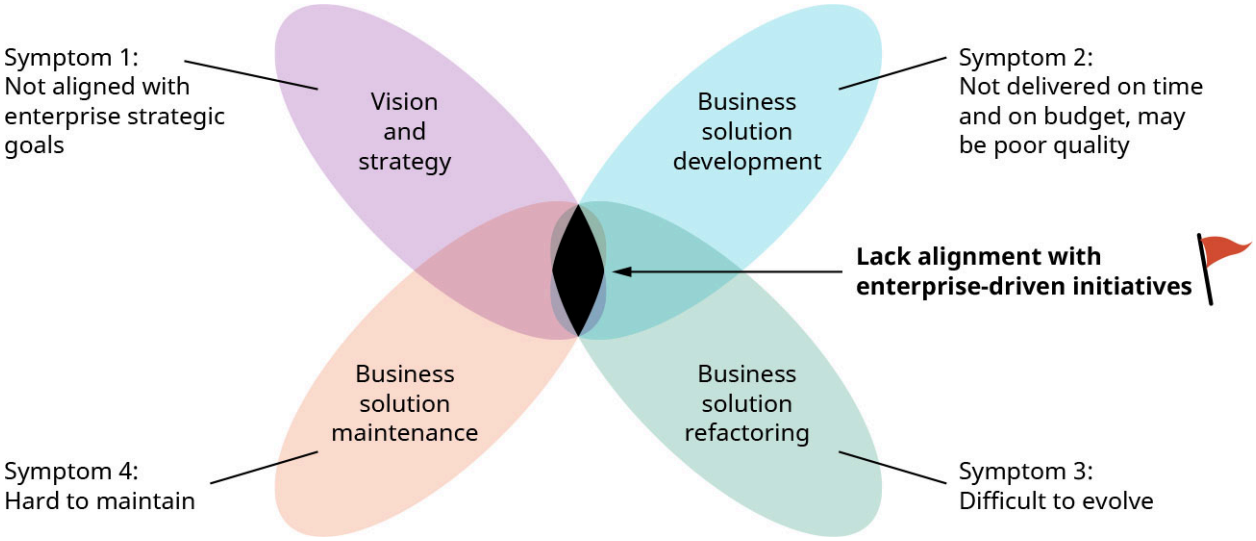


Figure 10.17 When the four symptoms do not overlap, enterprise business solutions lack alignment with enterprise-driven initiatives. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The first symptom appears when the created solution is not aligned with the enterprise goal. The second one appears when the business solutions are not delivered on time and on budget or if the outcome doesn't reach the expected quality. The third one appears when it's difficult to maintain the system or the solution. The fourth symptom appears when it's hard to evolve the business solutions.

GLOBAL ISSUES IN TECHNOLOGY

EAM across the Globe

All the EAF frameworks in existence were developed in the United States and Europe. The most widely recognized EAFs are TOGAF (The Open Group Architecture Framework) and the Zachman Framework. EAFs are not typically designed to be adaptable to different organizational contexts and different cultures. Therefore, there is uncertainty about how effective these frameworks are in diverse cultural contexts, such as in Asia and Africa, where decision-making processes and organizational structures may differ from those in the United States and Europe.

In addition to the alignment, enterprise architecture management uses enablers to provide enterprise capabilities to keep business solutions aligned with the enterprise strategy. An **enabler** supports the activities needed to extend the architectural runway to provide future business functionality. Figure 10.18 shows five different kinds of enablers, which are listed in Table 10.1.

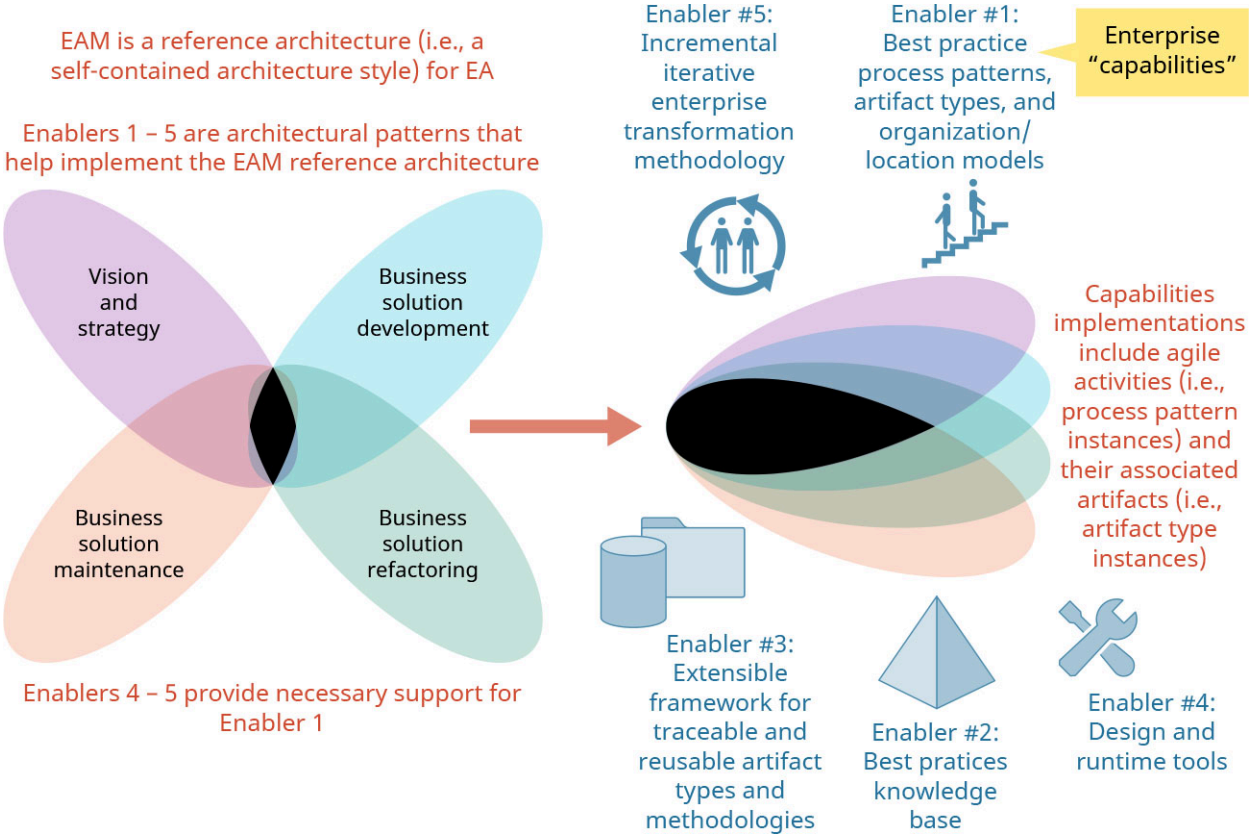


Figure 10.18 Enterprise architecture management provides enterprise capabilities, known as enablers, to keep business solutions aligned with the enterprise strategy. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Type	Description
Best practice process patterns, artifact types, and organization models	Subsumes specific organization and location models within the enterprise along with process patterns and artifact types to guide the implementation of business solutions
Best practice knowledge base	Extensible methodology based on business solution patterns and extensible knowledge foundation; best practices for ongoing strategies and business solution development
Extensible framework for reusable artifact types and methodology	Selects subsets of process patterns and artifact types, customizes implementation of process patterns that fit the enterprise environment, and leverages existing enterprise approaches
Design and runtime tools provide necessary support for enabler #1	Selects best practice processes, approach, tools, and artifacts to help implement reference architectures
Incremental iterative enterprise transformation methodology	Suggests the use of an incremental iterative (CMMI-compliant) enterprise transformation methodology that improves the maturity of the enterprise, or its ability to keep business and IT aligned over time, as illustrated in Figure 10.19

Table 10.1 Types of Enablers

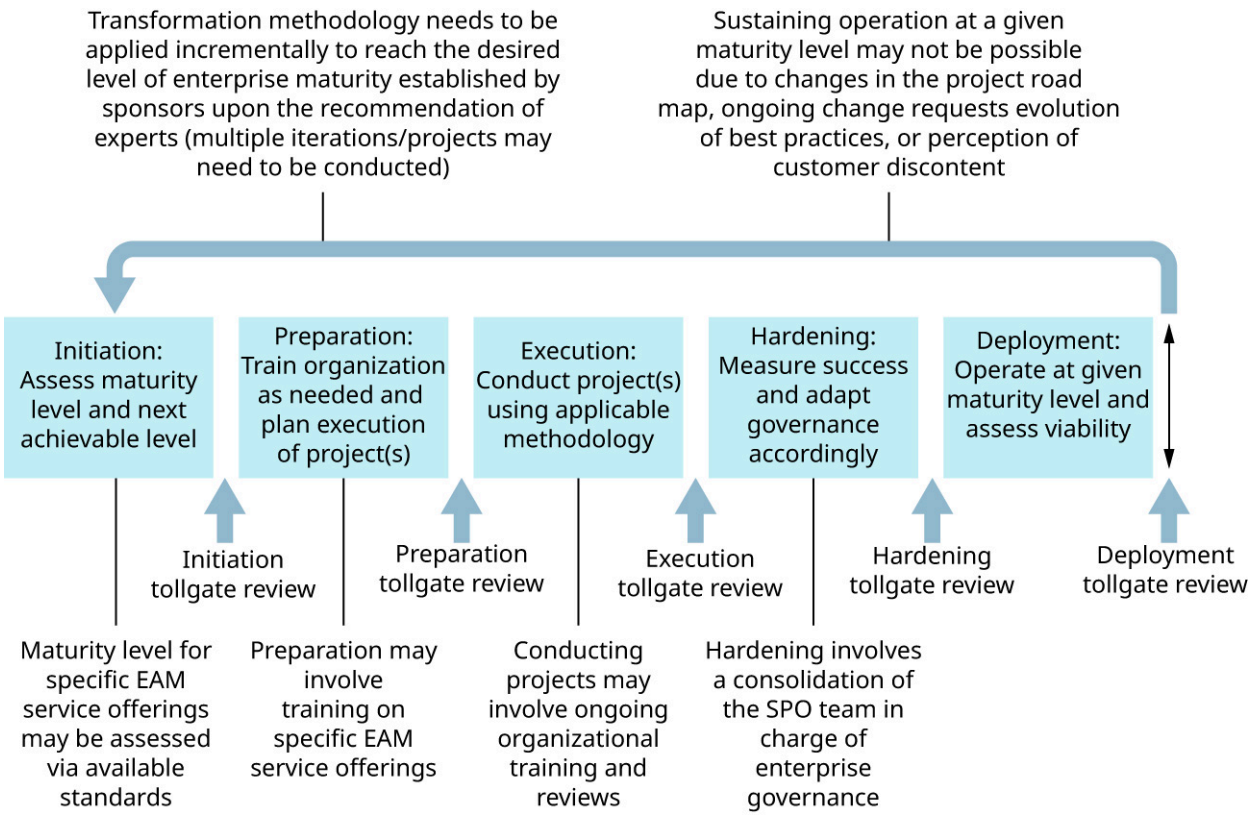


Figure 10.19 This figure shows how the generic EAM Awareness-Desire-Knowledge-Ability-Reinforcement change management methodology applies in the context of a Business Pattern-Driven Modeling process pattern. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

THINK IT THROUGH

Aligning Strategies

Aligning business and technology strategies involves ensuring that goals and objectives of business and technology are supportive. How is it possible to align business and technology strategies?

Enterprise Architecture Frameworks

EAM is a management practice that establishes, maintains, and uses a coherent set of guidelines, architecture principles, and governance regimes that provide direction and practical help in the design and development of an enterprise's architecture to achieve its vision and strategy. EAM brings the highly distributed knowledge of all experts to the table and allows every participant to provide such knowledge and input in terms that best fit the experience and expectations of the contributing stakeholders.

INDUSTRY SPOTLIGHT

EAM in Health Care

EAM is important in every industry today. It plays a vital role in health care to handle unexpected challenges, such as those seen during the COVID-19 pandemic. By providing a framework that aligned business and IT strategies, EAM allowed hospitals and other medical organizations to quickly adapt to the crisis. EAM was used to manage supply chain disruptions, scale telemedicine services, and rapidly build respirators to ensure that health-care infrastructures were agile, resilient, and capable of maintaining high-

quality care in the face of surprises.

As EA and EAM regard the enterprise as a large and complex system or system of systems,[3] it is necessary to manage the scale and complexity of this system. To address this requirement, an **enterprise architecture framework (EAF)** provides tools and approaches that ensure that enterprise solutions are in alignment with the evolving vision and strategy of the organizations that use these solutions to operate and conduct day-to-day business. EAFs provide structured guidance that is divided into three main areas:

1. Descriptions of architecture: how to document the enterprise as a system
2. Methods for designing architecture: overarching enterprise architecture process composed of phases and broken into lower-level processes composed of finer-grained activities
3. Organization of architects: guidance on the team structure and the governance of the team, including the skills, experience, and training needed

To describe architectures, EAFs employ flexible data models built upon metamodels that provide evolving outlines for capabilities and relationships. At a high level, EAFs provide foundational principles, an organizing framework, a comprehensive and consistent method, and a set of governing processes and structures, as illustrated in Figure 10.20.

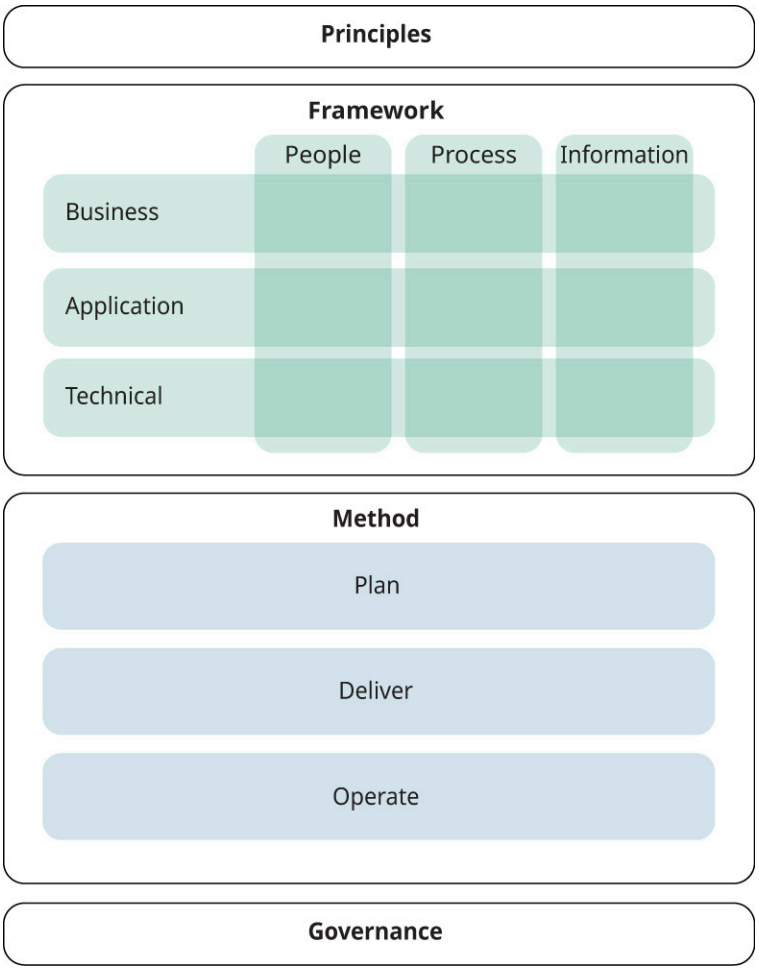


Figure 10.20 This high-level view of EAF reference architecture shows the main components: principles, framework, method, and governance. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

3 A single house is much like a single system—it has various types of architecture within it, and it exists within ever-larger ecosystems.

The reference architecture is built upon principles. A **principle** forms the foundational rules that guide its design and implementation. The reference architecture uses architectural terms as well as numerous principles, policies, and guidelines to govern the architecture. The framework serves as the organizing structure and outlines architectural domains and disciplines to ensure separation of concerns and alignment between business goals and IT. The **method** consists of defined, repeatable processes that ensure a consistent and controlled execution of the reference architecture. The processes and organizational structures that ensure adherence to the reference architecture are considered **governance**.

TECHNOLOGY IN EVERYDAY LIFE

EAMs and EAFs in Real Life

EAMs are found in common technologies and are used to ensure systems operate smoothly as user needs change. For example, home automation systems are often expanded, with the user requiring the ability to add new devices, such as a smart refrigerator or thermostat. However, the system needs to maintain compatibility with the existing devices it is connected to. EAFs provide structured guidelines to manage these changes, ensuring flexibility and scalability.

How can EAMs ensure seamless integration of new smart devices into your home system? How do EAFs help maintain flexibility in personal technology ecosystems?

As the practice of EA evolves, the supporting EAF frameworks that enable communities and enterprises to apply EA also evolve. Indeed, as solution complexity increases, the ingenuity needed to master it must continuously improve. In fact, architecture strategies keep playing an enormous and ever-increasing role in determining whether an enterprise is successful. The next generation of EAFs is focused on enabling the creation of dynamic ecosystems and related architectures that provide value for all business participants via systems of insight that integrate and facilitate decisions across systems of record (i.e., focus on costs), systems of operations/automation (i.e., focus on operational expense [Opex]), systems of design (i.e., focus on innovation), and systems of engagement (i.e., focus on customer experience). Figure 10.21 shows the evolution of EA frameworks that has taken place over the years.

IT/Business Drivers	EA Drivers
Optimize company assets	IT fuels business innovation
Business process flow	Complexity
Individual productivity	Distribution
Automatic transaction processing	Costs

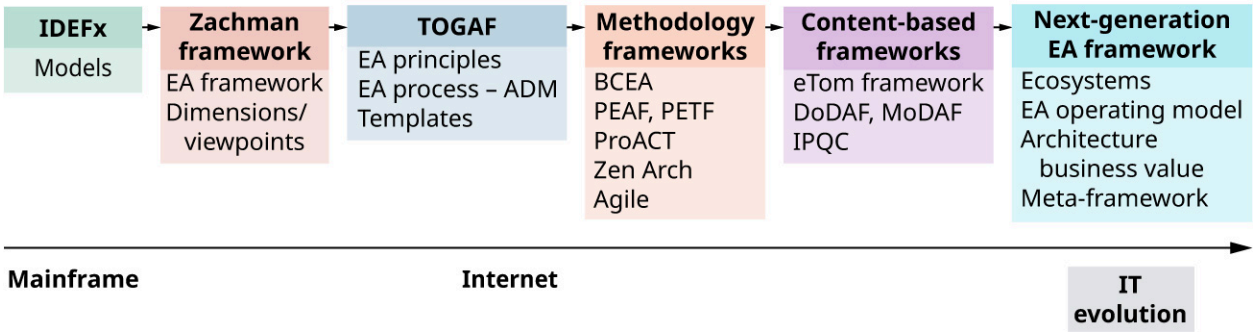


Figure 10.21 This chart shows the relationship representation of IT/business drivers and IT evolution over time. The evolution ranges from the mainframe to smart computing. The next-generation EA framework will optimize human, physical, and environmental assets. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The architecture subsumed by this new breed of ecosystems is illustrated in Figure 10.22. It relies on various layers that can be modeled (using next-generation EAFs) as follows:

- Trust models ensure that all participants within the ecosystem share and agree on additional values rather than just operate only for financial profits (e.g., respect privacy, guarantee security, provide customer-oriented value and social values, improve the level of quality) as well as shared outcomes, including externalities such as pollution or regulation.
- Business models ensure that the values identified in trust models are shared and business models can be established that split the margin and revenue between ecosystem participants.
- Orchestration models optimize the split of responsibilities and orchestrate services to deliver the best values for customers at the right price/quality (e.g., the links navigation digital experience journey or the knowledge flows journey maps to services and processes at the ecosystem level). Shared services models include supporting data and application services.
- Business networks enable the sharing of processes and data; for example, cloud applications shared between partners in digital business networks via a mix of shared and multitenancy architectures.

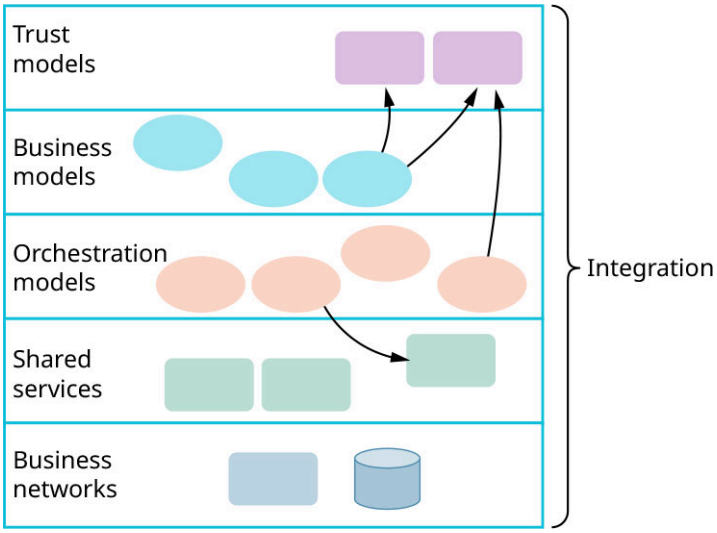


Figure 10.22 Using next-generation EAFs, various layers can be modeled as trust models, business models, orchestration models, shared services models, and business networks. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

To provide adequate support, next-generation EAFs must supply an extensive set of generic capabilities as part of an underlying meta-framework. A **meta-framework** describes the framework in more detail, as illustrated in Figure 10.23. Clearly, these capabilities go beyond modeling and related features and must be spanned across EAF requirements for IT automation, IT governance, and IT context management. **IT automation** is the process of creating systems to reduce manual intervention. **IT governance** is the process that ensures the effective and efficient use of IT in enabling an organization to achieve its goals. **IT context management** is a dynamic IT process that uses data in one application to point to data resident in a separate application.

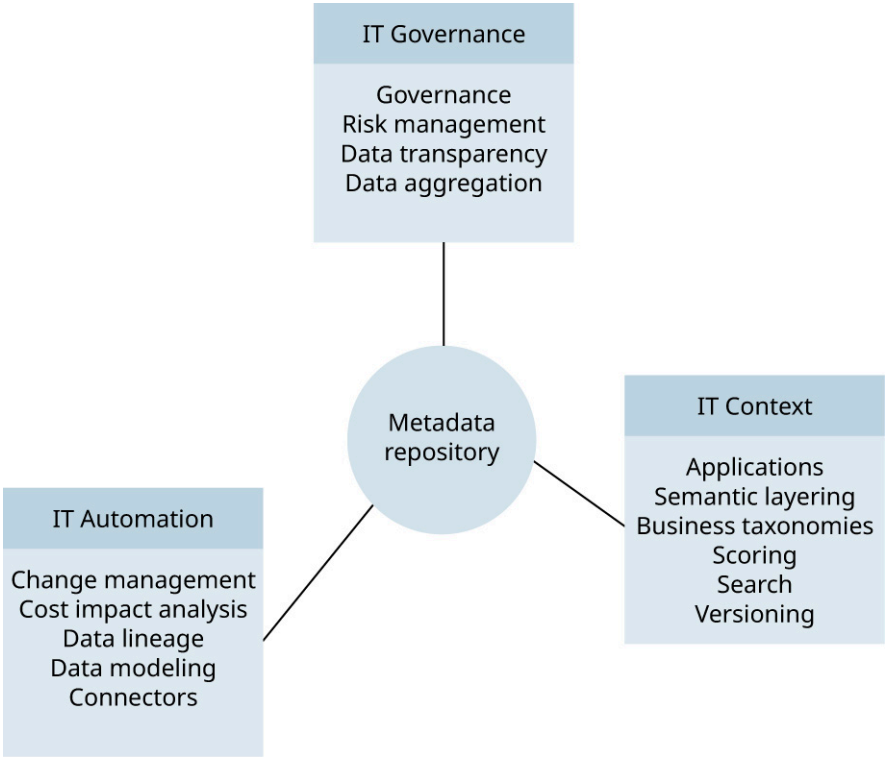


Figure 10.23 EAF requirements for IT automation, IT governance, and IT context management are shown. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

There are many examples of EAFs, such as TOGAF, Gartner Enterprise Architecture, C4ISR, CORBA, Federal Enterprise Architecture, Zachman Framework, and many more. Each one of the EAFs follows a specific format

and uses a different specification.

Table 10.2 shows examples of EAFs and their specifications.

Enterprise Architecture Framework	Specifications
TOGAF	• The most popular EAF today[4] • Uses the Architecture Development Method (ADM) with four interlinked domains (business, application, data, technology) and provides a set of views for each domain
Gartner Enterprise Architecture	• Identifies and analyzes the execution of change toward desired business vision and outcomes
C4ISR—Command, Control, Computers, Communications, Intelligence, Surveillance, and Reconnaissance	• Replaced Technical Architecture Framework for Information Management (it was renamed as DODAF in 2003) • Has minimal focus on methodology unlike ADM • Focuses on operational, system, and technical views
CORBA—Common Object Request Broker Architecture	• Object Request Broker–based architecture • Application architecture
Enterprise Architecture Planning	• Method for planning development of business, data, applications, and technology • Analogous to TOGAF but does not have TRM and SIB
Federal Enterprise Architecture Practical Guide	• End-to-end process to manage enterprise architecture • Aligned to TOGAF life cycle
Federal Enterprise Architecture Framework	• Provides guidance on structuring enterprise architecture • Organizes architecture into five reference models to provide a standard methodology • Analogous TOGAF views
ISO/IEC TR 14252 (IEEE Std. 1003.0)	• Withdrawn • TOGAF is more detailed
ISO RM-ODP	• Formal description techniques for architecture specifications—distributed process and heterogeneous environments
SPIRIT Platform Blueprint	• Referenced within TOGAF

Table 10.2 Enterprise Architecture Frameworks and Specifications

4 www.opengroup.org/TOGAF

Enterprise Architecture Framework	Specifications
Technical Architecture Framework for Information Management	• Basis on which TOGAF is built
Zachman Framework	• More detailed than TOGAF in capturing view points and views, which can be filled in using ADM; does not specify any method • Security and manageability are not explicitly specified in Zachman[5]

Table 10.2 Enterprise Architecture Frameworks and Specifications

The Open Group Architecture Framework

The Open Group Architecture Framework (TOGAF) is an EA methodology and framework used by leading organizations to improve business efficiency. TOGAF focuses on elements of consistent methods, standards, and communication. TOGAF enables organizations to pursue a systematic approach toward the development process in order to reduce errors, manage timelines, stay within planned budget and scope elements, and have efficient processes to produce quality results. TOGAF helps practitioners avoid being locked into proprietary methods, utilize resources more efficiently and effectively, and realize a greater return on investment.

In 1995, the original development of TOGAF Version 1 was published as a reference model for enterprise architecture, offering understanding into the U.S. Defense Department's (DoD) own technical infrastructure, including how it's structured, maintained, and configured to align with specific requirements (Figure 10.24). As of today, the TOGAF is a detailed method and a set of supporting tools for developing an enterprise architecture. It may be used freely by any organization desiring to develop an enterprise architecture for use within that organization.

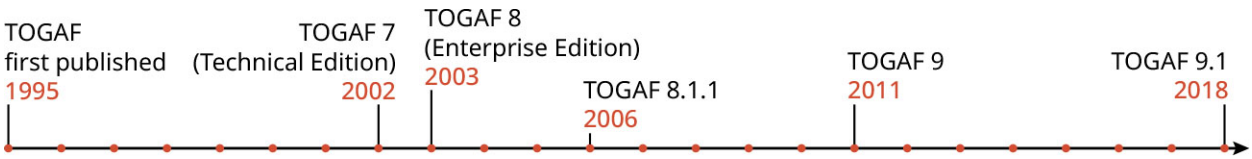


Figure 10.24 This timeline shows the development of TOGAF over time. The first version was in 1995, and the most recent version was in 2018. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

TOGAF is the widely used framework to establish and define enterprise architecture; it presents an approach to planning, designing, implementing, and governing the enterprise-level information technology architecture of the organization. It mainly focuses on four modular structures, which include application, business, technology, and data. The main purpose of defining the TOGAF framework was to have an architectural model that can be replicated with few errors in each progressing phase. Establishing a common language and understandability mechanism bridges the gap between IT and the business by bringing clarity to the information, establishing methodology, and having practical implementation guides.

The TOGAF standard includes the **Architecture Development Method (ADM)**, which is a detailed step-by-step process for developing or changing an enterprise architecture. It also allows for changing a content framework to help drive greater consistency in the outputs that are created when using the ADM (Figure 10.25). The TOGAF content framework provides a detailed model of architectural work products. Common system architectures are used by many enterprises. Industry architectures are industry-specific, while organizational

5 www.zifa.com

architectures are company-specific.

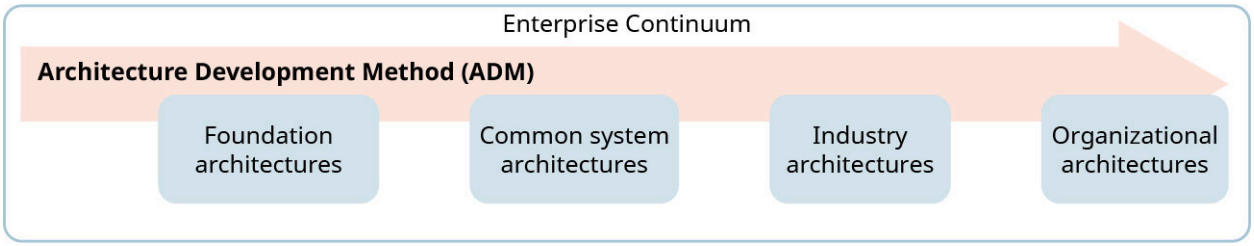


Figure 10.25 The most generic component of the continuum is the foundation architecture, which is usable by any IT organization. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

LINK TO LEARNING

ADM defines the TOGAF approach (https://openstax.org/r/76TOGAF) for establishing processes linked with enterprise architecture. It provides a recursive and tested process development business architecture; every phase of the ADM is iterative in nature to develop an enterprise-wide architecture. The ADM can be adapted and customized to a specific organizational need to help inform the business's approach to information architecture. ADM helps businesses develop processes that involve multiple checkpoints and firmly establish requirements to repeat the process with minimal errors.

The categories of patterns are specification patterns, vision patterns, process patterns, governance patterns, migration patterns, usability patterns, information patterns, business patterns, and interoperability patterns. The phases suggested by the ADM are as follows:

- Preliminary Phase. This phase defines enterprise principles, IT principles, and architecture principles (i.e., how do we do architecture?). It is not considered a part of the ADM cycle, as it may be revisited at any point throughout the cycle. This phase involves the organization and governance of the architecture, its general principles, methods, tools, and the architecture repository, and is the jumping-off point during which an organization starts an ADM cycle. The outputs of this phase are framework definition, architecture principles, and reference to an organization's principles and goals.
- Phase A: Architecture Vision. This is the first phase of the ADM cycle and begins with the receipt of a request for architecture work within a sponsoring organization. During this phase, an organization establishes the project, identifies business goals, reviews architecture and business principles, defines the scope and constraints of an architecture effort based on an assessment of resource and competence availability, and identifies stakeholders and business requirements. A vision is developed regarding the project's organization, orientation, road map, baseline architecture, and risks. In the end, a business has a document statement of architecture work to submit for approval. Phase A is where the organization answers the questions of where we are going, how we are getting there, and with whom.
- Phase B: Business Architecture. During this phase, an organization describes the baseline business architecture that currently exists and the target business architecture that they will work to implement. The next step is creating business architecture models, organization structure, business goals and objectives, business functions, business services, business processes, business rules, correlation of organization and function, and trade-off analysis reports. To accomplish this, gap analysis and modeling must be performed for and between the baseline and target business architectures. The gap analysis includes people, processes, tools, information, measurement, financial, and facilities. Such modeling may include a variety of tools, including activity models, use-case models, class models, node connectivity diagrams, and information exchange matrices. Figure 10.26 illustrates common activities among several phases.



Figure 10.26 Phases B, C, and D of TOGAF AMD represent multiple activities that include describing the baseline architecture, describing the target architecture, measuring gaps, evaluating impact, and drafting the road map. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

- Phase C: Information System Architecture. Information system architecture is composed of data architecture and application architecture subphases. The design framework that structures how data is collected, stored, managed, and utilized within a system is called **data architecture**. The patterns used to design and implement an application are considered its application architecture. The objective of this phase is to develop target architectures covering either or both subphases, depending on the project scope. The business processes supported in this phase are those that are supported by IT, as well as the interfaces of those processes with non-IT-related processes.
  Data architecture reviews and selects principles, reference models, viewpoints, and tools. It creates data architecture models for each viewpoint, such as the C4ISR Architecture Framework and conceptual data model map to business architecture. Then, it performs trade-off analysis, completeness, and conformance. After that, it selects data architecture building blocks, conducts check point review of the architecture model, reviews qualitative criteria, completes the data architecture, conducts checkpoint/impact analysis, performs gap analysis, and creates reports.
  Application architecture subphases develop a baseline applications architecture description; review and validate application principles by selecting reference models, viewpoints, and tools; create architecture models for each viewpoint; identify candidate applications (i.e., business architecture and data architecture); conduct checkpoint review; review the qualitative criteria; complete the applications architecture; perform gap analysis; and create report.
- Phase D: Technology Architecture. The purpose of this phase is to develop a technology architecture that defines the platforms and execution environments on which the organization's applications run and the data sources are hosted. The key steps involved in this phase are:
  1. Develop the baseline technology architecture description to the extent required to support the target technology architecture.
  2. Develop the target technology architecture by considering different architecture reference models, viewpoints, and tools. Then, create an architecture model of building blocks, select the services portfolio required per building block, confirm that business goals and objectives are met, choose the criteria for specification selection, complete the architecture definition, and conduct gap analysis.
- Phase E: Opportunities and Solutions. Phase E identifies the key business drivers, the change parameters, the major phases required, and the projects that are undertaken to move the organization to the target environment. Phase E reviews gap analysis (i.e., studying the gap between the current system and the future system), performs architecture assessment, and identifies software packages for projects. This phase also involves the study of the technical, organizational, and financial requirements and constraints of the project.
- Phase F: Migration Planning. During this phase, the various projects required to implement the architecture are sorted into priority order based on dependencies and the cost-benefit assessment of the various projects. Migration schedules (i.e., outline the timeline, tasks, and resources needed), risk assessment (i.e., identifying and analyzing potential risks), project goals (i.e., defines the goals and outcomes), project constitutions (i.e., outlines the main principles of the system), implementation road map (i.e., steps to follow in the implementation), and project organizations are established.
- Phase G: Implementation Governance. This phase establishes the final version of architecture contracts that govern the overall implementation and deployment process and involves developing recommendations for each implementation project. Implementation governance functions to ensure implementation projects conform to the defined target architecture. The steps in this phase are:

1. Formulate project recommendations
2. Document architecture contract
3. Establish architecture compliance review process

- Phase H: Architecture Change Management. This phase establishes the process through which the deployment of the architecture is managed. Architecture change management provides for continuous monitoring of factors that may affect the implementation process, such as change requests, new developments in technology, or changes to the business environment. Such factors may trigger new ADM cycles as the project evolves. The steps involved are:
  1. Monitor technology changes
  2. Monitor business changes
  3. Assess changes
  4. Hold architecture board meetings

LINK TO LEARNING

The TOGAF Content Framework contains vision, requirements, business architecture, information system architecture, technology, and architecture realization. You can learn more about it by exploring the Content Framework and Enterprise Metamodel (https://openstax.org/r/76OpenGroup) in The Open Group's official guide. This framework provides an essential structure for managing and aligning IT strategies with business goals.

TOGAF's foundation architecture sets forth the following:

- Architecture building blocks and standards
- Generic services and functions to build specific services
- Technical reference model
- Standards information base

TOGAF also provides an Enterprise Continuum that addresses the Architecture Continuum and the Solution Continuum, and an Architecture Governance with guidelines relating to the following:

- Management and control of architecture works at an enterprise level
- Repositories
- Process flow control
- Architecture board, including architecture compliance and contracts
- Process and content details (Table 10.3)

Process	Content
Environment management	Regulatory requirement
Assessment/selection of models, architectures, technologies, and products	Service-level agreements (SLAs) and online licensure application systems (OLAs)
Dispensation	Authority structures

Table 10.3 Process and Content Details of Architecture Governance

Process	Content
Policy management	Organizational standards and architectures
Retirement of assets	Technology/product set

Table 10.3 Process and Content Details of Architecture Governance

CONCEPTS IN PRACTICE

EAM Faces New Challenges

The business of evolving solutions to constantly adapt to change is akin to what people face in real life as they react to change when facing unforeseen situations. This happened recently because of the COVID-19 pandemic. It is not always possible to rely on the current state of a solution and its ability to evolve. Sometimes a leap forward must be made, and it is more important to manage the need to evolve rather than try to adapt an existing, and perhaps obsolete, solution. While it differs from traditional EA techniques, EAM provides a management practice that establishes, maintains, and uses a coherent set of guidelines, architecture principles, and governance regimes that provide direction and practical help in the design and development of an enterprise's architecture to achieve its vision and strategy. As a result, EAM allows leaps forward to face unexpected changes such as having to innovate to remain competitive rather than simply focusing on being the best at providing a given service.

Strategic Adoption Road Map

EA is typically used by companies to produce a blueprint of the future state and a road map for getting there.

Companies' shift to lean and flexible operating models directly affects how their IT workforce architects design future business solutions, wherein it needs to enable the new operating model by advancing business capabilities through efficiently integrated and standardized business processes and corresponding solutions. EA ensures the alignment of business architectures with solution architectures by providing a holistic view that links all architecture domains through an enterprise reference architecture (ERA).

The result of this work, which is produced through an integrated planning process and methodology, aligns and sequences IT investments to optimally achieve the companies' strategies. The assembly of the road map draws upon three distinct architectural domains:

- Business architecture
- Business solutions architecture
- IT solutions architecture

The adoption road map results from implementation planning, as illustrated in step 8 of Figure 10.27. Creating a road map requires a current state assessment, future state definition, and gap analysis.

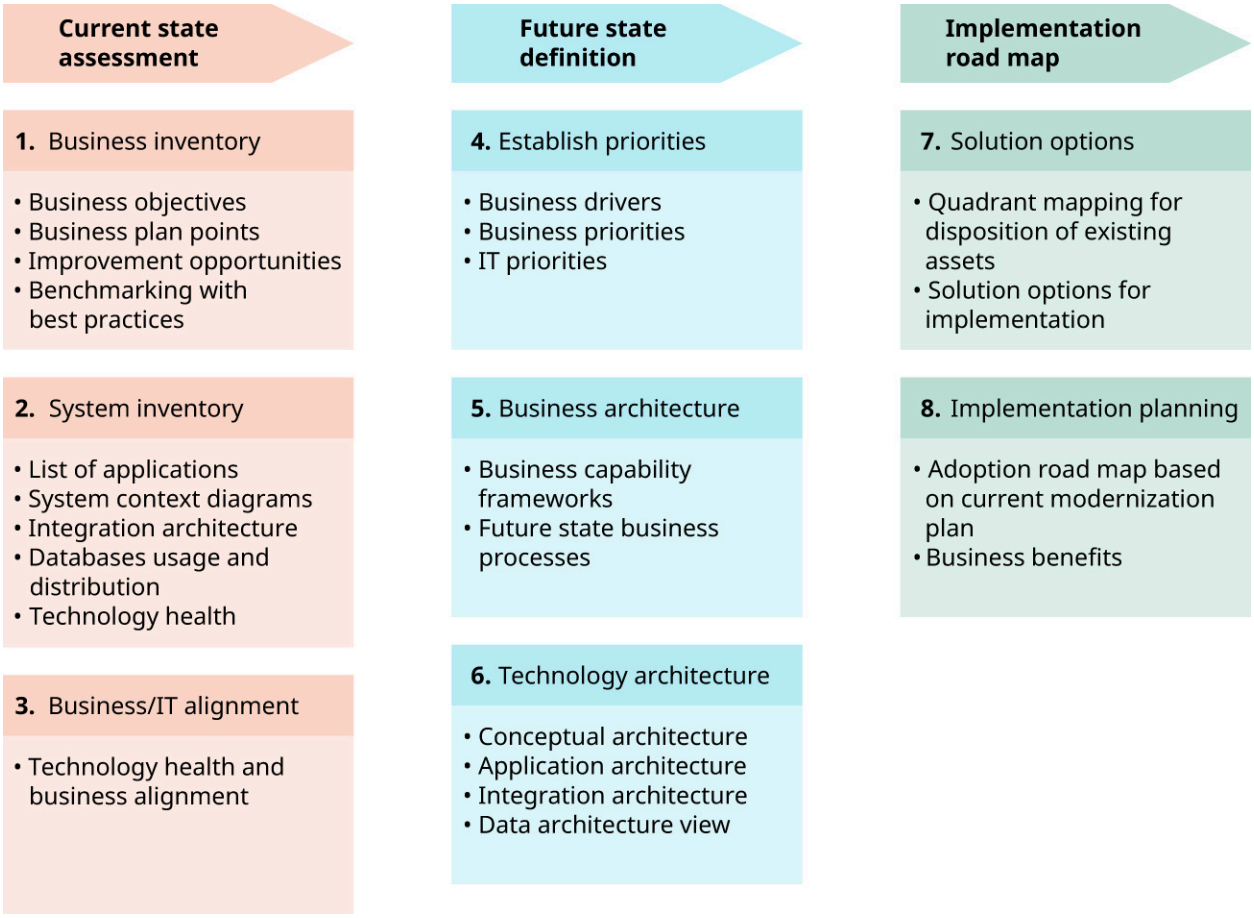


Figure 10.27 These are the steps of adopting a road map. The first step is current state assessment followed by future state definition, and the last step is road map implementation. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Creating a future state model definition requires an understanding of future state needs and the definition of a target strategy based on various business and IT drivers, as illustrated in Figure 10.28.

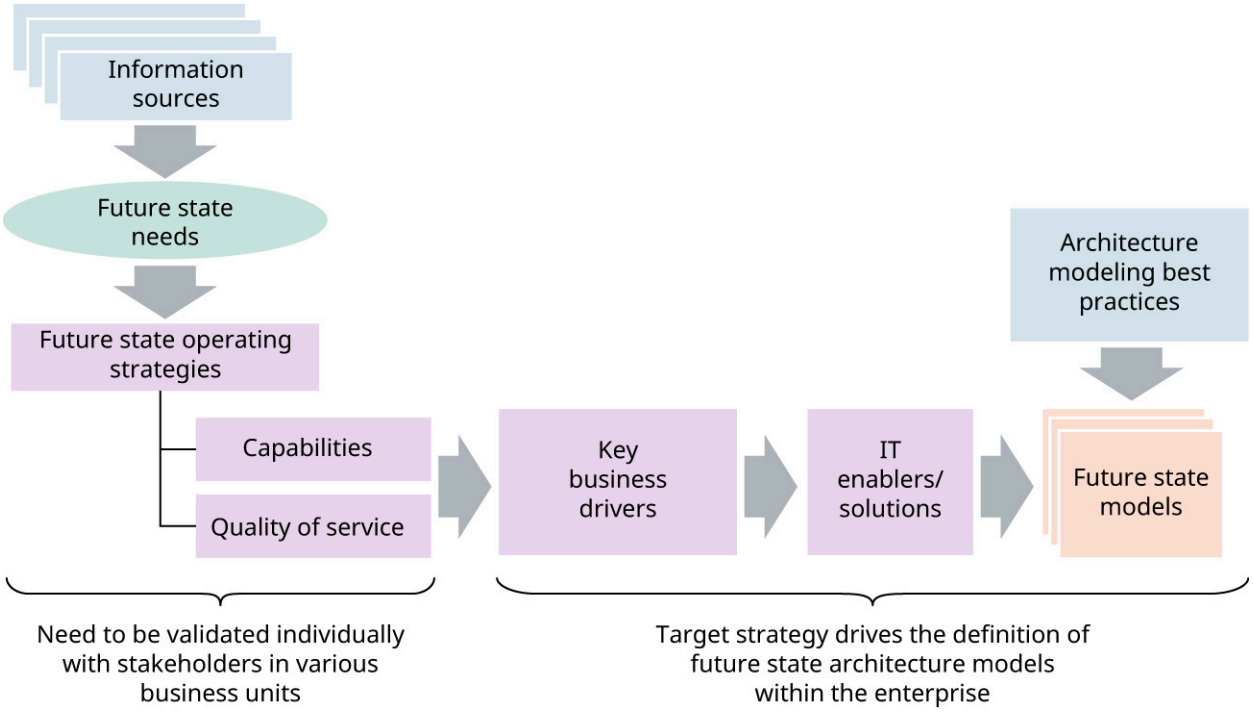


Figure 10.28 Business and IT drivers define the future state need and how it will be validated by individuals and stakeholders. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Developing business drivers requires an evaluation of the dynamics of change that are necessary to keep sustaining the business. Businesses often use the five Porter forces illustrated in Figure 10.29 to analyze business dynamics. To ensure that a product competes in the market, you should study these factors: is there any barrier to entering the market? Are there any competitors competing with your product's price? Is there any threat of substitute products? What is the buyer and supplier power?

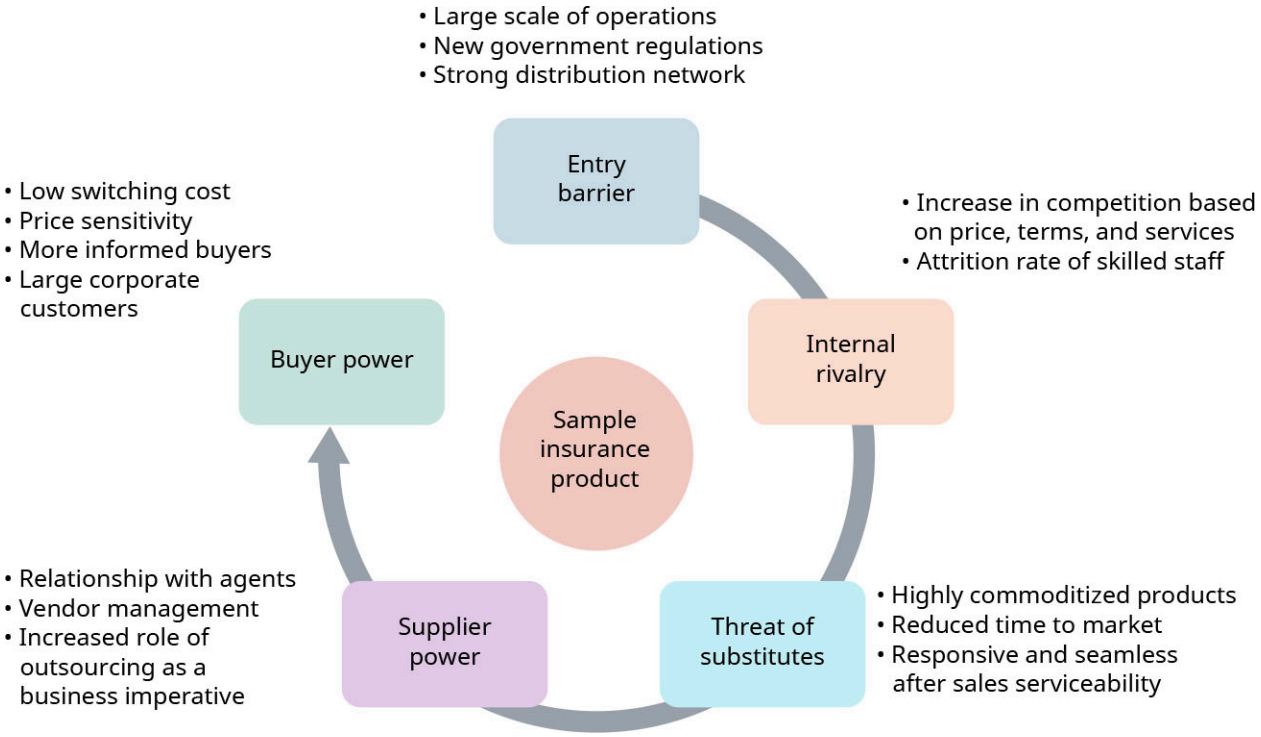


Figure 10.29 Business dynamics can be analyzed using the five Porter forces: entry barriers, internal rivalry, threats of substitutions, supplier power, and buyer power. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Enterprise Business and Technical Architectures

A reference architecture diagram may be used to create a view of a company's enterprise architecture. For example, fictional Airlines opted for a unification operating model, and its reference architecture is depicted in Figure 10.30. In the current competitive environment, Airlines concentrates on products and services through the value chain to increase customer satisfaction and the company's profit.

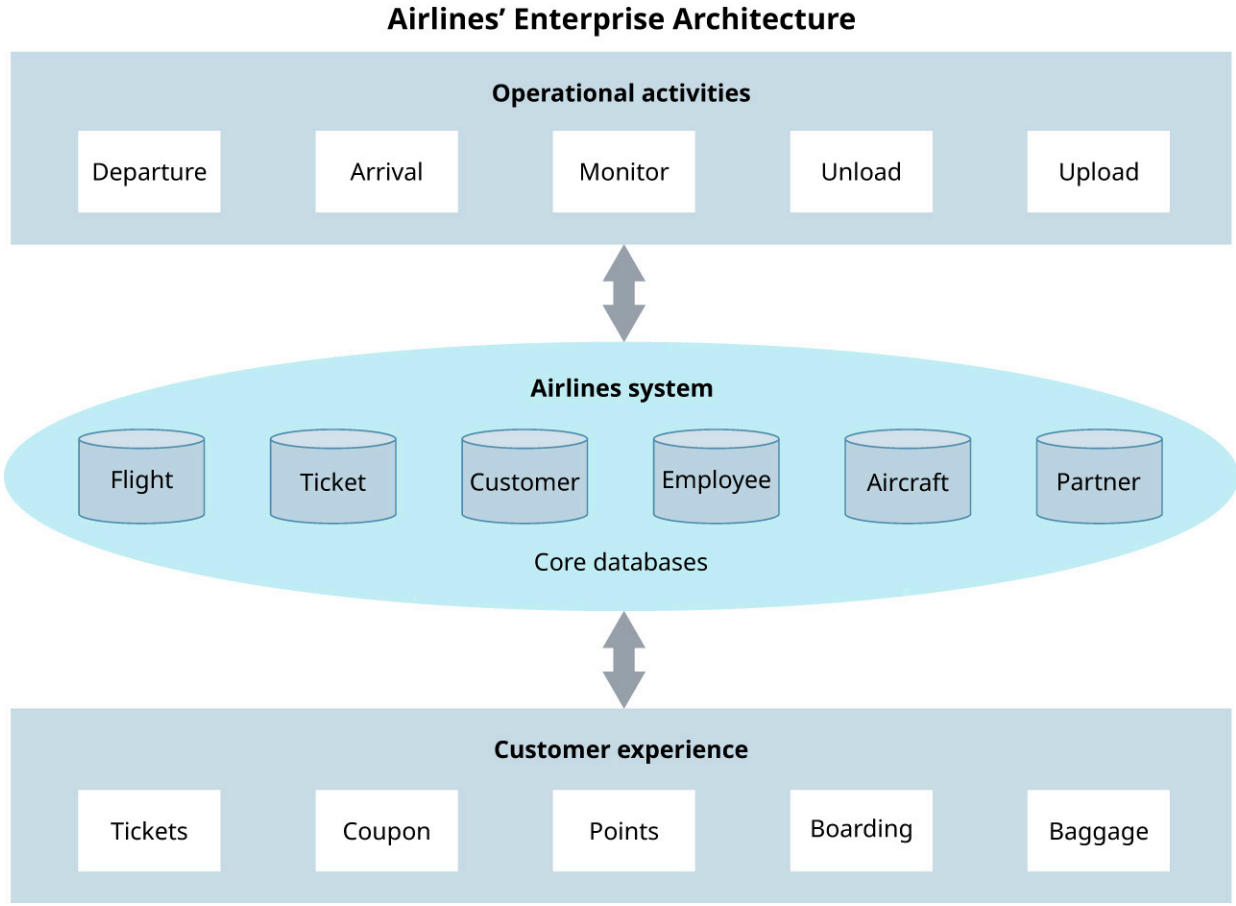


Figure 10.30 Airlines' enterprise architecture includes the customer experience pipeline to support the Airlines system with events and to create relationships between the records. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Figure 10.31 illustrates how various business and enterprise technical architecture assets may be represented. The technical architecture is divided into three layers: future state business capabilities models (e.g., Management Liability, Claims, Finance, and MIS stakeholders), business needs for the next three to five years, and the technology architecture model to support the evolving business needs with respect to best practices and strategic IT goals.

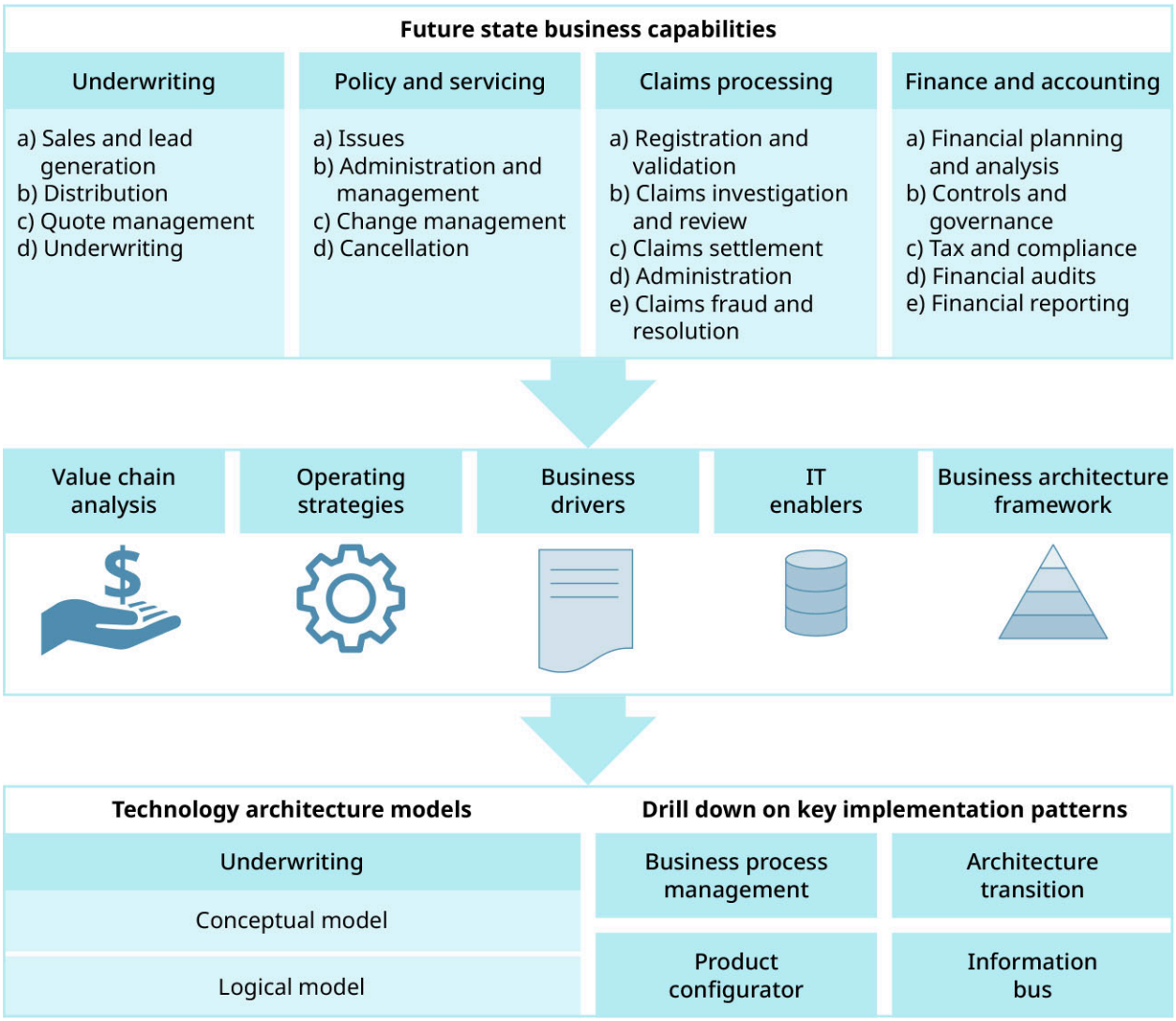


Figure 10.31 The technical architecture is divided into three layers: future state business capabilities models, business needs, and the technology architecture model. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

These assets are architectural drawings referred to as blueprints and are stored in an asset catalog according to their domains and levels of abstraction. Different UML models and blueprints provide high-level architectural details meant to help visualize the big picture. Figure 10.32 shows different levels of abstraction, including conceptual, logical, and physical.

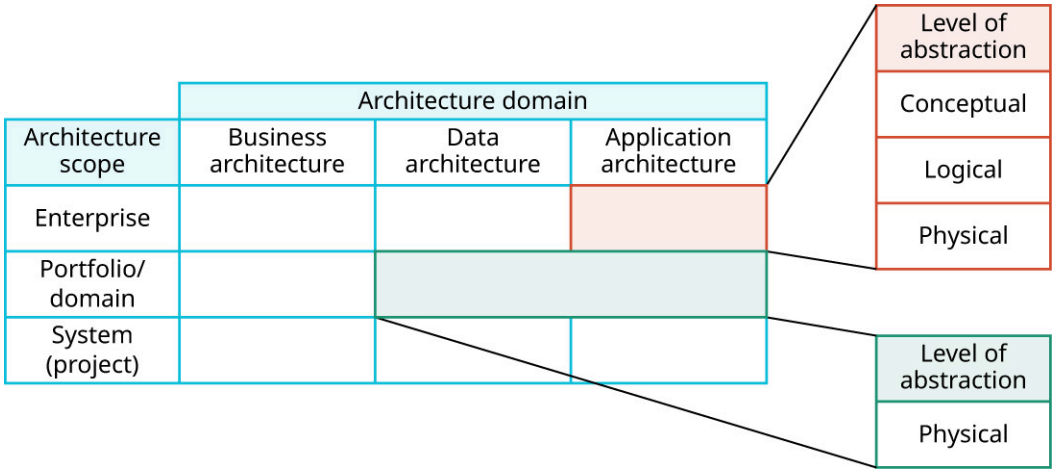


Figure 10.32 Focusing more on an application domain in the enterprise provides more details about the level of abstraction. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Enterprise Architecture Process Frameworks and Related Patterns

Various process patterns may be applied to create enterprise architecture assets. EAFs such as TOGAF implement specific processes and provide patterns and templates that may be followed to derive enterprise architecture assets. TOGAF provides an architecture metamodel that allows content adaptations in order to meet specific enterprise requirements. A **metamodel** provides evolving outlines for capabilities and relationships. Figure 10.33 illustrates a business architecture blueprint. The blueprint represents client focus, risk and financial management, business management and support based on product, and sales and/or service.

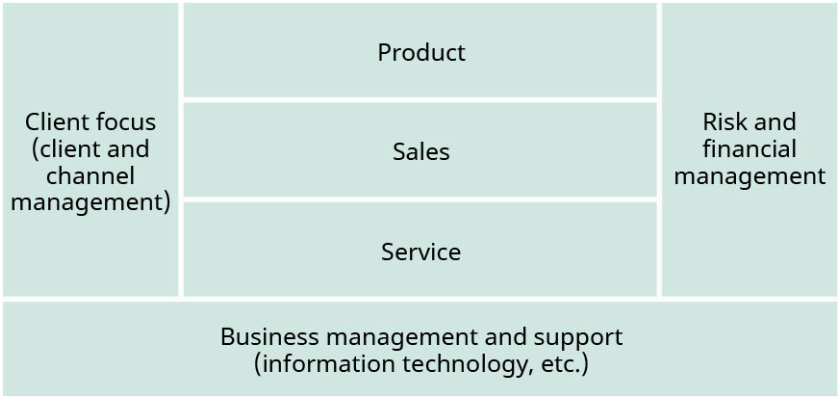


Figure 10.33 Business management focuses on the client and manages the risk for products, sales, and services. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Figure 10.34 illustrates a sample application (information system) architecture blueprint. The blueprint represents participants, interaction through electronic business gateways, processes, services, components, data about all of the components, rules, and technical services.

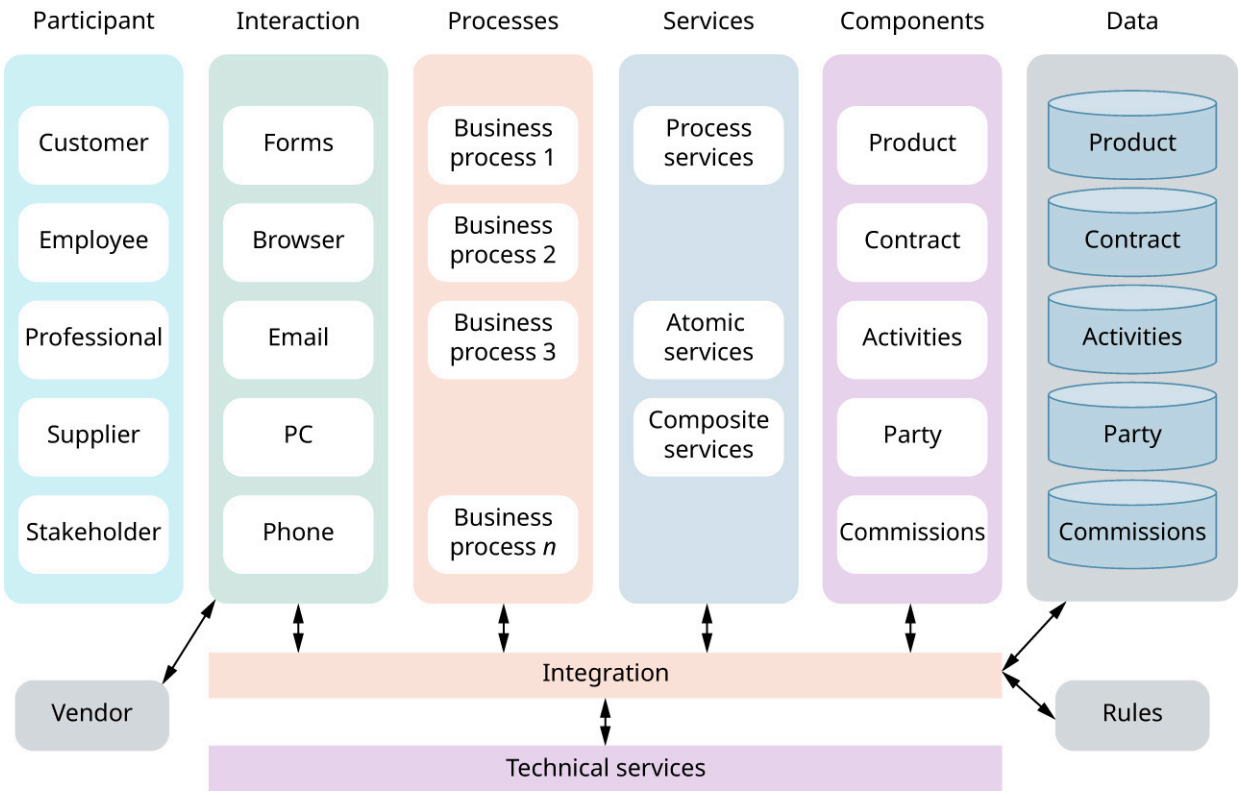


Figure 10.34 This sample representation of the blueprint for an information system shows all of the components and how they relate to each other. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Figure 10.35 illustrates a sample information architecture blueprint. The blueprint shows the relationships among transactional data stores (e.g., database that keeps track of daily transactions for one store), operational data store (e.g., database that keeps track of daily transactions for all stores), data warehouse (e.g., data management system that facilitates analytics and business intelligence queries at the enterprise level), and data marts (e.g., data management system that facilitates analytics and business intelligence queries at a business unit or department level). After running the data bus, the reporting and analysis step starts to produce operational reports, management reports, analytic reports, and e-commerce files. The blueprint for enterprise architecture helps align business and IT strategy, as well as provides a clear vision on how technology supports the enterprise goals. In addition, it's a main reference tool to coordinate the collaboration between the different enterprise components.

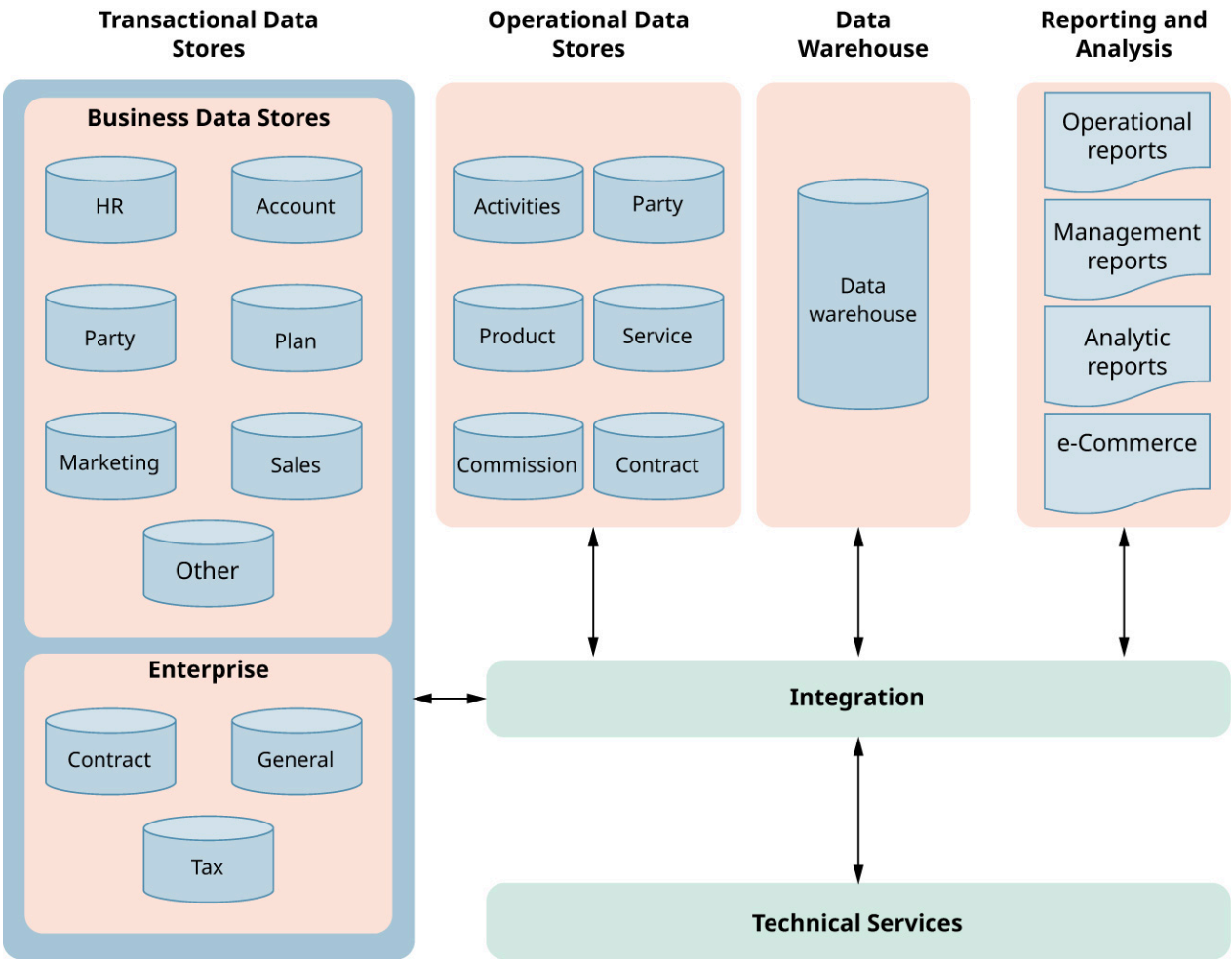


Figure 10.35 This sample representation of the blueprint for information architecture shows the relationships among the components. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Figure 10.36 illustrates a sample technology infrastructure architecture blueprint. This blueprint shows three different types of clients:

1. Browser client that is using the browser to access the Internet web service
2. Pervasive client that is using public/private network to access the gateway
3. Partner service with a database that uses a public/private network to access the gateway

In addition, the blueprint represents the interaction and security level for each user to access the integration application and data bus.

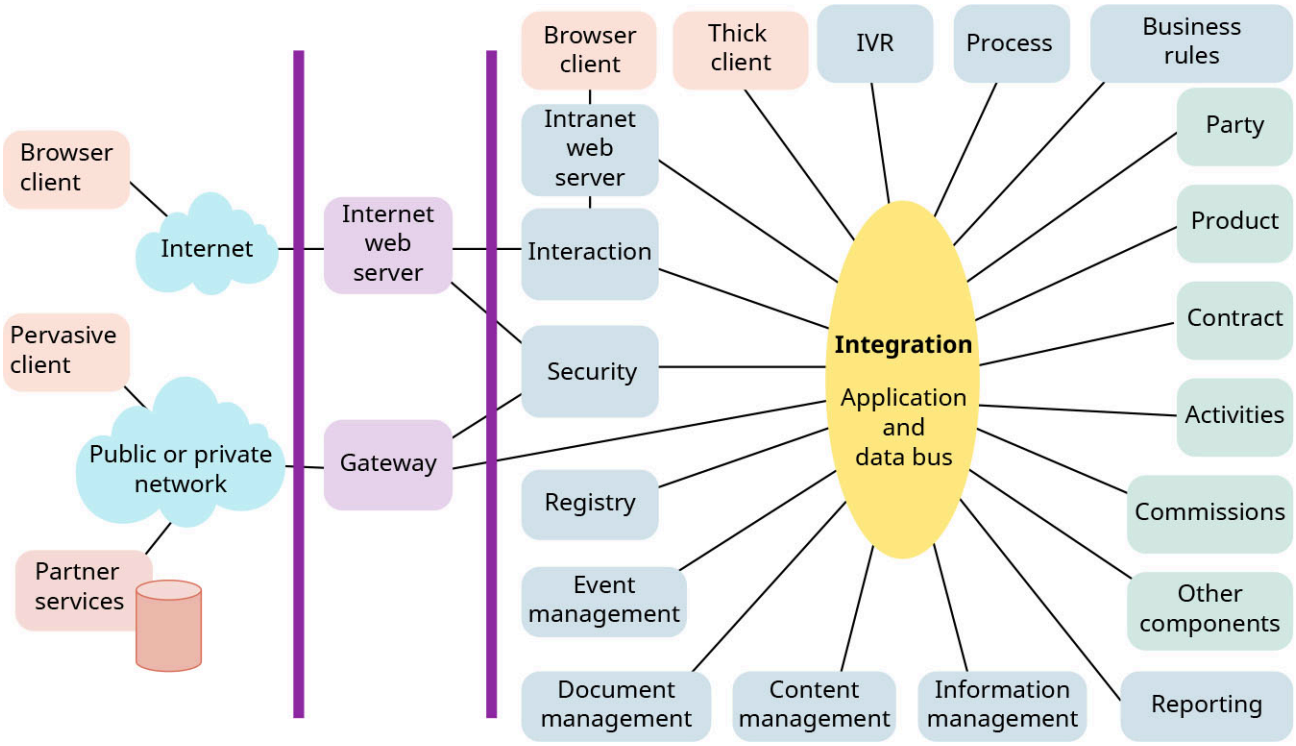


Figure 10.36 This design shows a sample representation of the technology infrastructure architecture blueprint. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

ArchDev (SecOps)

DevOps describes an approach to improving collaboration between the development and operation of services. DevOps is a blend of the terms development and operation.

ArchDev is an example of an Accelerated Architecture-Driven Digital Transformation Process, which helps companies proactively embed stakeholder interest and sustainability into the company's digital growth. **Agile EA Management (AEAM)** is a methodology used for software development and project management. AEAM uses divide and conquer methodology by breaking individual projects into smaller pieces to make them easier to manage, which speeds up design processes and produces quality products. ArchDev is enabled by methodology, tooling, and IP. It incorporates AEAM, and is compatible with TOGAF, to assess As-Is and To-Be states and a gap analysis to identify transformation initiatives that feed into the portfolio management process.

The business context driving prioritization is a major factor in selecting appropriate architectures for initiatives. Where DevOps integrates the tasks required to accelerate the development to deployment cycle, **ArchDev (SecOps)** integrates the disciplines required to identify a target architecture, which informs the solution design, monitors its implementation, and ensures its continued validity in the face of a changing environment. If changes in requirements or business context necessitate architecture revisions at any point in the development cycle or thereafter, ArchDev (SecOps) processes facilitate identifying the best replacement and following the most efficient path to migrating to it. ArchDev (SecOps), therefore, enables businesses to strike a balance between expedience and architecting business, which allows preserving business agility and sustainability.

Blueprinting Templates

An enterprise architecture blueprint is a visualization of the architecture at the conceptual, logical, and physical level of an enterprise, showing concepts, their elements, and the components that implement the elements and their interrelationships. Various blueprinting templates are available to help derive architecture diagrams. Figure 10.37 illustrates a template that may be used to create conceptual business architectures. The diagram

shows how the template may be applied in an insurance industry context. The diagram represents the enterprise value chain, which includes sales and marketing, product development, underwriting, finance and accounting, servicing, and claim processing. The blueprint includes many components and services, such as BPM processes, business capabilities, and business services.

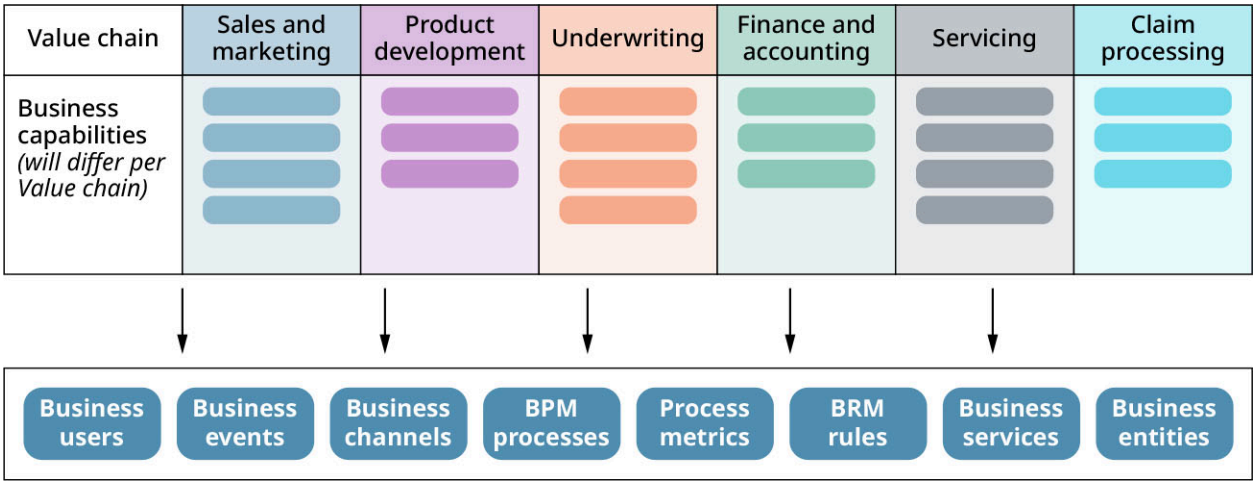


Figure 10.37 The diagram represents the enterprise value chain, which includes sales and marketing, product development, underwriting, finance and accounting, servicing, and claim processing. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

To specify the conceptual business architecture of the underwriting value chain element, Figure 10.38 drills down into the "underwriting" value chain element.

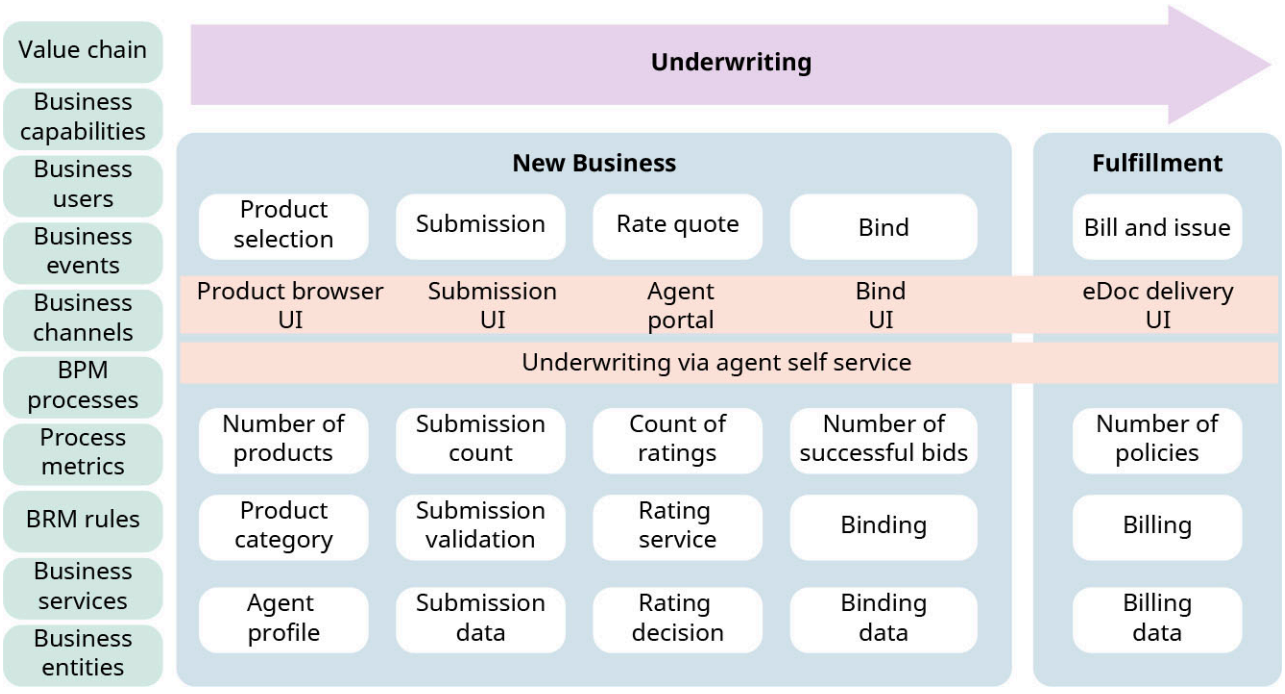


Figure 10.38 Underwriting is an element of the value chain and incorporates new business and fulfillment. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Representing the logical architecture models using a blueprint template is different. Figure 10.39 illustrates a blueprint for the logical architecture model, shows the separation of concerns through layering, and enables high cohesion and low coupling across the application component. This blueprint represents interaction integration, bus process integration, application integration, service integration, data integration, and infrastructure integration. Using this template helps to create a detailed logical architecture model that may then be split into separate layers to convey more readable details.

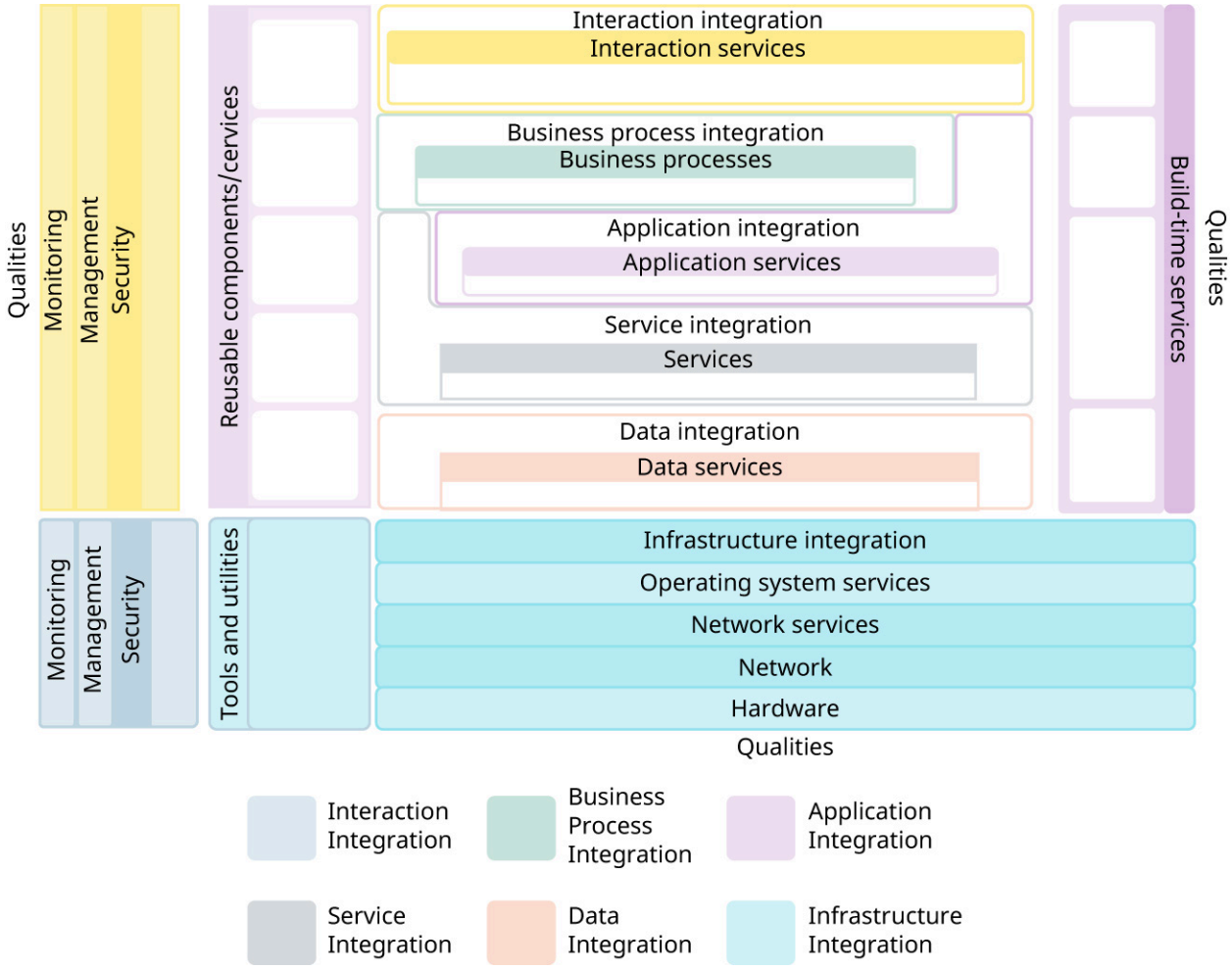


Figure 10.39 This blueprint represents interaction integration, business process integration, application integration, service integration, data integration, and infrastructure integration. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Architectural and Implementation Styles

As you may recall, the specification of architectures starts at a high level within the enterprise to help align business and IT strategy. The main strategies are typically to achieve operational excellence as well as competitiveness. Figure 10.40 represents the relationship between the customer and the enterprise to improve the digital customer experience and digital operational excellence. The digital customer experience architect roles develop architecture strategy for a customer life cycle, collaborate on digital product and service design, and guide customer technology choice. The digital operational excellence roles guide integration with ecosystem partners, support innovation with technology, and codevelop agility strategy.

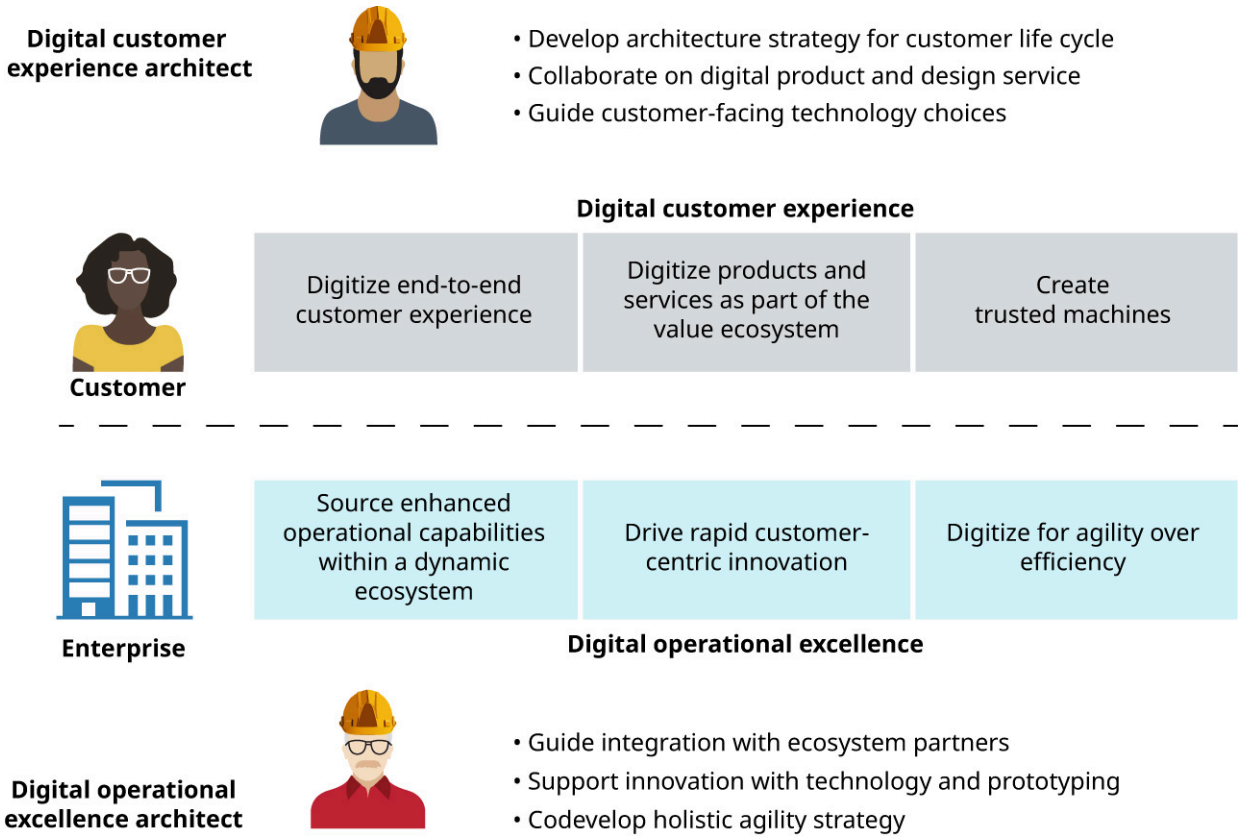


Figure 10.40 The relationship between the customer and the enterprise can improve the digital customer experience and digital operational excellence. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

This leads to the creation of enterprise reference architectures and specific business and technical architecture blueprints that represent the various views of architecture within the business, application, information, and technology domains. The scope of these views may then be refined to drill down into specific business units (portfolio) and project-level architectures. Within each level of scope, it is possible to apply architectural styles and corresponding implementation styles available within a pattern catalog. As we have learned, patterns can be applied to project-level architectures as part of high-level design and via leveraging of the architecture management umbrella activity.

Architectural and implementation styles may be reflected within blueprints. For example, Figure 10.41 represents how a conceptual technical architecture blueprint leverages the Business Process Management (BPM) architectural style. The diagram answers six questions:

1. Who is initiating the business event?
2. What is the classification of the business event type?
3. What channels are provided to initiate the business event?
4. What is the method used to digitize recognized business events?
5. How are business events digitally managed during their life cycle?
6. What automation, organizations, and processes are needed to support digital business event management?

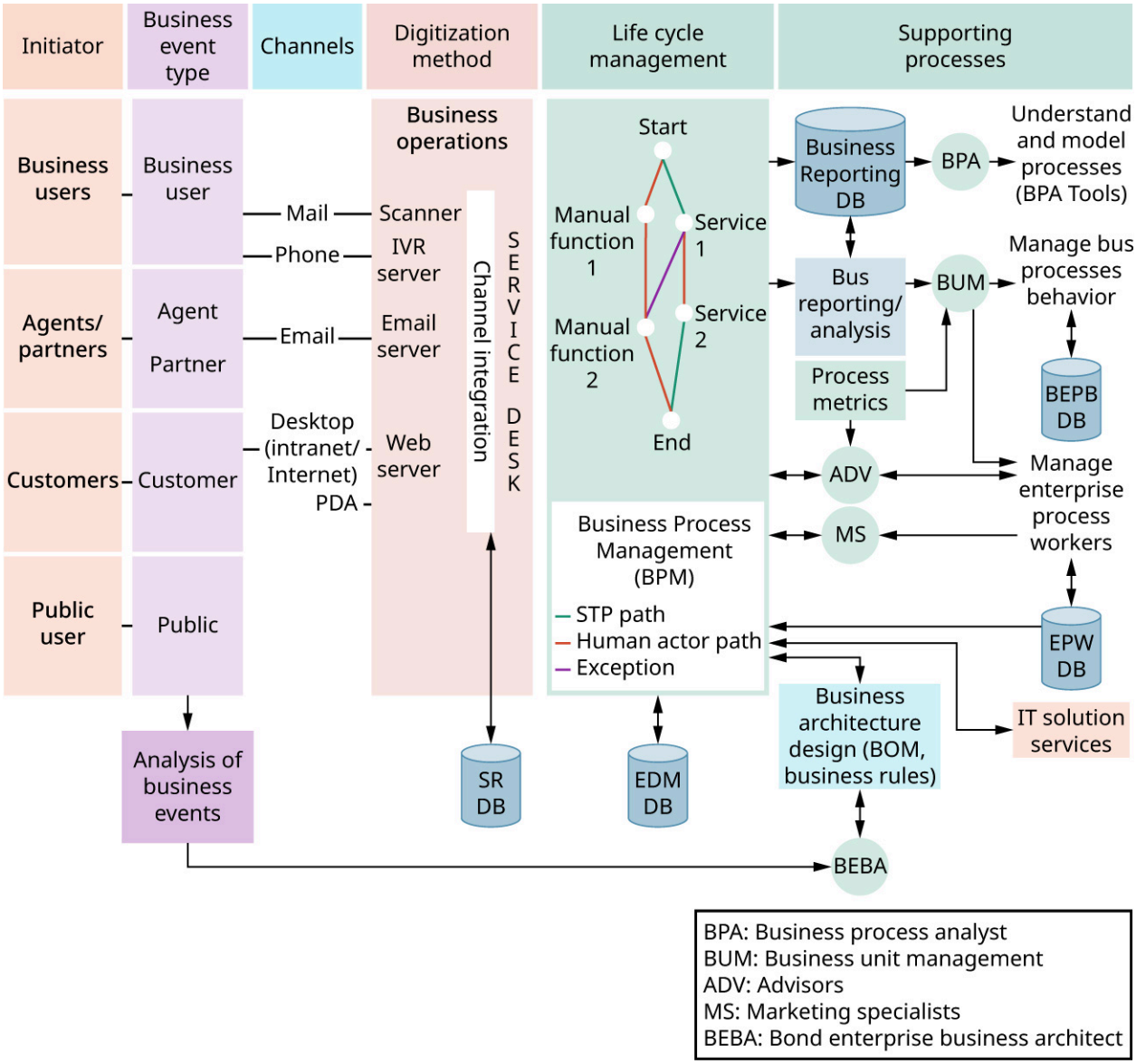


Figure 10.41 A conceptual technical architecture blueprint leverages the Business Process Management (BPM) architectural style. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Figure 10.42 illustrates a technical implementation architecture blueprint that leverages specific implementation styles, and Figure 10.43 illustrates the same along with callouts identifying specific products and related vendors selected to implement and deploy the solution.

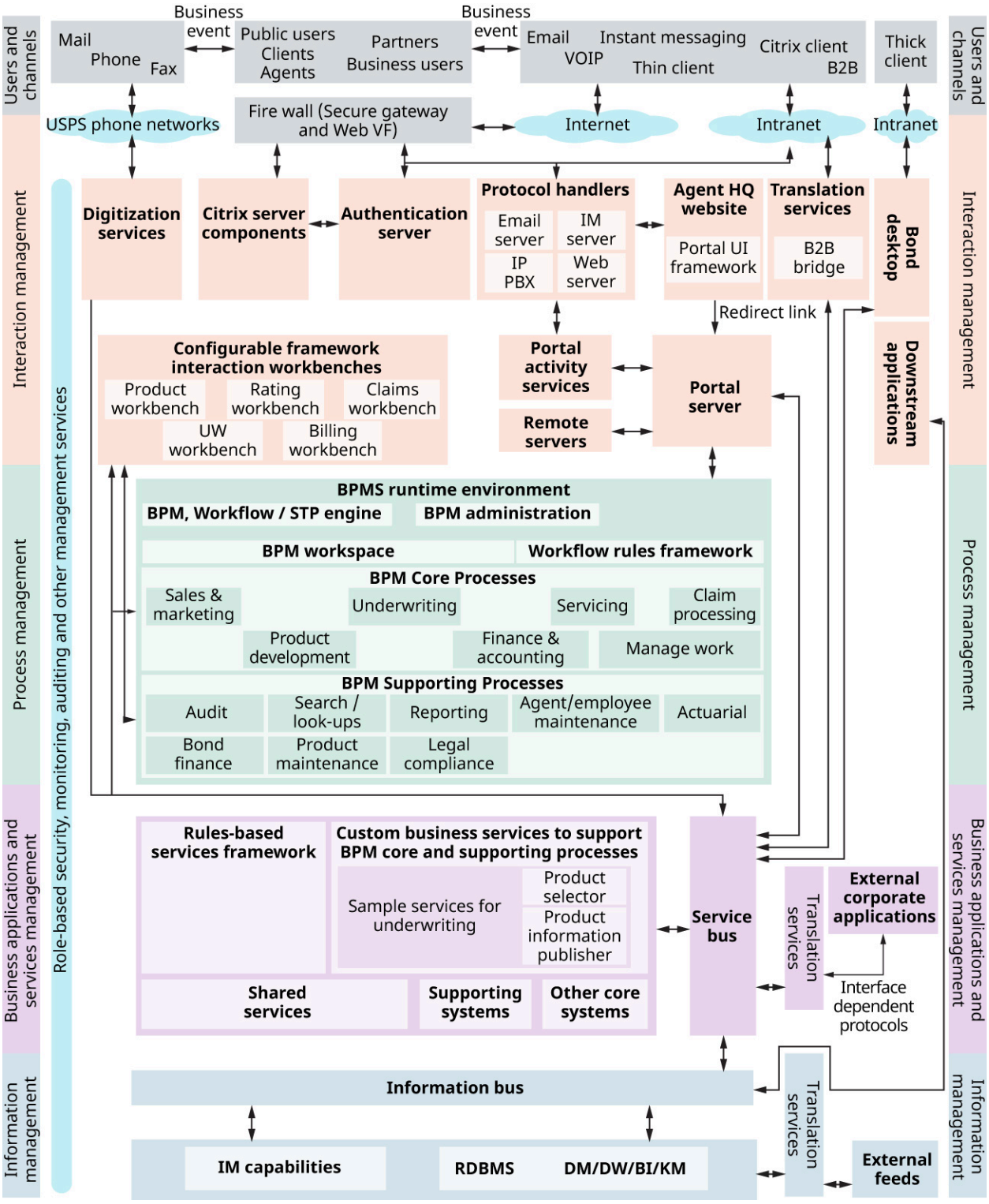


Figure 10.42 A technical implementation architecture blueprint leverages specific implementation styles. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

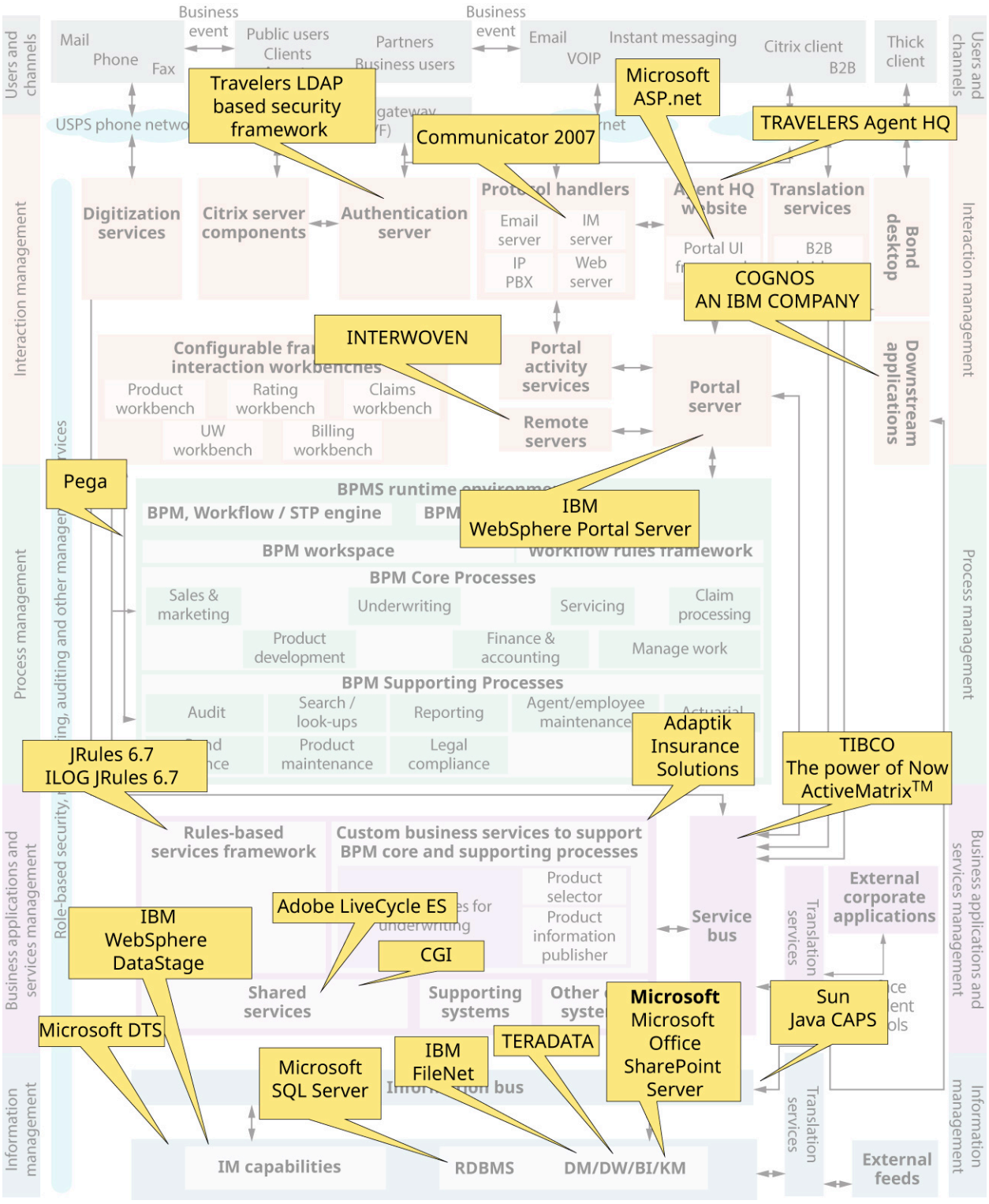


Figure 10.43 Callouts added to the technical implementation architecture shown earlier specify products and related vendors selected to implement and deploy the solution. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Human-Centered Computing Architectures and Related Patterns

Human-centered computing (HCC) is a newer term that subsumes human-computer interaction (HCI) and includes social computing and HCI. HCC studies the design, development, and deployment of mixed-initiative human-computer systems. HCC/HCI is a subfield within computer science that is concerned with the study of interaction between people and computers.

The main objective of HCC/HCI is to make computer systems more user-friendly and more usable. HCI studies the design, evaluation, and implementation of user interfaces for computer systems that are receptive to the user's needs and habits. It is a multidisciplinary field, which incorporates computer science, behavioral sciences, and design. The field of computing basically seeks to provide a framework for the integration of human sciences and computer science in a manner that promotes the progress of human activities. With this respect, human-centered computing models design computing systems that support and enrich human existence.

Users interact with computer systems through a user interface, which consists of hardware and software that provides means of input, allowing users to manipulate the system, and output, allowing the system to provide information to the user. The design, implementation, and evaluation of interfaces is, therefore, a central focus of HCC/HCI. Corresponding process patterns are applied when designing HCC/HCI architectures. Pattern-based frameworks may be used to study, analyze, and translate the interactions of people with their surroundings into collaborative solution architectures.

Cloud-Based Architectures and Related Patterns

Big clouds (e.g., Amazon AWS, Google Cloud Platform, IBM Cloud, Microsoft Azure) are available today to help put together complex solutions in an enterprise context by providing Platform as a Service (PaaS) capabilities that can be assembled very quickly. For example, the cloud-based implementation architecture shown in Figure 10.44 uses three different clouds (social media cloud for data collection, private cloud for predictive analysis, and public cloud for data mining). These clouds collaborate in a mashup configuration and provide a cost-effective way to devise and implement solution architectures that leverage data collection, secure data mining, and intelligent analytics patterns. Scalable data processing framework implementations (e.g., Hadoop, Spark) are also readily available on the big clouds to provide efficient means of handling large amounts of data.

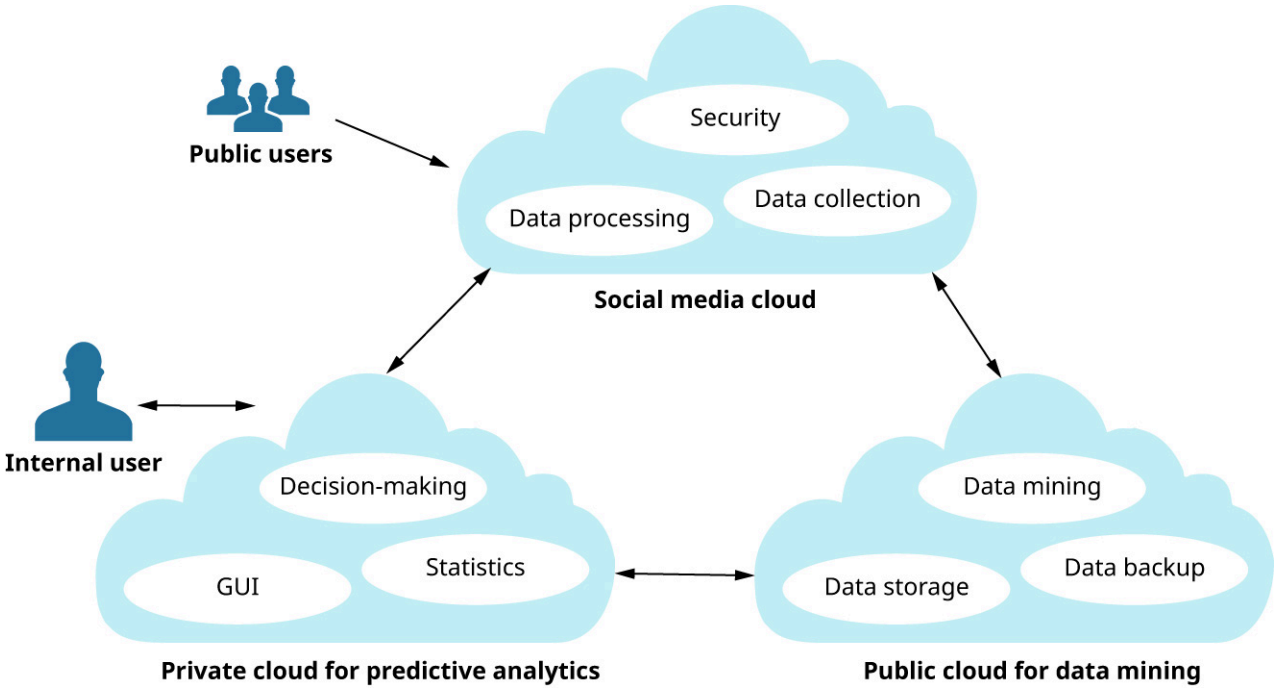


Figure 10.44 Three different clouds that include a social media cloud for data collection, private cloud for predictive analysis, and public cloud for data mining are collaborating in a mashup configuration. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Microservices Architectures and Related Patterns

A microservices architecture, called a **microservice**, is an approach to building a software solution as a set of small services (i.e., independent parts) that may be deployed locally or on the cloud. The **cloud** holds data,

software, and services that run on the Internet instead of on a local computer. While microservices architectures are mainly oriented toward providing backend services, the approach is also being used to enable solutions' front ends. Each service runs in its own process and communicates with other processes using protocols such as HTTP/HTTPS, WebSockets, or AMPQ. Each microservice implements a specific business capability within a certain context boundary, and each must be developed autonomously and be deployable independently. Finally, each microservice should own its related domain data model and domain logic and could be based on different data storage technologies such as SQL or NoSQL, and different programming languages. Figure 10.45 illustrates many patterns associated with microservices architectures such as motivating patterns and solution patterns. Multiple service hosts, database per service, access token, and health check API are examples of microservices. At a high level, microservices provide long-term agility and enable better maintainability in complex, large, and highly scalable systems by making it possible to create applications based on many independently deployable services that each have granular and autonomous life cycles.

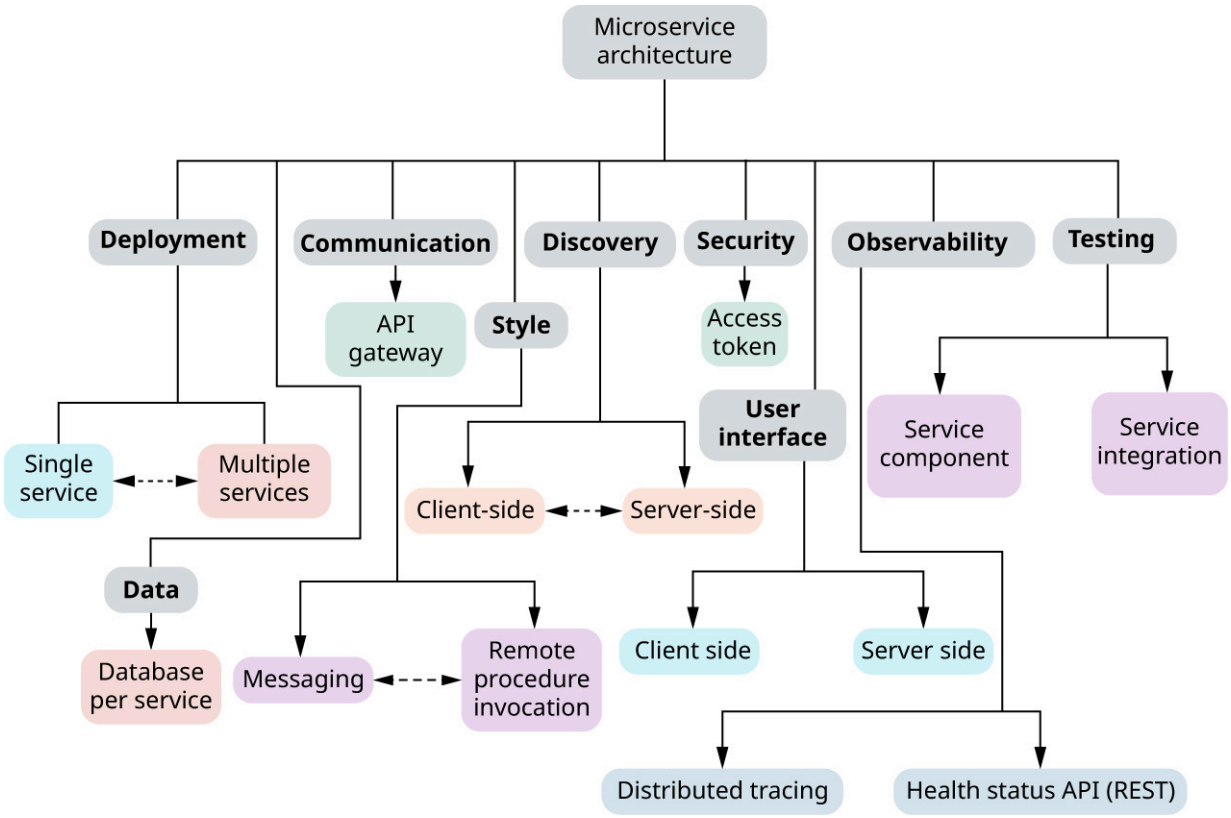


Figure 10.45 Multiple service host, database per service, access token, and health check API are examples of microservices. (data source: C. Richardson, "Microservice Architecture pattern," https://microservices.io/patterns/microservices.html; attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

As an additional benefit, microservices can scale out independently. Instead of having single monolithic applications that must be scaled out as units, specific microservices may be scaled instead. This enables scaling just the functional area that needs more processing power or network bandwidth to support demand rather than scaling out other areas of the application that do not need to be scaled, which results in cost savings because less hardware is required.

10.3 Solution Architecture Management

Learning Objectives

By the end of this section, you will be able to:

- Identify software engineering process patterns
- Describe the progression from software models to software implementation

The process of managing, designing, and describing the solution engineering in relation to specific business problems is called **solution architecture management**. A **solutions architect manager** is responsible for building teams, establishing relationships, setting strategy, and measuring and delivering results for any problem or opportunity the enterprise may face.

INDUSTRY SPOTLIGHT

Solution Architecture Management

Solution architecture management is important in every industry today. It plays an especially crucial role in the health-care industry. Solution architecture is used to maintain electronic health records and ensure patient data privacy and security. As health-care systems become more complex and data becomes more sensitive, solution architecture can be used to engineer secure, scalable, and efficient systems. These frameworks allow providers to access reliable information while maintaining compliance with data regulations.

Software Engineering Process Patterns

As we learned in Chapter 9 Software Engineering, the software process can be defined as a collection of patterns that define a set of activities, actions, and tasks required to develop computer software. Software engineering is a detailed study of engineering to design, develop, and maintain software. Software engineering process patterns establish collaborative communication between customers and software engineers to guarantee the successful completion of task patterns within the project requirements and the project scope.

As you may recall, various processes and related process patterns were discussed in Chapter 9 Software Engineering to explain how software solutions may be developed by refining an architecture model into a fully coded ready-to-deploy solution.

TECHNOLOGY IN EVERYDAY LIFE

Architecture Management in Real Life

Solution architecture management may not directly impact individuals in their everyday lives, but it indirectly influences their experiences through the development and management of various technological solutions. It ensures that systems like banking apps, online shopping platforms, and smart home devices function efficiently by managing how the different components of technology work together. When well-implemented, it improves the reliability and performance of these tools, making daily tasks more seamless.

LINK TO LEARNING

To learn more about enterprise architecture and how it is used in modern business and tech, explore the

websites of industry publications like CIO Magazine for current insights and trends (https://openstax.org/r/76EntArch) in technology management.

From Software Models to Software Implementation

Recall that a subsystem is a set of collaborating components that perform a given task included in software systems; it is considered a separate entity within a software architecture. A **component** is an encapsulated part of a software system, which has an interface and serves as a building block for the structure of a subsystem. Subsystems interact with other subsystems and components to perform their designated tasks. At a programming language level, components may be represented as modules, classes, objects, or as a set of related functions.

When designing architectures for software systems, software designers typically start with a requirements model (either explicit or implied) that presents an abstract representation of the system. The **requirements model** is a model that describes the problem set, establishes the context, and identifies the system of forces that hold sway, such as design quality attributes. The method of hiding background details (i.e., unnecessary implementation) about the data so that users only see the required information is called **abstract representation**.

In a traditional architecture design approach that does not leverage patterns, software designers gather the "big picture" from the requirements model and proceed with defining external entities such as other systems, devices, and people. The software system interacts with the external entities and defines the nature of the interaction. Software designers then need to identify a set of architectural archetypes and specify the structure of the system by defining and refining software components that implement each archetype. An **archetype** is an abstraction (similar to a class) that represents one element of system behavior.

THINK IT THROUGH

Architecture and Software Engineering

Architecture activities and software engineering activities collaborate throughout the software development process. The architecture provides the blueprint for software engineers to follow, while software engineering activities provide feedback that helps architectural decisions.

Given the fact that architecture is maintained at various levels of scope within the enterprise, how do architecture activities integrate with software engineering activities?

Figure 10.46 illustrates how enterprise architecture management activities typically integrate with the solution development life cycle within enterprises. The integration of enterprise architecture management activities with the solution development life cycle ensures that solutions are developed in alignment with the organization's strategic objectives, architectural principles, and governance requirements. It helps optimize resource allocation, streamline development efforts, enhance interoperability, and improve the overall effectiveness of technology solutions within the enterprise.

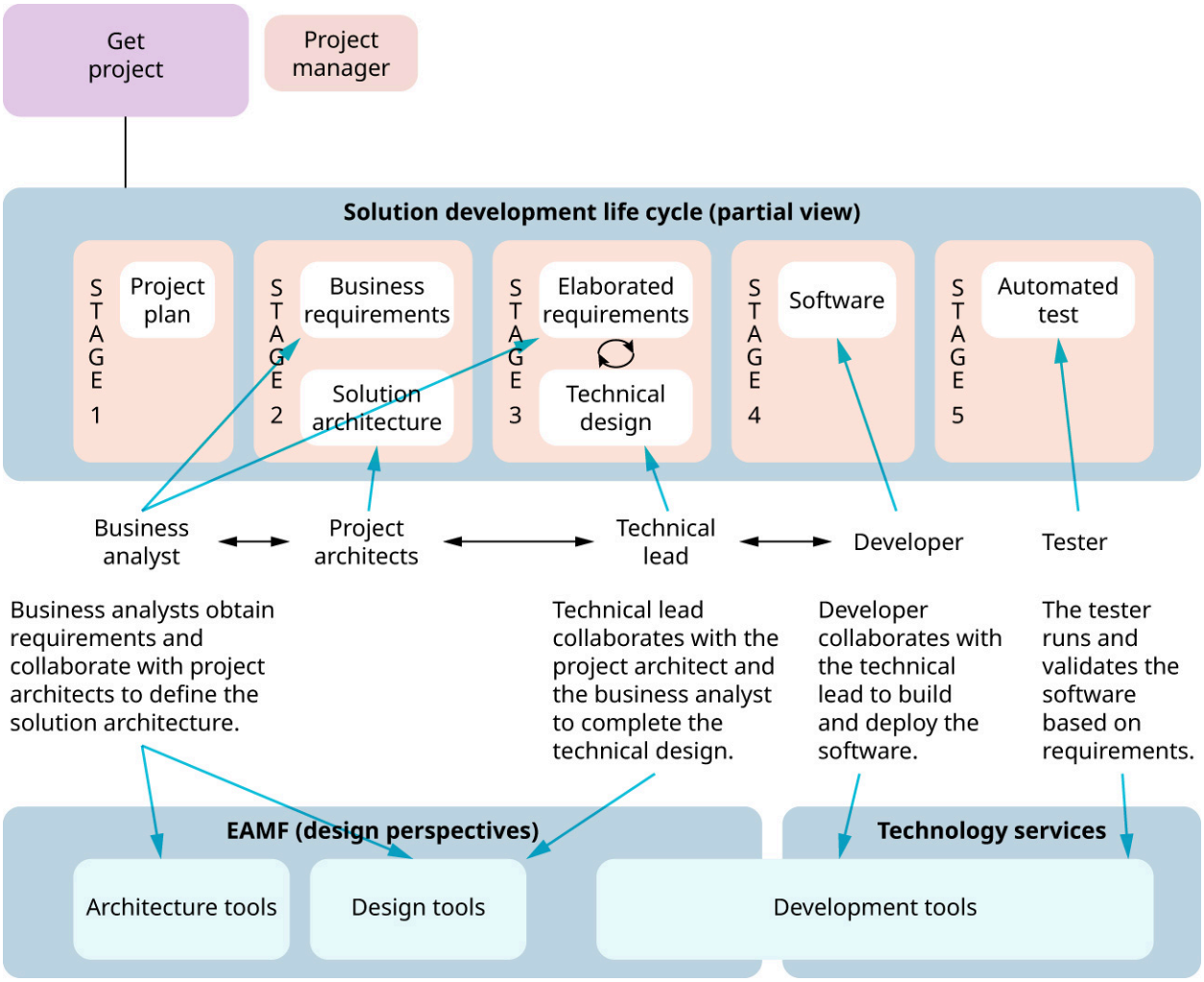


Figure 10.46 This illustrates how enterprise architecture management activities typically integrate with the solution development life cycle within enterprises. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Architectural and Design Patterns

Design patterns are a representation of the previous test solutions to a specific problem, which is useful for solving future problems. When software designers think about applying patterns or start thinking in patterns, they first try to relate patterns to the requirements model, which communicates the big picture and the context in which the software to be built resides. After that, they extract the patterns that are present at that level of abstraction and begin their design with big-picture patterns that set a context or skeleton for further design work. They then work inward from the context, looking for patterns at lower levels of specialization that contribute to the design solution. The same steps of applying patterns are repeated until the complete design is fleshed out, and the design is refined by adapting each pattern to the specifics of the software being built. Per the approach, the various patterns mentioned earlier are applied to the big picture with their level of abstraction in mind.

Remember that an architectural style is a set of characteristics that make a software system notable or identifiable; it is also a transformation that is imposed on the design of the entire software system. In any software system, there is a set of architectural constraints that restrict the roles/features of architectural elements. An architectural style corresponds to coordinated roles and allows relationships among the elements within any architecture that conforms to that style. An architectural pattern expresses a fundamental structural organization schema for the software system by predefining the subsystems, predefining the subsystems' responsibilities, creating rules and guidelines for organizing the subsystems, and building

relationships between subsystems. Finally, refining the subsystems or components of the software system, or the relationships between them, is the rule of a design pattern. This provides a scheme that describes a commonly recurring structure of communicating components and solves a general design problem within the particular context of the software system.

When delving into the architectural design of a software system, it typically becomes clear that the system cannot be designed outright by just applying patterns. Thinking in patterns does help relate patterns to a technology environment requirement model and typically reveals that software solutions should leverage characteristics from several architectural styles (e.g., cloud, P2P, and microservices). It may also help identify the need to leverage the EAM, BPM, and Intelligent Autonomous Agents architectural patterns as part of a software solution.

CONCEPTS IN PRACTICE

Enterprise Architecture and Land Development

It is easy to think about a land development project as being focused on developing a number of home communities populated with custom homes. The land development company can be considered at the enterprise level. Each individual home community is developed by business units within the enterprise, and individual homes within each home community are created as individual projects. This can be compared to what is happening as enterprise architecture is applied to manage enterprise, portfolio, and project-level architectures. At the project level, managing software projects becomes the concern of the software engineer, but it is clear that enterprise architecture is required to ensure that the individual projects are implemented to follow the guidelines established at a higher level.

Enterprise Technical Architecture Frameworks

While the architectural design approach leads the way to pattern-based design, patterns themselves may not always be sufficient to develop a complete design. In some cases, it may be necessary to provide a framework. Remember that a framework is an implementation-specific skeletal subsystem for design work. Mini-architecture provides the generic structure and behavior for a family of software abstractions. Memes (i.e., metaphors) specify the subsystems' collaboration within a given domain. You can reuse the mini-architectures along with a meme to create the framework, which is not an architectural pattern but rather a skeleton with a collection of "plug points" that enable the framework to be adapted to a specific problem domain. The plug points—also called *hooks* and *slots*—enable you to integrate problem-specific classes or functionality within the skeleton. Therefore, a framework defines the architecture of a family of subsystems, provides the basic building blocks to create them, and defines where adaptations for specific functionality should be made. In general, patterns can be used to describe the framework as information about the system and the frameworks are software that defines a generic design.

Software Stacks

A **software stack** is a collection of independent components or subsystems such as operating systems, protocols, databases, architectural layers, and function calls that work together to support the execution of an application. As architectural models are being transitioned into implementation architectures, it is necessary to identify specific products that may be used as part of the implementation. This is where the selection of product/software stacks takes place. For example, a choice may be to select among IBM, Oracle, or Microsoft product stacks such as Dynamic 365 (or a combination) to implement a particular architecture.

Figure 10.47 is an example of a simple software stack for a website implementation. The stack consists of a user interface, application layer, application infrastructure, and tech infrastructure.

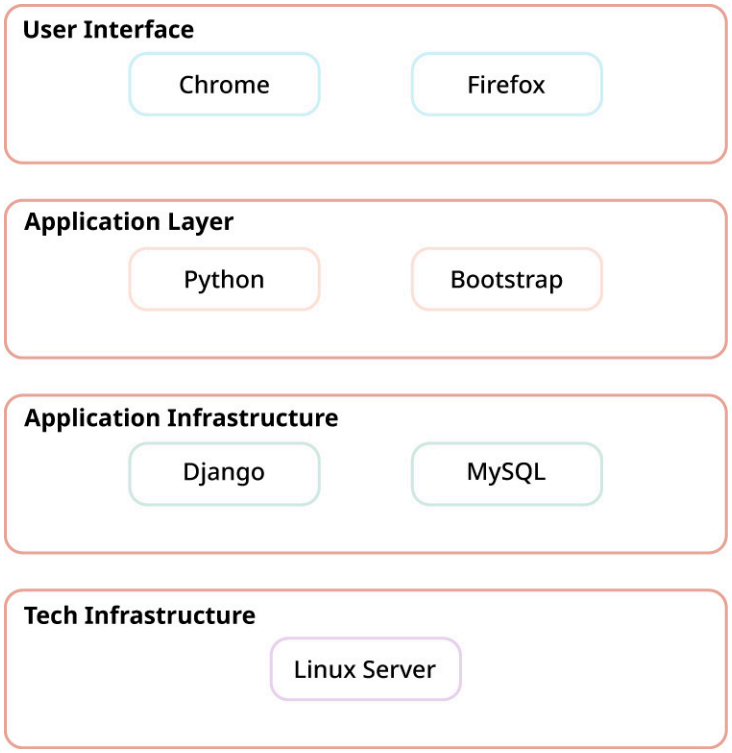


Figure 10.47 The stack consists of a user interface, application layer, application infrastructure, and tech infrastructure. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Once a software stack is selected, various products can be added via callouts on the corresponding implementation architecture blueprint.

GLOBAL ISSUES IN TECHNOLOGY

Security of Architecture Management around the Globe

Global issues in technology can have significant implications for solution architecture management worldwide. Solution architecture management needs to address data protection regulations, ensure secure handling of sensitive data, and implement robust security measures to safeguard against cyber threats. Complying with the European Union's GDPR requires organizations to implement strict data privacy controls. This, in turn, influences the design of global IT solutions as they must account for local and regional regulations.

Implementation Patterns and Idioms

As we discussed before, design patterns address general structural principles; in contrast, idioms represent low-level patterns that describe how to solve implementation-specific problems in a programming language, such as managing memory in C++ or Java.

As architectural models get transitioned into implementation architectures, it becomes necessary to specify how architectural and design patterns are realized practically within the solution. This mapping actually starts at a high level with the architectural styles themselves. For example, a BPM architectural style may be realized via the selection of an enterprise REST Services and with a corresponding product such as MuleSoft. This subsequently dictates the use of specific implementation patterns and idioms specific to the MuleSoft product stack.

Chapter Review

Key Terms

abstract representation used to hide background details
Agile EA Management (AEAM) methodology used for software development and project management; AEAM breaks individual projects into smaller pieces to be easier to manage, which speeds up design processes and produces quality products
Agile software development interactive approach to software development that delivers value to the customers
ArchDev (SecOps) example of an Accelerated Architecture-Driven Digital Transformation Process, which helps companies proactively embed stakeholder interest and sustainability into the company's digital growth
archetype abstraction (similar to a class) that represents one element of system behavior
architectural style abstract patterns located at the top of the pattern hierarchy; they capture sets of characteristics and features that make a structure identifiable
architecture continuum represents a structure of building blocks to reuse the architecture assets that conform to a language pattern
Architecture Development Method (ADM) detailed step-by-step process for developing or changing an enterprise architecture
business process series of steps performed by a group of stakeholders to achieve an enterprise concrete goal
business service choreography example of business architecture pattern that focuses on the observed sequence of messages exchanged by peer services when performing a unit of work
business service orchestration generalization of business service composition and a pattern used to describe the sequencing of business services
cloud storage solution that holds data, software, and services that run on the Internet instead of on a local computer
component encapsulated part of a software system, which has an interface and serves as a building block for the structure of a subsystem
data architecture design framework that structures how data is collected, stored, managed, and utilized within a system
enabler supports the activities needed to extend the architectural runway to provide future business functionality
enterprise architecture (EA) conceptual blueprint that defines the structure and operation of organizations
enterprise architecture framework (EAF) ensures that enterprise solutions are in alignment with the evolving vision and strategy of the organizations that use these solutions to operate and conduct day-to-day business
enterprise architecture management (EAM) helps guide the adoption of technology stacks and corresponding implementation frameworks
framework implementation-specific skeletal subsystem for design work
governance processes and organizational structures that ensure compliance with the reference architecture
idiom phrase or expression whose meaning cannot be inferred from the literal definitions of its individual words, but instead is understood through common usage within a language
IT automation process of creating systems to reduce manual intervention
IT context management dynamic IT process that uses data in one application to point to data resident in a separate application
IT governance process that ensures the effective and efficient use of IT in enabling an organization to achieve its goals
meta-framework framework in more detail

metamodel evolving outlines for capabilities and relationships
method set of repeatable processes that ensure consistent and controlled execution of the architecture
microservice approach to building a software solution as a set of small services that may be deployed locally or on the cloud
pattern catalog collection of patterns organized according to specific characteristics and according to how the relationships between them are defined
pattern hierarchy architectural styles, architectural patterns, and design patterns
pattern language connected view of how to apply one pattern in the presence of another
patterns management series of patterns that create an organization chart for developing software
principles core rules that guide the design and implementation of the reference architecture
requirements model model that describes the problem set, establishes the context, and identifies the system of forces that hold sway, such as design quality attributes
road map used to guide an organization with planning and achieving business goals over time through technology
singleton design pattern that restricts the instantiation of a class and ensures that only one instance of the class exists
software engineering detailed study of engineering to design, develop, and maintain software
software stack collection of independent components or subsystems such as operating systems, protocols, databases, architectural layers, and function calls that work together to support the execution of an application
solution architecture management managing, designing, and describing the solution engineering in relation to specific business problems
solutions architect manager responsible for building teams, establishing relationships, setting strategy, and measuring and delivering results for any problem or opportunity the enterprise may face
subsystem set of collaborating components that perform a given task included in software systems, and it is a separate entity within a software architecture
The Open Group Architecture Framework (TOGAF) EA methodology and framework used by leading organizations to improve business efficiency

Summary

10.1 Patterns Management

- A pattern documents a recurring problem/solution pairing within a given context.
- Patterns can be used to construct architectures at various levels of scope to guarantee specific properties.
- Patterns are classified as design or implementation centric. Design-centric patterns are organized in a pattern hierarchy that includes architectural styles, architectural patterns, and design patterns. Implementation-centric pattern hierarchy includes implementation styles, implementation patterns, and idioms.
- In patterns terminology, styles are named collections of architectural decisions that are applicable in a given solution context, constrain architectural decisions that are specific to a solution within that context, and elicit beneficial qualities in each resulting system.
- An enterprise architecture (EA) is a conceptual blueprint that defines the structure and operation of organizations.
- Design modeling provides a variety of different views of the system like architecture plans for the enterprise.
- The architecture may appear in different levels of focus from a top-down standpoint (i.e., enterprise, portfolio, or project).
- A pattern catalog is a collection of patterns that are organized according to specific characteristics and the relationships between them are defined.
- Implementation patterns are typically specific to technology stacks that are selected as part of the specialization of a solution design.

- Patterns management comprises a series of patterns that create an organization chart for developing software.

10.2 Enterprise Architecture Management Frameworks

- Enterprise architecture (EA) is a comprehensive, well-defined approach of business planning to utilize information technology to meet the objectives of the business vision by aligning business and technology strategies.
- Enterprises focus on activities that allow them to meet their current and future objectives.
- The enterprise IT strategy is based on a collective set of principles that form a consistent framework for technology decision-making and reflect a level of consensus among key stakeholder technology groups.
- Business and technology executives are responsible for managing IT projects so that they achieve division-wide and company-wide objectives by engaging with the enterprise architects.
- Aligning business and technology strategies is typically difficult to do on an ongoing basis because of the lack of alignment with enterprise-driven initiatives. Enterprise architecture helps align business and technology strategies.
- There are many enterprise architecture frameworks, such as TOGAF, Gartner, C4ISR, CORBA, FEA, and Zachman.
- The Open Group Architecture Framework (TOGAF) is an EA methodology and framework used by leading organizations to improve business efficiency.
- Architecture Development Method (ADM) is a detailed step-by-step process for developing or changing an enterprise architecture as well as a content framework to help drive greater consistency in the outputs that are created when using the ADM.
- ADM defines the TOGAF approach for establishing processes linked with enterprise architecture. It provides a recursive and tested process development business architecture; every phase of the ADM is iterative in nature to develop an enterprise-wide architecture.
- ADM phases are Preliminary Phase, Architecture Vision, Business Architecture, Information System Architecture, Technology Architecture, Opportunities and Solutions, Migration Planning, Implementation Governance, and Architecture Change Management.
- EA is typically used by companies to produce a blueprint of the future state and a road map for getting there.
- A road map is used to guide an organization with planning and maintaining business goals over time through technology.
- ArchDev is an example of an Accelerated Architecture-Driven Digital Transformation process, which helps companies proactively embed stakeholder interests and sustainability into the company's digital growth.
- Agile EA Management (AEAM) is a methodology used for software development and project management.
- An enterprise architecture blueprint is a visualization of the architecture at the conceptual, logical, and physical level of an enterprise, showing concepts, their elements, as well as the components that implement the elements and their interrelationships.
- HCC studies the design, development, and deployment of mixed-initiative human-computer systems.
- HCC/HCI is a subfield within computer science concerned with the study of the interaction between people and computers.
- Microservices is an approach to building a software solution as a set of small services that may be deployed locally or on the cloud.

10.3 Solution Architecture Management

- Solution architecture management is managing, designing, and describing the solution engineering in relation to specific business problems.
- The software process can be defined as a collection of patterns that define a set of activities, actions, and tasks required to develop computer software.
- Software engineering process patterns establish collaborative communication between customers and software engineers to guarantee a successful completion of task patterns within the project requirements

and the project scope.

- Subsystems are sets of collaborating components performing a given task included in software systems and it is considered a separate entity within a software architecture.
- At a programming language level, components may be represented as modules, classes, objects, or as a set of related functions.
- The requirements model describes the problem set, establishes the context, and identifies the system of forces that hold sway.
- Design patterns are a representation of the previous test solutions in relation to specific problems, which are useful for solving future problems.
- An architectural style corresponds to a coordinated role and allows relationships among the elements within any architecture that conforms to that style.
- The framework is an implementation-specific skeletal subsystem for design work.
- A software stack is a collection of independent components or subsystems such as operating system, protocols, databases, architectural layers, and function calls that work together to support the execution of an application.
- As architectural models get transitioned into implementation architectures, it becomes necessary to specify how architectural and design patterns are realized practically within the solution.



Review Questions

1. You are writing a class that has been identified as a key object of the system; therefore, only one can exist in the system. What type of design pattern should you take to design the class?
   a. singleton pattern
   b. builder pattern
   c. abstract factory method pattern
   d. proxy pattern

2. What do you use for documenting a recurring problem and providing a reusable template in the form of a problem-solution pair within a given context?
   a. patterns management
   b. pattern hierarchy
   c. pattern
   d. pattern catalog

3. What is the term for abstract patterns located at the top of the pattern hierarchy that capture a set of characteristics and features that make a structure identifiable?
   a. architectural patterns
   b. architectural style
   c. architecture continuum
   d. enterprise architecture

4. What is the difference between architectural pattern and architectural style?

5. What is the difference between analysis and design models and implementation patterns?

6. Why are pattern catalogs needed?

7. What is the current state of patterns management today and are there exhaustive sources of patterns that can be easily consulted? Why or why not?

8. Is TOGAF primarily concerned with enterprise architecture/detailed application architecture? Explain your answer.

9. In what TOGAF ADM phase is business architecture gap analysis done?
   a. Phase A

b. Phase B
c. Phase C
d. Phase D

10. How are business and technical architecture assets cataloged within enterprises?

11. What is a blueprinting template?

12. What is a key benefit of microservices?
a. fewer lines of code to be written than in a large monolithic application
b. increased agility and improved scalability
c. more available design patterns
d. specifically align with TOGAF

13. What software engineering process follows a linear process flow?
a. waterfall model
b. Agile development model
c. Scrum
d. DevOps

14. What is an example of a software stack?
a. scripting language, object-oriented language, database
b. client Android tablet, Windows server
c. microservice 1, microservice 2, API gateway
d. Windows, Java, Apache Server, Mongo DB, Java Script, React

15. What term is defined as a model that describes the problem set, establishes the context, and identifies the system of forces that hold sway, such as design quality attributes?
a. architecture model
b. requirements model
c. business model
d. operating model

16. What is a solutions architect manager responsible for?
a. determining user and system requirements
b. writing test cases for verification and validation of user and system requirements
c. planning out project schedules, providing cost estimates, and generating invoices
d. building teams, establishing relationships, setting strategy, and measuring and delivering results

17. Which two choices should the main drivers of determining what a technology stack should be?
a. developer's experience and knowledge of technologies
b. user capability requirements and system requirements
c. technologies the organization has already purchased and those it has integrated
d. implementation patterns and idioms

Conceptual Questions

1. Can object-oriented concepts such as inheritance and composition be used to assemble patterns? Provide examples to illustrate your answer.

2. Why is it difficult to build complete solutions by assembling patterns at various levels of specialization?

3. How does the enterprise architecture as a strategy process pattern apply to the foundation for execution?

4. What is the major difference between ArchDev and DevOps?

5. The approach discussed in this chapter appears to be top-down as it comes to deriving solution

architectures from higher levels of architecture scope. Is it possible to start from a solution architecture and establish the architecture scopes that correspond to it at the portfolio and enterprise levels?

6. Does a software stack relate to a particular implementation style? Assuming that the target solution architecture implements more than one style, how is it generally possible to establish a software stack for it?

Practice Exercises

1. Provide examples of architectural styles, architectural patterns, and design patterns for the business, application, information, and technology domains.
2. Provide an example of architectural patterns in our daily activities.
3. Research how large corporations such as Amazon and Netflix have adopted microservices. What are the benefits of implementing a microservice design pattern according to these companies?
4. Give an example of a framework that can be used to integrate the TOGAF architecture domains. What is the benefit of integration?
5. Research Zachman and FEAF. Which framework has views and viewpoints very similar to TOGAF?
6. State process patterns that are used for solution architecture management purposes as part of software engineering.
7. Give examples of enterprise technical architecture frameworks that can be used to construct cloud-based solutions.
8. Research examples of full stack technologies for different application types. Is there a fixed number of tools and technologies that meet the full stack definition? What is the main driver to determine the number of tools and technologies needed in a stack?
9. How would the system deployment environment affect the possible technology stack used to implement a solution? Provide a specific example of how we could not use a given technology for a given deployment environment.

Problem Set A

1. Document the design patterns set forth by Erich Gamma in terms of the pattern template, pattern hierarchy, pattern catalog, and pattern languages that are provided to work with these patterns. Documentation about Erich Gamma's catalog of design patterns is widely available on the Internet.
2. Perform some research on the Internet and explain the difference between OMG's OMA architectural style and the microservices architectural style.
3. Perform some research on the Internet to become familiar and explain/document the following common industry standards and acronyms: PRINCE2, ITIL, US NIST, US FIPS, SIB, COBIT, and ATAM.
4. Perform some research on the Internet and provide a pattern catalog for HCC architecture models.
5. Perform some research on the Internet and provide a pattern catalog for cloud-based architecture models.
6. Perform some research on the Internet and provide a pattern catalog for cloud-based implementation architectures (it is fine to limit the scope of vendors used to implement the solution).
7. Perform some research on the Internet and provide a pattern catalog for microservices implementation architectures (it is fine to limit the scope of vendors used to implement the solution).
8. What is the difference between an architectural pattern and a design pattern?

9. Provide an example of a technology stack that could be utilized to create a cloud-based application.
10. Research commercially available development tools. Provide an example of a company and product line that are available to meet full-stack development needs.

Problem Set B

1. Pattern catalogs that enable the classification and retrieval of software patterns at various levels of specialization are quite limited in existence today. Cataloging typically takes into account the architecture domain, the architecture scope, and the degree of specialization. Can you identify additional criteria that would be useful to provide a more comprehensive way to catalog software patterns?
2. Demonstrate how complex architectures may be derived by combining architectural styles.
3. Perform some research on the Internet and identify implementation styles that correspond to implementations of the microservices architectural style.
4. Consider various large companies you know about and state their operating model (i.e., diversification, unification, coordination, replication). Explain your reasoning.
5. Consider creating a vaccine passport as a smart contract on the blockchain platform. Create conceptual business, information system, and technology architecture blueprints to document the architecture of such a solution at a high level.
6. Leverage patterns and create logical business, information system, and technology architecture blueprints for the vaccine passport as a smart contract on the blockchain platform.
7. Leverage patterns and create information system and technology architecture implementation blueprints for a vaccine passport as a smart contract on the blockchain platform.
8. Provide a technical implementation architecture for a vaccine passport as a smart contract on the blockchain platform.
9. What is a software stack and what role does it play in solution architecture management?
10. Research what the term *full stack developer* means. Why would this skill set be useful to an organization? Do you think this skill set provides an opportunity for career development into technical leadership positions?
11. Research commercially available development tools. Provide pros and cons of using an approach that utilizes a commercially available technology stack tool set.

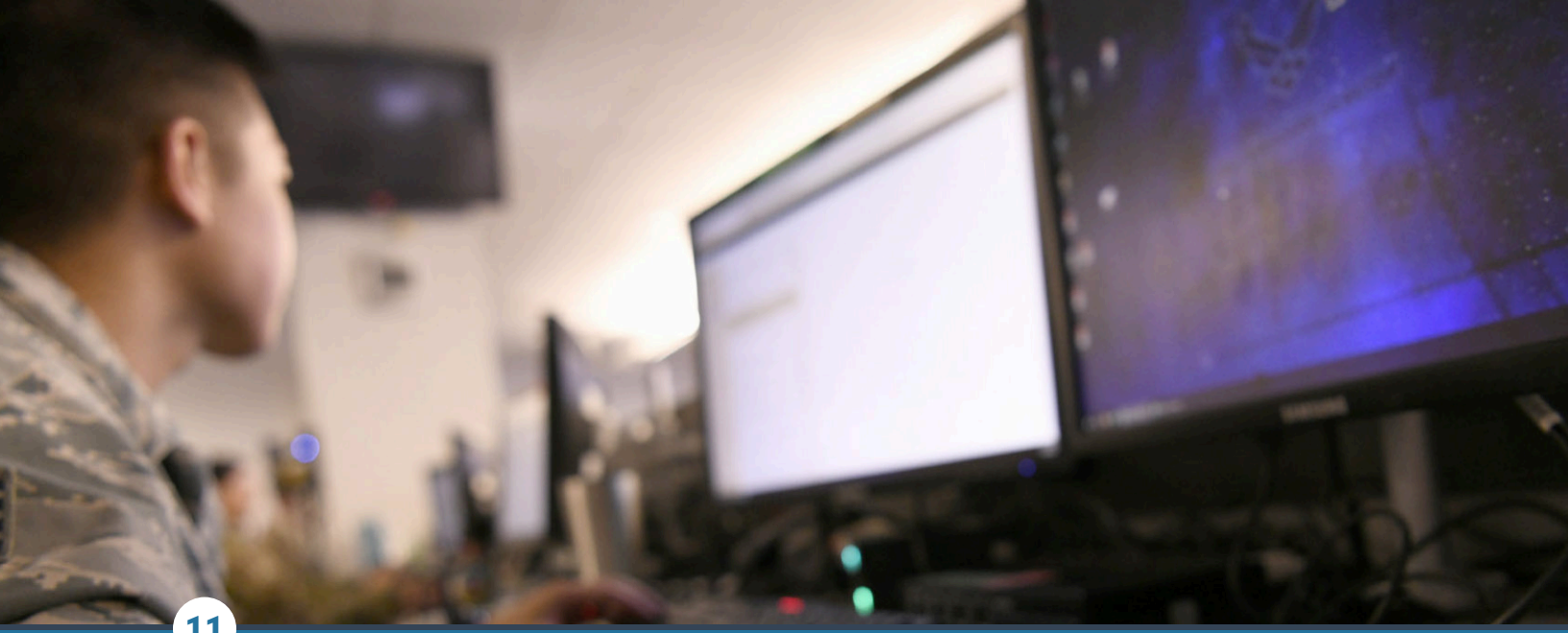
Thought Provokers

1. Consider our start-up company, which is 100% committed to leveraging innovative technologies as a business growth facilitator. Describe how it can best take advantage of patterns (e.g., pattern cataloging) to create products or services that can generate business. Give precise examples and explain how the start-up would be able to ensure the scalability of the resulting business (i.e., keep sustaining the cost of doing business while increasing its number of customers).
2. Consider our start-up company that is 100% committed to leveraging innovative technologies as a business growth facilitator. Describe how it can best take advantage of EAMs and EAFs to create products or services that can generate business. Give precise examples and explain how the start-up would be able to ensure the scalability of the resulting business (i.e., keep sustaining the cost of doing business while increasing its number of customers).
3. Consider our start-up company that is 100% committed to leveraging innovative technologies as a business growth facilitator. Describe how it can best take advantage of solution architecture management

to create products or services that can generate business. Give precise examples and explain how the start-up would be able to ensure the scalability of the resulting business (i.e., keep sustaining the cost of doing business while increasing its number of customers).

Labs

1. Creating a pattern catalog involves systematic documentation and organization of patterns that can be easily referenced and utilized by developers and architects. Search on the Web for the steps to create a corresponding pattern catalog.
2. Search on the Web for blueprints catalog for a web application framework of your choice (e.g., Flask, Django, React). Identify a comprehensive set of criteria that would be useful to provide a more comprehensive way to catalog blueprints.
3. Use the Internet to investigate the use of the Essential EA tool on an enterprise architecture project. Demonstrate how you would use the tool to support all the facets of EAM.
4. Investigate the use of software development IDEs that provide round-trip engineering capabilities and allow the management of implementation patterns as part of the software engineering construction activity (e.g., IBM Software Architect). Compare the various tools available and implement a sample application that leverages the use of round-trip engineering and the use of implementation patterns. How do these tools integrate with EAFs?



11

Web Applications Development

Figure 11.1 It takes many roles to build a responsive design in web applications development for multiple system applications. (credit: modification of "190827-F-ND912-035" by Tech. Sgt. R. J. Biermann/Lt. Col. Wilson/U.S. Air Force, Public Domain)

Chapter Outline

11.1 Modern Web Applications Architectures
11.2 Sample Responsive WAD with Bootstrap and Django
11.3 Sample Responsive WAD with Bootstrap/React and Node
11.4 Sample Responsive WAD with Bootstrap/React and Django
11.5 Sample Native WAD with React Native and Node or Django
11.6 Sample Ethereum Blockchain Web 2.0/Web 3.0 Application

Introduction

TechWorks is creating several web applications this year for a new product line. One application is an AI-image generator website and auction house for selling images. An outside consultant has been brought in, and they have determined that a hybrid Web 2.0/3.0 architecture is best suited for this solution. However, the engineering team who will perform the work needs to gain experience with Web 3.0 technologies. Therefore, the consultant recommended that TechWorks take a popular internal desktop application for managing to-do lists and re-create it as a hybrid Web 2.0/3.0-based application so engineers can gain practical experience with various web application frameworks and technologies, with the added benefit of accessing their to-dos from anywhere.

The TechWorks engineering team has decided to perform iterative releases of the to-do application, starting with responsive web apps, as they will render well on various screen sizes, from large monitors to smaller displays like phones and tablets. Next, they will employ a native web application framework to target specific devices like Android and iPhones. Lastly, they will explore building a Web 2.0/3.0-based to-do application using blockchain technology, as they believe this approach will give them the necessary skills for creating future solutions.

11.1 Modern Web Applications Architectures

Learning Objectives

By the end of this section, you will be able to:

- Understand the use of server-side rendering and MVC patterns
- Relate to the technology used to create responsive Web 2.0 applications
- Become familiar with the technology used to create native mobile applications
- Understand how to create Web 3.0 applications as well as hybrid Web 2.0/3.0 applications

The **World Wide Web**, or **the Web** as it is known today, started as a way to link content (primarily text and images) stored on different servers or machines. It was invented by Tim Berners-Lee in 1989 while he worked as a researcher at the European Organization for Nuclear Research (CERN), home to the European Particle Physics Laboratory. Sir Tim Berners-Lee was knighted by Queen Elizabeth II in 2004 for his pioneering work. He created the Hypertext Transfer Protocol (HTTP) that operates on top of Transmission Control Protocol/Internet Protocol (TCP/IP), the principal protocols used on the Internet. Clients (web browsers) transmit HTTP requests to a **web server**, which is a software application that runs at a Uniform Resource Locator (URL) that specifies a location on the Web to access it, and responds by providing pages rendered in the hypertext markup language (HTML). This simple request and response paradigm, a client-server model, was easy to implement and allowed for the rapid growth of the Web. This phase of the Web, which began after 1989 and ended around 2004, would become known as **Web 1.0**, a period where the user's interaction was limited primarily to reading and selecting web pages. A **web page** is a document commonly written in HTML and viewed in a browser. Figure 11.2 shows a simple Web 1.0 architecture and common usage. An encryption layer was later added to the HTTP protocol, which resulted in creating the HTTPS protocol. This made it possible to protect the sharing of sensitive information over the Web from eavesdropping attacks. While web servers only served web pages initially, a Common Gateway Interface (CGI) was subsequently added after the initial implementation of web servers to make it possible to link to applications via URLs on the Web.

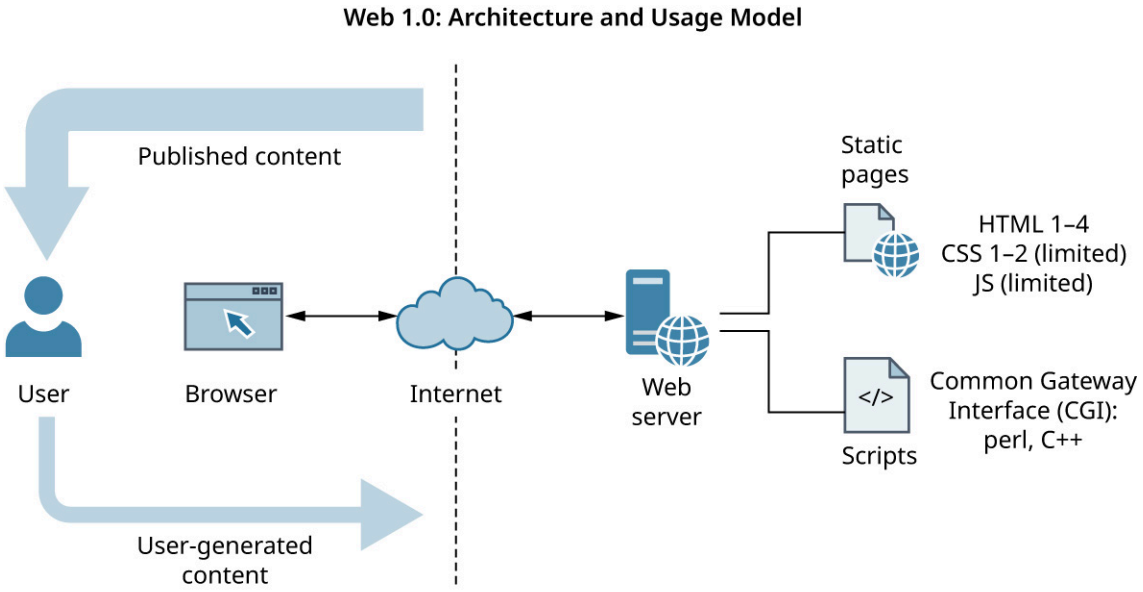


Figure 11.2 This architecture outlines a user's interaction with a Web 1.0 website. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

As the Web evolved from this basic architecture, the need for more dynamic and interactive experiences became apparent. Users were no longer content with simply viewing static pages; they wanted to contribute content and engage with other users. Also called online publishing, **web publishing** publishes content on the Web while applying traditional publishing models. It was akin to digitizing an encyclopedia (images and text)

and putting it online with hyperlinks. Today, simple websites with limited functionality, such as early blogs or static sites, still follow this model. A more interactive model that could scale to meet user demand was needed to support users' desire to provide content and interact with other users. A design pattern is a reusable solution to a common software design problem. Figure 11.3 illustrates the Model-View-Controller (MVC) design pattern that was employed to separate a traditional web application's data model, presentation, and business logic layers into its components.

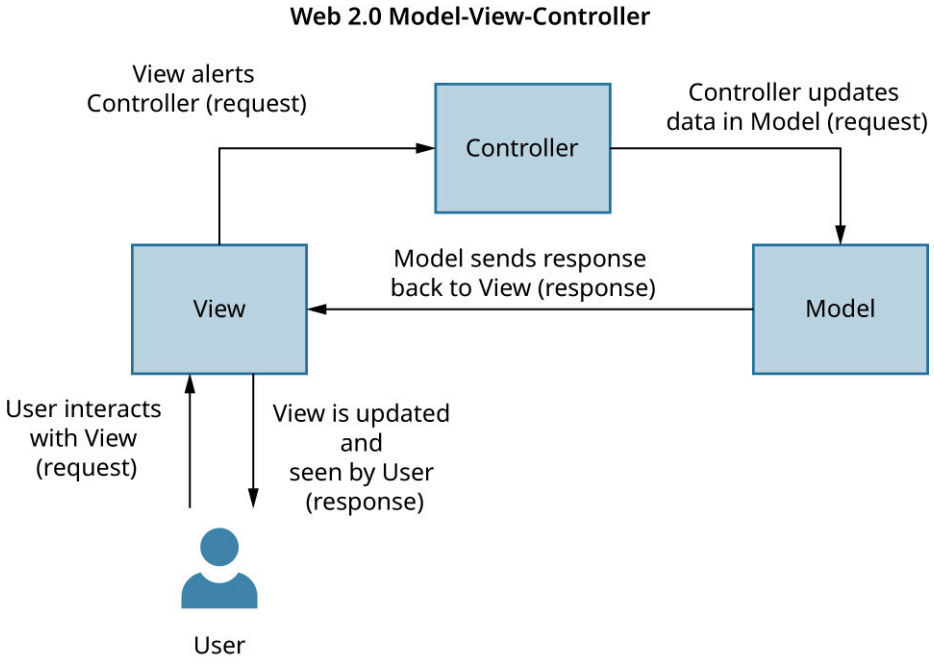


Figure 11.3 This figure shows a user's interaction with the Model-View-Controller pattern. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The original implementation of the MVC pattern on the Web was such that the View would send requests to the Controller and the Controller obtained data from the Model and rendered it within HTML pages that were passed to browsers for presentation purposes. AJAX technology was later introduced to enable a more complete implementation of MVC on the Web that allowed asynchronous updates to page components and did not require refreshing pages in the browser to fetch data.

Following **Moore's law**, which states that the number of transistors on an integrated circuit doubles roughly every two years, the processing power of smaller devices like laptops, tablets, and mobile phones became the preferred way users interacted on the Web. As Internet data transfer speeds and bandwidth increased through better hardware, fiber-optic cables, and mobile wireless technology, the rendering of web applications' interactive interfaces moved from the server side to the client side. This led to being able to run native applications (apps) on mobile wireless phones that could take advantage of specific device rendering features. In a paradigm shift that was opposite to web apps, the data model moved off the phone and onto remote servers. These solutions led to a richer user experience known as Web 2.0, which is a phase of the Web focused on social interactions. This phase started in 2004, and social media websites using Web 2.0, such as Facebook (now Meta) and Twitter (now X), are well-known examples of this web phase of social interactivity.

The next phase of the web, Web 3.0, which is a phase of the Web where user activities may focus on decentralized access to solutions and data, is seeing a shift from the more traditional client-server model to a peer-to-peer networking model. A **peer-to-peer (P2P)** network is one in which devices connect and can share data and processing without needing a centralized server. Peers in this scheme can perform the role of a traditional client, server, or both. This shift fosters a trusted, decentralized, and open web, where large companies do not own the data, but rather where data is collectively owned by users. The use of other technologies like generative artificial intelligence (GenAI), which is powered by machine learning, aims to

improve information access by mediating end-user searches to generate corresponding meaningful information out of the content available on the Web. While the benefits of artificial intelligence and machine learning are vast, enabling enhanced data processing and decision-making, it is equally important to consider the risks. These technologies can pose ethical concerns, such as data privacy issues, biased algorithms, and the potential for misuse, thus highlighting the importance of responsible development and implementation. In particular, while GenAI technology is appealing and can be successful in some cases, GenAI is also known to hallucinate and generate unpredictable and often inaccurate responses to web searches. The exemplar apps of the Web 3.0 and later phases are still to be determined. Still, if the previous phases of the Web are any indication, it will fundamentally change how we operate in an ever-evolving technological world.

Throughout this section, we will discover how the application architectures found in Web 2.0 apps, native mobile apps, and Web 3.0 apps are designed.

GLOBAL ISSUES IN TECHNOLOGY

Importance of Internationalization and Localization in Frameworks

Web applications are used worldwide, and specific internationalization and localization requirements must be observed to facilitate the creation of web applications that people all over the world can use. This can be challenging. To meet internationalization and localization prerequisites, services and products must be flexible to compete in international markets. This includes having the malleability to adapt to the cultural needs of target markets around the world. In global markets, language barriers are just one internationalization and localization issue that must be considered for web applications.[1]

Server-Side Rendering and MVC Patterns

The Web transitioned from its 1.0 phase, where the primary usage was viewing content composed of text and images, to its 2.0 phase of user interaction. As it evolved, three technologies dominated the Web's programming landscape, commonly referred to as the trifecta:

- **hypertext markup language (HTML)**: a standard markup language used to describe the structure and content of web pages.
- **cascading style sheets (CSS)**: a standard style sheet language used to alter the presentation style of the content found in HTML (or other markup languages).
- **JavaScript (JS)**: a scripting language that adds interactivity to web content and server-side functionality. Various other scripting languages and competing approaches were used prior to the adoption of JavaScript.

The adoption of HTML was already a given on the Web, and with the introduction of CSS in 1996, a stronger push for separating content and style was introduced so that the styling of content could be specified solely as part of style sheets rather than HTML tags. Web pages at this time mostly consisted of static content, which would be generated and delivered on the web server. Essentially, the user would select an action in their browser that sent an HTTP request to the server such as the following:

- When clicking a hyperlink on a web page, the browser would send an HTTP GET request to the web server. This request asked for a specific resource (such as a web page or an image), and the server would respond by sending the requested data back to the browser.
- When filling out a form on a web page and clicking the submit button, the browser would send an HTTP POST request to the server. This request included the data that was entered (like a username and password), and the server processed it and responded accordingly.

1 To learn more, check out Lionbridge's blog (https://openstax.org/r/76Lionbridge) post about globalization.

Essentially, the web server performed all the HTML content rendering on the server side, and the client (web browser) would present what it received. The browser in this model acts as a thin client, as it has minimal functionality. Figure 11.4 illustrates the components of a traditional Web 2.0 architecture using Java-based technologies. Notice the shift in user interaction with the website compared with a Web 1.0 website. Also note that users are able to interact with applications via an application server that can retrieve data from a database or file system. This enables support for managing web sessions that allow navigation across multiple pages.

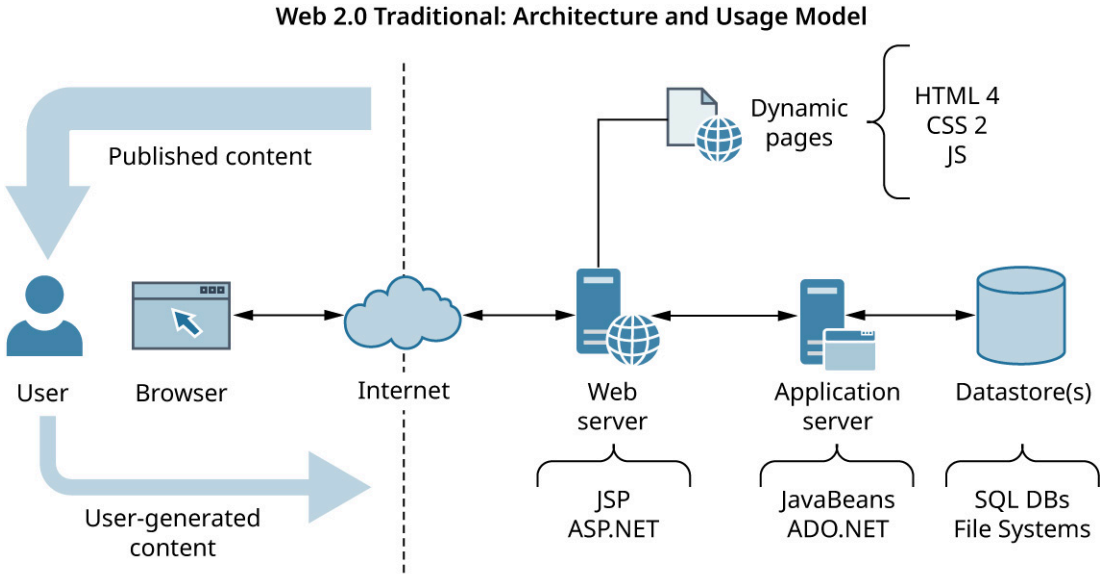


Figure 11.4 This illustrates a user's interaction with a traditional Web 2.0 architecture. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Many of the original web applications were stateless, meaning that prior requests had no bearing on future requests. For example, it was not possible to create a shopping cart as part of a web session that would keep track of the session and what was purchased on a site and maintain the state (i.e., content in this case) of the cart. In contrast, a **stateful application** is software that maintains the state of an application over time, while in the case of a **stateless application**, state is not maintained by the system and previous actions do not impact future ones. Stateless applications are simpler and easier to implement and maintain but offer limited functionality.

As previously mentioned, Web 2.0 was partly driven by user demand for more interactive functionality. Interactivity requires maintaining some state (e.g., the web session and the content of the cart as per the previous example), thereby increasing the system's complexity. This increased the demands on the web server for almost all the processing needed to generate and present the website content. Increased functionality led to more complex systems, and a clear separation of responsibility between the website's rendition, business logic (i.e., the logic implemented as part of the web application), and persistence layers were needed to improve the quality and performance of the website while leveraging engineering expertise in given domains.

As you learned in Chapter 10 Enterprise and Solution Architectures Management, the Model-View-Controller pattern is tailored to address this separation of responsibility. In the MVC pattern, the Model is the persistence layer responsible for data storage and retrieval. It has a well-defined API that the Controller uses. The **View** is the presentation layer that handles the user interface. Finally, the Controller acts as the business logic layer that performs processing and enforces rules to generate applicable content within a given application domain and separates the user interface from the data. It also has a well-defined API that the View understands. The best-practice design concept used to create software systems in a way that ensures weak associations with other components is called **loose coupling**. This concept allows for separation of concerns between components, which leads to maintaining high cohesion within websites' functionalities. MVC components are loosely coupled in that the various components can interact with one another to access the specific

functionalities provided by each component. High cohesion ensures that everything that is needed to provide a specific functionality is included in one of the components. For example, on a banking website, functionality for deposits and withdrawals may be collocated on the server within the same component to ensure high cohesion; however, features for applying for a loan may be located within another component. Three popular server-side web application frameworks that implemented the MVC pattern were Apache Struts, ASP.NET, and Ruby on Rails.

CONCEPTS IN PRACTICE

APIs Play a Large Role in Web Services

Understanding APIs and their protocols helps determine which platform is being accessed and at what level. APIs allow access to any software with a specific purpose or functionality. APIs can be used as contracts between multiple applications.

How do APIs work? The API architecture is usually explained in terms of client and server. The application sending the request is called the client, and the application sending the response is called the server. For example, in the case of an API for a weather service, the weather service database is maintained on the server side, and the mobile app is running on a client mobile device.

The server side performs the majority of the functionality. It uses a combination of templated HTML, controller and application server technologies (e.g., ASP.NET, C#), and SQL. JavaScript is sent to the browser using jQuery for cross-browser support, or it may use Asynchronous JavaScript and XML to communicate with the web server without refreshing the page. **Asynchronous JavaScript and XML (AJAX)** exchanges small amounts of data between a client and server. The engineering team using such a web platform must understand and enforce the separation of responsibilities between the MVC components to ensure future modifications, especially significant changes, can be made without rearchitecting the system. Because the engineering team needs to know different programming languages for different layers, there may be some internal resistance to this, and it may be tempting to go around a layer to make a quick fix. Essentially, making sure that the architecture of the web platform is understood and used properly, architectural adherence is predicated on the expertise of the engineering team members and more on the nature of the tools used.

Responsive Web 2.0 Applications

By the mid-2000s, JavaScript became popular for client-side browser functionality. AJAX and jQuery technologies allowed developers to create single-page applications (SPA). For the end user, SPAs offered functionality like a traditional desktop application and drastically improved user experience by reducing latency. No longer were expensive web server requests needed that resulted in full page refreshes in the browser. Instead, small data transmissions could be sent and received between the browser and server through AJAX. Figure 11.5 compares the life cycle of SPAs to a traditional Web 2.0 application.

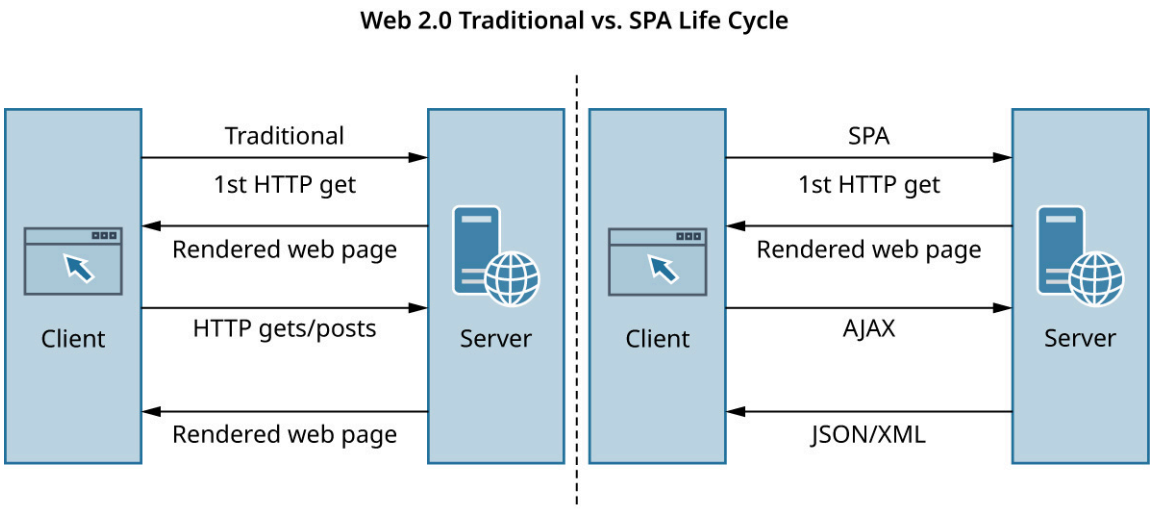


Figure 11.5 This is a comparison of the life cycles between traditional Web 2.0 applications and SPAs. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

LINK TO LEARNING

The World Wide Web Consortium (W3C) (https://openstax.org/r/76W3C) develops standards and guidelines for the Web. You can discover more about them and examine some of their current draft standards.

jQuery, an open-source JavaScript library, ensured that web developers could write JavaScript for a generic browser document object model (DOM) that would run regardless of the user's browser. The DOM is a programming interface provided by browsers. It allows scripts, written in JavaScript for example, to interact with the structure of a web page. When a web page is loaded into the browser, the browser creates a DOM of the page. The DOM structure is a hierarchical treelike structure that organizes the elements of the page as objects. The model used by the DOM enables dynamic access and facilitates the manipulation of content, structure, and style of web pages. Figure 11.6 illustrates a typical SPA architecture where a single web page is delivered to the browser. Note that in this model, the user's interaction with the site has increased in the amount of content generated by the user.

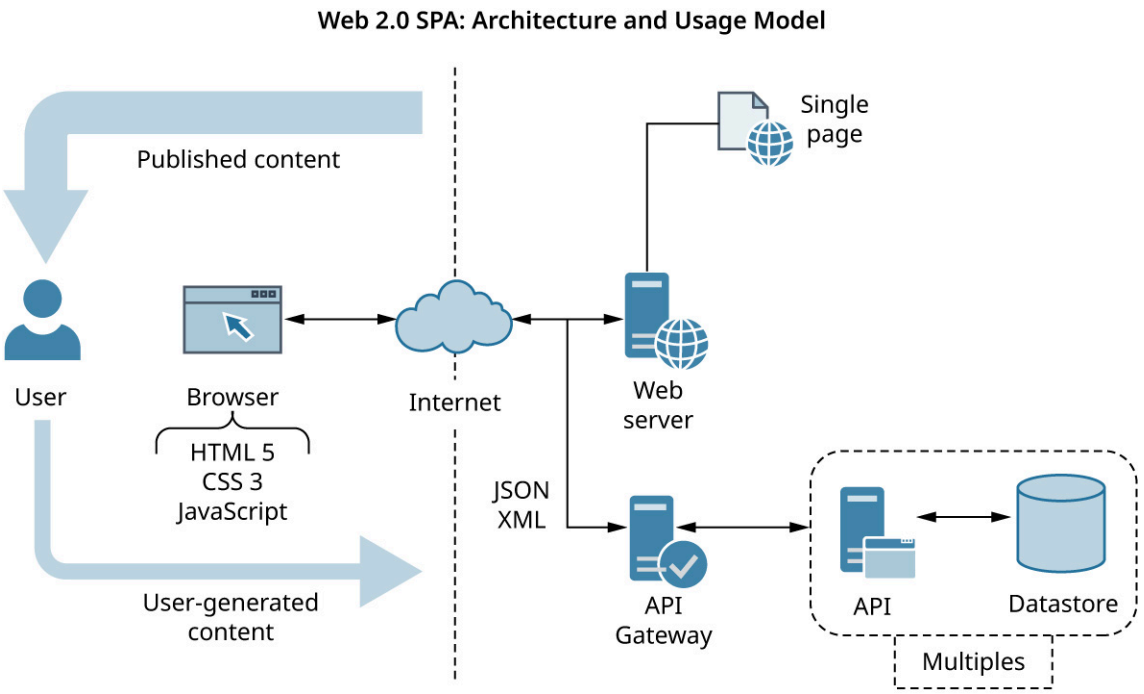


Figure 11.6 This illustrates a user's interaction with a single-page application (SPA). (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The more sophisticated SPAs required large amounts of JavaScript on the client side. Many end users had underpowered machines or out-of-date web browsers, and the SPAs performed poorly.

In the early 2010s, web application frameworks were introduced to create complex, client-side web applications that performed well, gave a native desktop application-like experience, and were easier for developers to create. These applications followed a Model-View-ViewModel (MVVM) pattern. Figure 11.7 illustrates the relationship between the View, ViewModel, and Model components.

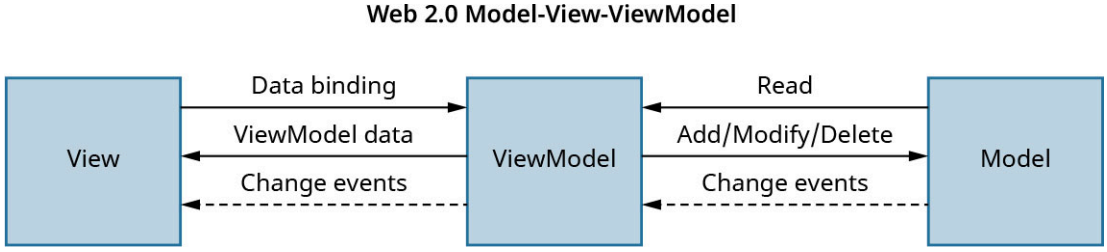


Figure 11.7 Various data binding, events, and actions occur between the components of the Model-View-ViewModel pattern. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

This pattern is like the MVC pattern we've previously explored; however, several key differences exist. The following are some of the similarities and differences:

- The View is responsible for the presentation and only interacts with the ViewModel. This is similar in responsibility to the View role and interaction with the Controller in the MVC pattern. The View here binds to functions and properties in the ViewModel and receives notifications on operations and changes to the data. It doesn't interact directly with the Model.
- The Model is similar to the Model in the MVC pattern and is responsible for data retrieval and storage. It doesn't know anything about the ViewModel.
- The ViewModel is similar to the Controller in that it decouples the relationship between the View and Model and handles data manipulation. The ViewModel responds to notifications from the Model and will send events to the View if needed. Because the View binds to the ViewModel, the ViewModel doesn't know anything about the View. The ViewModel can work with a local Model in the browser, a remote Model, or

both.

Unlike the previous MVC pattern regarding server-side rendering, the MVVM pattern is run entirely in the client. A Representational State Transfer (REST) API decouples the client-side Model and ViewModel components from the server-side business logic and persistence store. REST-based (aka RESTful) APIs follow the architecture style designed for the Web. These APIs use the **JavaScript Object Notation (JSON)** file format that represents data as text-based attribute-value information. Figure 11.8 illustrates the MVVM pattern as it applies to an SPA.

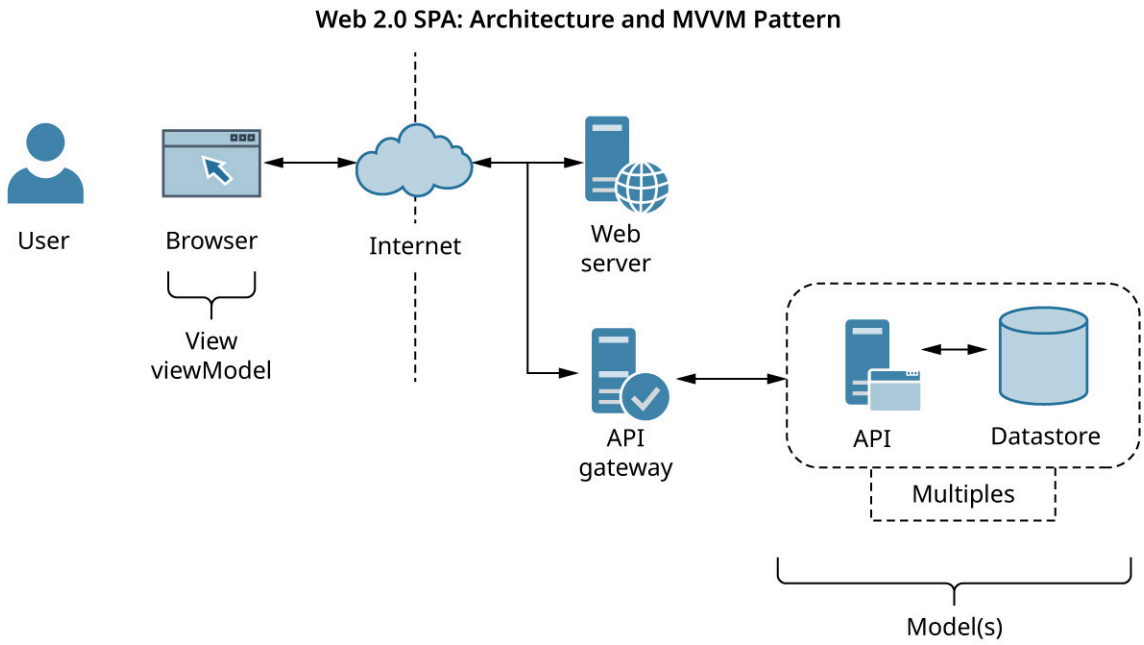


Figure 11.8 This illustrates the Model-View-ViewModel pattern as applied to the SPA architecture. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

As you can see from the diagram, this model is more complex than the simpler MVC pattern. The quality of the APIs partly determines the effectiveness of this pattern. REST-based APIs benefit from being discoverable (i.e., all API URIs can be found from the root API node) and should be versioned (i.e., to keep track of changes to interfaces for compatibility purposes) so that upgrades don't break API users. Without API versioning, coupling between client and server increases (due to semantic and syntactic coupling), and upgrades become costly. Similar to a URL, a Uniform Resource Identifier (URI) is a string of characters that identifies a resource on the Web.

Regardless of these flaws, early browser-based frameworks proved the value of creating rich client-side apps. Web standards evolved, and newer frameworks emerged that adhered to the newer standards and solved many of the problems of their predecessors. Popular SPA frameworks include Angular, Ember.js, React, and Vue.js. These are often used in conjunction with server-side tools, creating a "full stack" of technologies for developing the solution. A popular server-side technology in this stack is Node, a runtime environment for executing JavaScript code.

Approaches to minimizing the data transferred between the APIs and the client to reduce network traffic or simplifying access to the data resulted in two primary approaches. One was flattening the data model. Instead of returning data in a nested format (such as from a relational database), the returned data would be "flattened" to a series of key-value pairs, each resulting in a unique return value. If the specific request could be made, the flattened data would be preferred to navigating the nested data to improve performance. However, creating APIs to fit specific requests could be challenging. Another solution was using **GraphQL**, an open-source query and manipulation language. With GraphQL, callers of its API could craft specific requests to return only the needed data, which made access to many services such as the ones provided by microservices

architectures more scalable.

THINK IT THROUGH

Framework Selection

Given that many web and native application frameworks are available today, is there a process that facilitates the selection of these frameworks?

Native Mobile Applications

Native mobile applications are designed to run on specific mobile operating systems (i.e., Android, iOS). Android and iOS operating systems dominate the mobile device market. In 2023, Android had approximately 70% of the worldwide market share, with iOS at around 29%. Samsung, KaiOS, Windows, Linux, and others make up the remaining 1% of the market.[2]

However, in the United States in 2023, iOS was the predominant OS with roughly 61% of the market and Android at 38%. Here, we'll look at the specifics of native mobile app development by focusing on Android and iOS.[3]

Native Application Development for Android

Native application development takes advantage of the specific features available on mobile devices. The creators of the Android operating system (OS) originally designed it as an OS for digital cameras; however, they soon pivoted and changed it to be an OS for smartphones. Today, the Android OS runs on most mobile devices worldwide. It can be found on the Google Pixel, Samsung, OnePlus, and other phones.

Developers for Android mobile apps have a rich ecosystem composed of development tools, programming languages, training, and services. The primary developmental tools are:

- **Android Studio** is the official IDE for Android development, built off JetBrains' IntelliJ IDEA software.
- **Jetpack Compose** is a toolkit for building native user interfaces (UI).
- **Firebase** is an app development platform and a collection of services for authenticating users, integrating ads, running A/B tests, and more. Firebase includes an A/B testing tool that helps test changes to web apps to see how changes impact key metrics such as revenue and customer retention. Developers will likely write software in Java, Kotlin, or C++. Java was historically the primary language for Android development. It runs on the Java Virtual Machine (JVM), which allows for creating platform-independent software. Kotlin is the preferred Android development language. It runs on the JVM but also runs on the Android Native Development Kit (NDK), which is used for performance-critical parts of applications. The NDK performs better than the JVM, and it provides direct access to physical device components (e.g., sensors and touch screen). C and C++ code can also run on the NDK.

Applications developed for Android are distributed through App Stores. Google Play is the principal app store, but others include Amazon Appstore, HUAWEI AppGallery, and Samsung Galaxy Store.

LINK TO LEARNING

To explore Android app development further, Google offers free training. You can get started by creating your first "Hello World" (https://openstax.org/r/76AndroidDev) Android program.

2 Statcounter Global Stats, "Mobile Operating System Market Share Worldwide, Dec 2022–Dec 2023," January 2, 2024. https://gs.statcounter.com/os-market-share/mobile/worldwide/2023
3 Statcounter Global Stats, "Mobile Operating System Market Share United States Of America, Dec 2022–Dec 2023," January 2, 2024. https://gs.statcounter.com/os-market-share/mobile/united-states-of-america/2023

Native Application Development for iOS

Apple's iPhone was released in 2007 and quickly dominated the mobile phone market in the United States due to its simple, intuitive touch screen interface based on the success of its music player, the iPod, contributed to its popularity. The iOS operating system runs Apple's device ecosystem, which includes iPhones, Apple TVs, Apple Watches, and more.

Xcode is the primary IDE used for all Apple device development. Developers will code in Objective-C or Swift. Objective-C was the primary development language for iOS; however, it was challenging to learn. In 2014, Apple released the Swift programming language specifically designed for iOS development. It offers high-order language features, making it easier to develop software and built-in memory management, which makes less prone to crash.

INDUSTRY SPOTLIGHT

Framework Selection Based on Industry

Native application frameworks are important in every industry today. For example, mobile health applications have become quite popular. Smartwatches can collect health-related data using various sensors and mobile health applications can leverage that data to provide useful reports to users as they exercise. Using a native application framework provides quicker load times, smooth animations and transitions, optimal security, and a seamless user experience, to make it easier to immediately track and access health-care data.

Drawbacks to using native application framework include a longer development process, increased cost, complex maintenance and updates, platform dependency, regulatory and compliance issues, and end-user barriers.

The Benefits and Drawbacks Between Native, Web, and Hybrid Mobile Application Development

Native applications have the benefits of accessing device-specific hardware and sensors (e.g., camera, accelerometer), data (e.g., location, contacts), and are optimized for performance.

However, they have the drawbacks of being device-dependent and available primarily through proprietary app stores (Google Play for Android and Apple's App Store for iOS). Developers need to learn different languages and libraries for the devices. However, cross-platform development is possible with frameworks (e.g., Flutter, Kotlin Multiplatform Mobile), allowing developers to code in a single language and run solutions on both platforms.

Web apps can also run in a browser on mobile devices. They have the distinct advantage of responsiveness and can run on various screen sizes—thus allowing for a single codebase that can increase productivity and reduce cost. Disadvantages to web apps on mobile devices are limited access to hardware and software features, lower performance than native apps, and web apps may perform differently depending on the mobile browser. Traditionally, web apps didn't look like native apps. In 2017, Google introduced Progressive Web Apps for Android, allowing web apps to look and feel similar to native apps.

Finally, hybrid apps are web apps wrapped inside a native device framework (e.g., Apache Cordova). These apps have the advantages of using traditional web application development while accessing and utilizing device functions and running on multiple mobile platforms. The drawbacks are reduced speed and potential security vulnerabilities found in the framework.

CONCEPTS IN PRACTICE

Software Patterns and Frameworks

As discussed in Chapter 9 Software Engineering and Chapter 10 Enterprise and Solution Architectures Management, architectural styles, architectural patterns, and design patterns are typically used to enforce the quality attributes of software solutions. To facilitate the creation and maintenance of web applications and scalable transactional websites, web application frameworks and native application frameworks that leverage applicable patterns, such as MVC, were created and improved over the past couple of decades. Web frameworks are used within web application frameworks to facilitate the use of HTML5, CSS3, and JavaScript and to publish responsive web pages that can be rendered by modern browsers on all modern target devices. Web application frameworks help process UI-driven requests that result in publishing or updating web pages. Native application frameworks take advantage of capabilities available on specific target devices such as iPhones and Android phones. Organizations in many industries rely on web applications and related frameworks to conduct business daily.

Web 3.0 Applications

Web 3.0 is the next phase of the Web. It is still being realized, and similar to Web 1.0 and 2.0, it can only be fully understood in hindsight. The technology and uses of the Web overlap between the phases, with newer phases becoming preferred over their predecessor phases. If Web 1.0 was the read-only web, and Web 2.0 was the participation (write) web, then Web 3.0 may be the read, write, and execute web.

Tim Berners-Lee had referred to Web 3.0 as the **Semantic Web**, a system of autonomous agents, which are software programs that respond to events. That model has shifted over the years. While AI (and other technologies such as AR/VR) will likely form a part of Web 3.0 or Web *x*.0, the principles that govern the next phase are expected to be around a web that is decentralized, permissionless, and trusted.

Web 3.0 sees a shift from the more traditional client-server model to a peer-to-peer networking model. A peer-to-peer network is one in which devices connect and can share data and processing without needing a centralized server. Peers in this scheme can perform the role of a traditional client, server, or both. This shift will foster a trusted, decentralized, and open web, where large companies don't own the data, but everyone collectively owns it. Technologies such as a smart contract allow a trusted model that is needed in a decentralized system. Artificial intelligence and machine learning will improve information access through understanding the meaning of content available on the Web. The exemplar apps of the Web 3.0 phase will be defined in the future. Still, if the previous phases of the Web are any indication, it will fundamentally change how we operate in an ever-evolving technological world.

Finally, Web 3.0 apps will run on a web that supports Web 1.0 and 2.0 apps, with the likely result being hybrid architectures that are partially Web 2.0 and 3.0.

Web 3.0 Application Architectures

Web 3.0 apps are architected without a centralized server, so they are considered to be **decentralized Apps (DApps)**, which are applications that execute smart contracts and run over distributed ledger technology (DLT). Smart contracts are programs that can be executed on a distributed ledger. Distributed ledgers use independent network nodes to record, share, and synchronize transactions in their respective electronic ledgers instead of keeping them in a centralized server. The code is visible to all parties, and trust is put into the code instead of a centralized third party. Distributed ledgers chronicle transactions between accounts that are recorded in multiple places. Blockchain is a type of DLT.

Let's compare a traditional Web 2.0 application with a 3.0 DApp. In Web 2.0, if you created a website for users to post pictures and make comments on them, you would need functionality for authenticating users,

authorizing their actions, generating a UI for adding images, browsing images, and posting comments. To do this, you would need a front-end web server and back-end processing for business logic and data storage. These servers would run in a company's data center or on a cloud provider. The company would own all the code and data, including metadata about how you interact with the website (e.g., what you commented on, how long you looked at a picture). As a reminder, Figure 11.4 illustrates a traditional Web 2.0 architecture (note: for our purposes, the SPA architecture could also be used here).

In a Web 3.0 DApp, the functionality for creating an application for posting pictures and commenting on them would remain the same. The principal differences would be where the code runs and who owns the code and data. Let's break this down into a series of steps that shift away from a centralized solution to a distributed one.

To start, we can keep the web server's front end and replace the back-end code with the distributed application. The Ethereum blockchain, as an example, is a deterministic state machine that runs on a peer-to-peer network. A deterministic state machine has a model of computation that relies on a finite number of states and transitions between them to respond to inputs. It guarantees a single transition from one state to another in response to an input. Because it is based on a deterministic state machine, the Ethereum blockchain is often referred to as a "world computer." Ethereum blockchain transactions are calls to methods that are implemented in smart contracts. These method calls result in change to the data (aka, state) that is maintained by the contracts within the blockchain. The Ethereum blockchain records transactions into blocks that are added to its blockchain. Changes to the state machine itself (e.g., modify the block or transaction data structures used in the Ethereum blockchain) require consensus between the node providers that support the peer-to-peer blockchain network. Anyone in the world can read or write to the machine—a central authority does not govern it.

Smart contracts form the code and are written in high-level languages like Solidity or Vyper. The high-level contract code is compiled into bytecode that can be executed on the Ethereum Virtual Machine (EVM). Each block contains a hash pointer of the previous block, with a time stamp and the transaction data. The hash pointer includes a cryptographic hash (i.e., a fixed-size digest of the previous block that makes it possible to verify that the previous block was not changed since it was stored in the blockchain; the cryptographic part ensures that the hash value has some special properties and, in particular, that the original data cannot be derived from the hash value).

Figure 11.9 replaces the traditional back-end components of a website with a blockchain. The user's interaction with the website is consistent with a traditional Web 2.0 website, as the back-end processing is hidden from the user, though it has been replaced with a blockchain node.

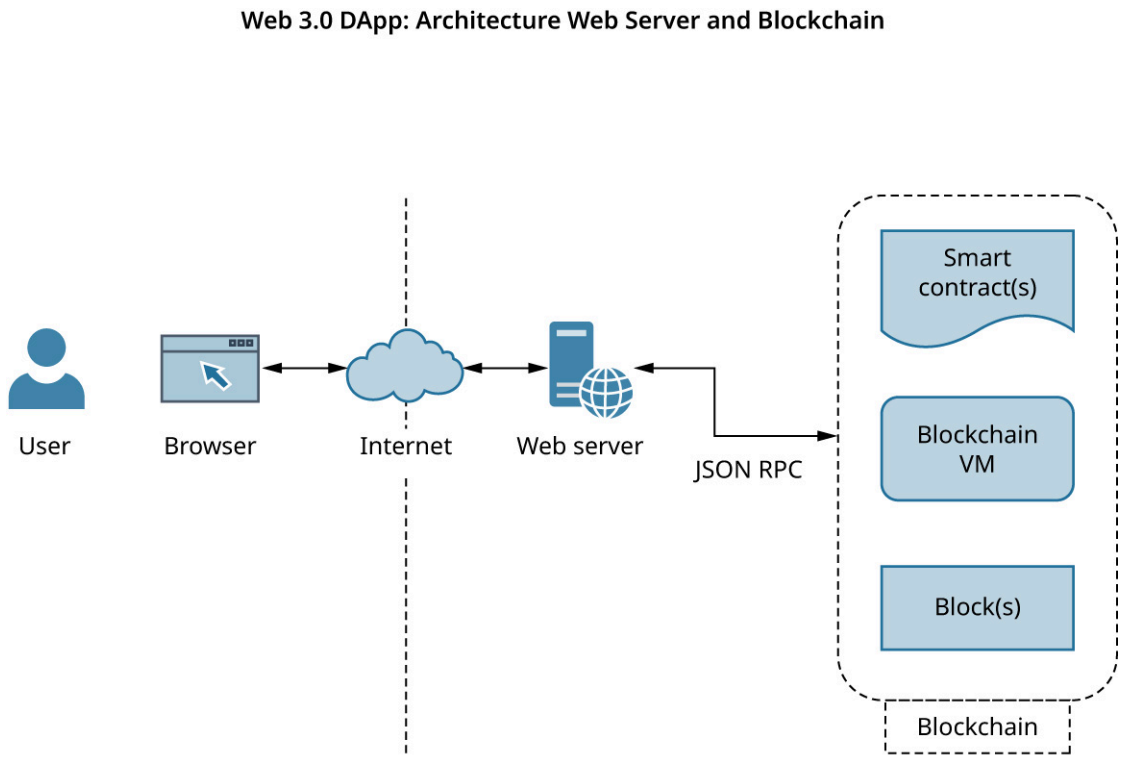


Figure 11.9 Here is a traditional Web 2.0 website with back-end components replaced with a DApp. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

In the figure, the web server talks to a node of the blockchain, referred to as a full node. A **full node** is a computer that maintains a copy of the blockchain and runs blockchain software. Operating a full node can become expensive because you need to provide the hardware and pay a fee to join the Ethereum network. Blockchain end users typically use a software wallet such as MetaMask to access nodes in the blockchain or use a third-party wallet offered by providers like Infura, Alchemy, or QuickNode. MetaMask is a wallet technology that stores a user's private keys in their browser.

Blockchain providers use the JSON-RPC specification for managing communication from clients/end users. This remote procedure call (RPC) is lightweight and transport agnostic.

As previously mentioned, any client can access the blockchain via a wallet provider, which requires creating an account and obtaining a wallet ID. Once logged into their wallets, clients can perform blockchain transactions, which end up reading and/or writing to the blockchain and changing the state of smart contracts. The wallet ID is used to sign transactions so they can be traced to the client wallet. In our example, browsing photos on the Web 3.0 application is a read transaction that would not require signing anything; however, adding photos and comments would.

Adding data to the blockchain incurs a cost as nodes now need to store that data. The user incurs this expense. Imagine paying every time you upload a photo to your favorite social media app. Instead of storing the data on the blockchain, a cost-effective approach would be to store the data using the InterPlanetary File System (IPFS) protocol. IPFS is a peer-to-peer distributed file-sharing protocol. You can go through a provider like Pinata to get a hash value for the data (i.e., a fixed length digest of the data that cannot be used to obtain the original data) you upload to the IPFS and store that hash value in the blockchain via a smart contract interface, thereby reducing the storage cost of data within the blockchain.

Figure 11.10 shows the addition of:

- the provider between the web server and the DApp
- a signer for when the user wants to write to the application

- the inclusion of the IPFS for data storage

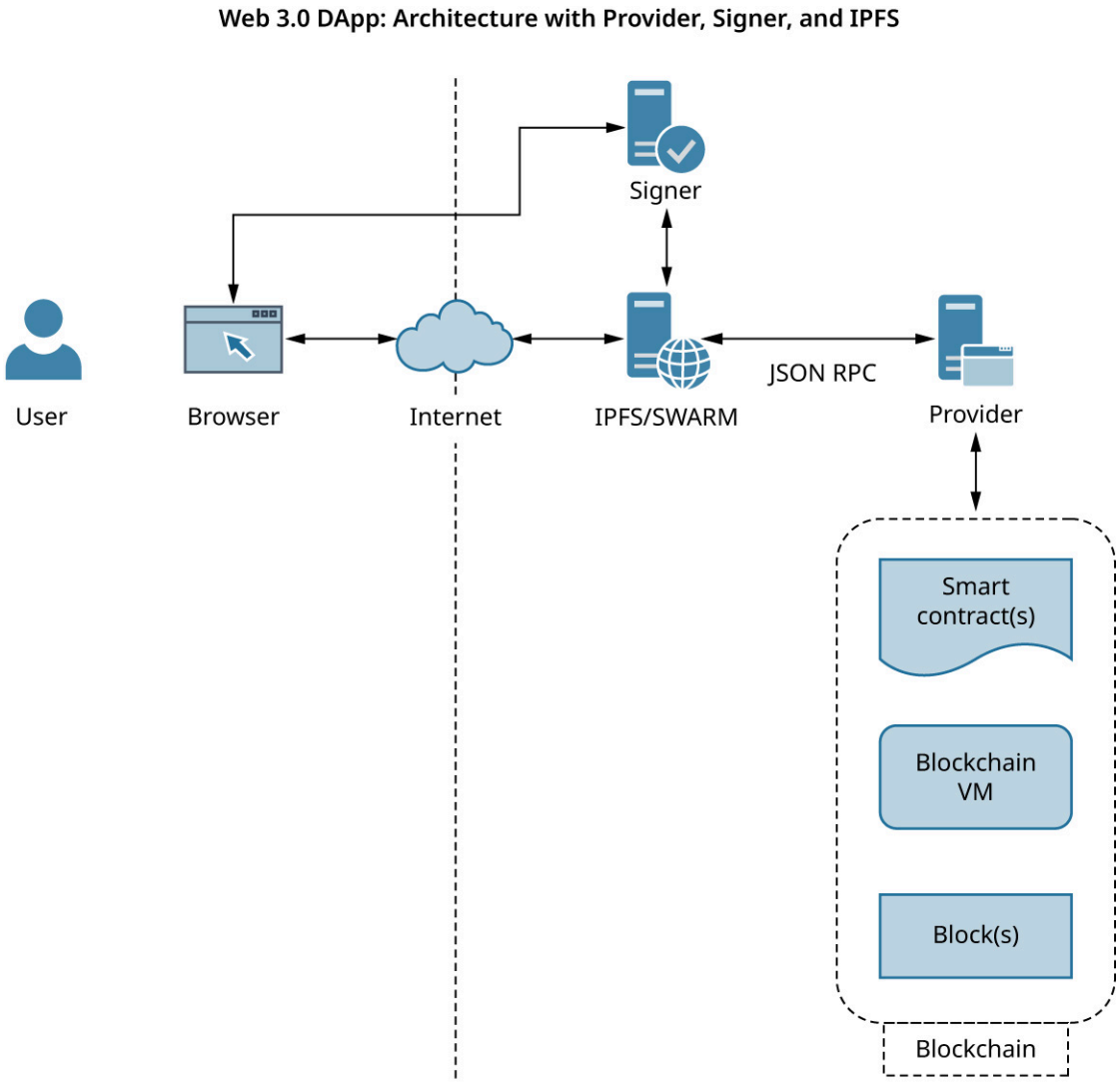


Figure 11.10 This architecture has added a signer, provider, and IPFS. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Up to this point, we've kept the front-end logic on a web server hosted in a centralized location. This is a good temporary approach for an organization that wants to transition a legacy solution to a DApp over time. Because IPFS will host some of the data of the DApp, let's move the front-end HTML, CSS, and JavaScript there and load it into the browser like an SPA app.

Like an SPA, we want the application to run asynchronously and respond to events (like changing data) as they occur. Web3.js is a JavaScript library that uses the JSON-RPC to respond to events fired when smart contracts execute. Alternatively, The Graph is a solution that uses GraphQL to make it easier to query data on the blockchain.

Figure 11.11 shows the updated architecture with the front-end residing in IPFS and the addition of The Graph (listed as GraphQL) for improved querying of the blockchain.

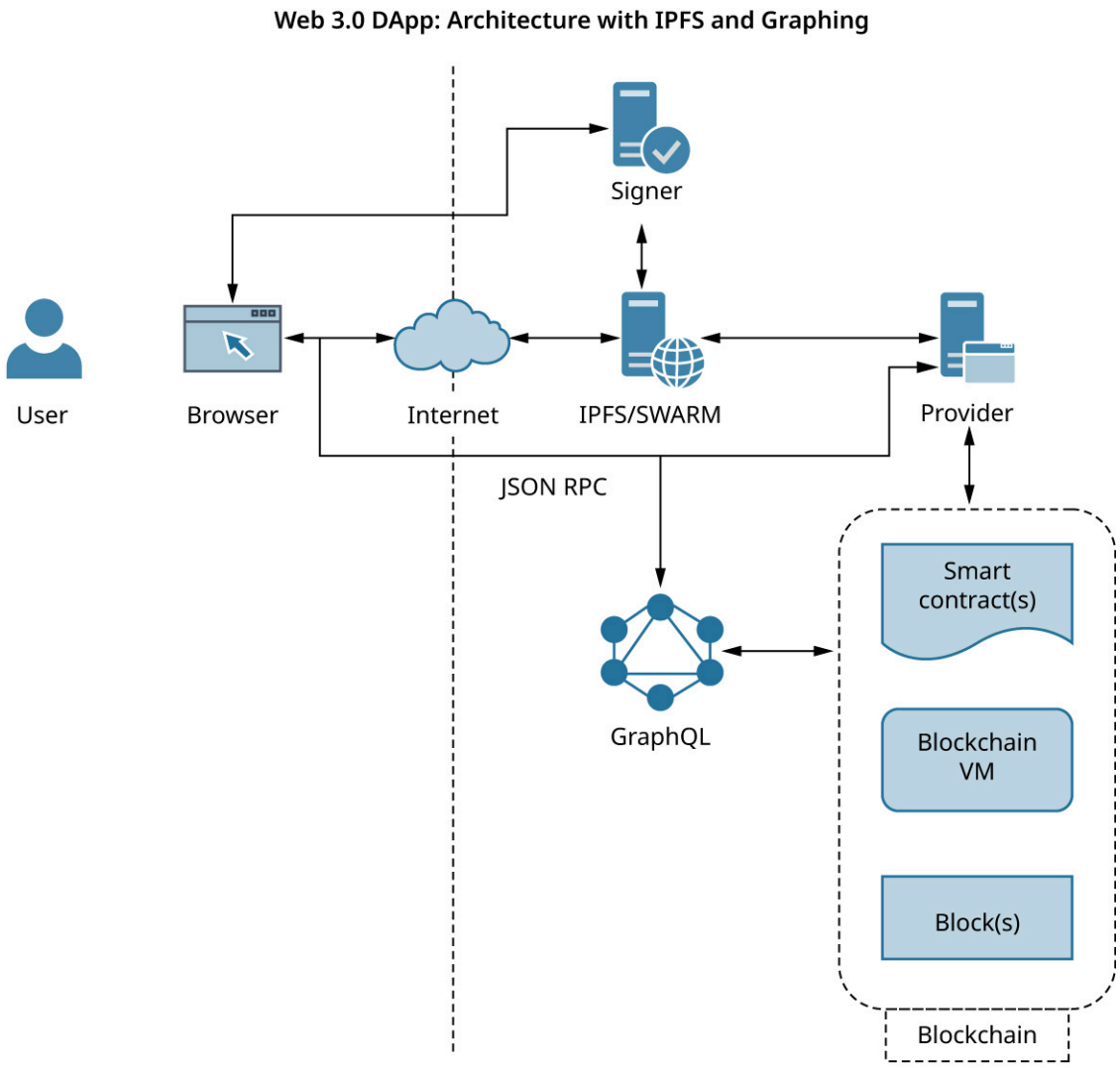


Figure 11.11 The front-end web page has been moved to IPFS, and graphing query capabilities have been added. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

This results in a Web 3.0 application; however, there are problems with this architecture that we will examine next.

Web 3.0 Application Challenges

Blockchains currently have a scaling problem. The average size of a block has increased over the years, and transaction fees are associated with each transaction. A common fee is the **gas price**, which is the cost of validating transactions and updating ledgers. Additionally, negative performance can occur on a blockchain with large amounts of transactions.

One approach to help with scaling is the use of sidechains. A **sidechain** is a secondary (i.e., level 2 or L2) blockchain that increases the blockchain network performance by aggregating transactions off-chain (i.e., off the mainnet/Ethereum network) and committing them to the mainnet at once. Polygon is a popular L2 scaling system for sidechaining. Other L2 scaling techniques include blockchain rollups, which are protocols designed to enable high throughput and lower costs. They address scaling by bundling transactions and reducing data sizes to increase transaction efficiency and limit storage costs. Examples of blockchain rollups include optimistic and zero-knowledge rollups. An **optimistic rollup** is a protocol that increases transaction output by bundling multiple transactions into batches, which are processed off-chain. In this case, transaction data is recorded on the mainnet via data compression techniques that lower cost and increase transaction speed.

Optimistic rollups on Ethereum can improve scalability by a factor of 10 to 100. A **zero-knowledge rollup (zk-rollup)** is a protocol that bundles transactions into batches that are executed off the mainnet. For every batch, a zk-rollup operator submits a summary of required changes once the transactions in the batch have been executed. Operators have also produced validity proofs that demonstrate that the changes are accurate. These proofs are significantly smaller than transaction data so it is quicker and cheaper to verify them. Additionally, zk-rollups are used on Ethereum to reduce transaction data size via compression techniques, which ends up reducing user fees.

Figure 11.12 shows the addition of sidechains to the architecture to mitigate some of the challenges associated with Web 3.0 DApps. Notice how the user's interaction with the Web 3.0 app has grown from mostly reading content in a Web 1.0 model to more fully participating in the content and functionality of the Web 3.0 app.

Web 3.0 DApp: Architecture and Usage Model

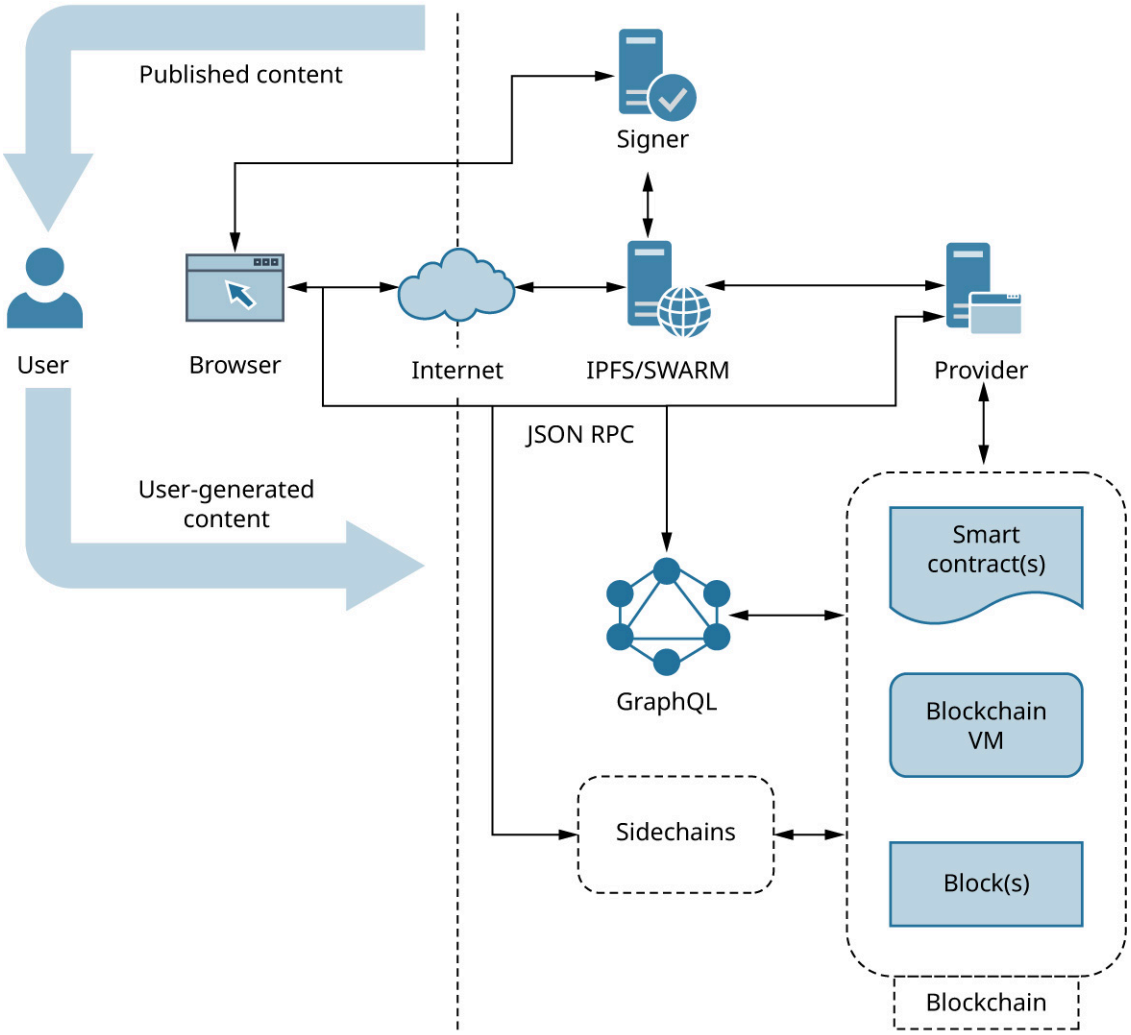


Figure 11.12 The user's interaction with the full Web 3.0 application is highlighted here. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

As you can see, the architecture has become rather complex to support DApps effectively. However, Web 3.0 is still in a nascent phase, and currently there are several solutions and tools being developed to make building and deploying Web 3.0 DApps even easier. Hardhat is a developer ecosystem for building and deploying smart contracts on a local network for testing.

This architecture focused on blockchains and used Ethereum as the principal implementation. However, there are other DLT solutions. Hashgraph is an approach where only selected nodes store the entire blockchain, and voting mechanisms are introduced to validate if the blocks are correct. Stacks is another DLT where only the smart contracts are decentralized, and the data is controlled by its owner. Owners can share or remove it—ensuring data privacy.

TECHNOLOGY IN EVERYDAY LIFE

Use of Frameworks

Web applications are used today to power commercial websites that are accessed by people to complete online transactions to buy goods of any kind. Mobile web applications and native versions of such are also available on smartphones and watches to help do the same. How does using web and native application frameworks help people in everyday life? Provide a couple of illustrative scenarios to explain your opinion. Your scenarios should not be limited to describing how the frameworks are used, but rather describe situations where these frameworks are applied in real life.

Hybrid Web 2.0/3.0 Applications

Due to the qualities and limitations of Web 2.0 and 3.0 architectures, it is likely that we'll see solutions that are a combination of the two approaches and leverage their best attributes. One way to do this would be to have workflow processes that require a high rate of change implemented in a Web 2.0 architecture, while processes that would benefit from using a distributed ledger being executed in Web 3.0. For example, traditional commercial websites can keep using the Web 2.0 architecture, while payment solutions can be extended by accessing the Web 3.0 architecture to make payments using cryptocurrencies (e.g., Bitcoin).

Let's consider an app for generating AI artwork. Many solutions exist today for doing this; however, we want our app to give "ownership" to the digital artwork a person generates. AI models are trained to recognize existing artwork (e.g., paintings available via the wikiart.org API) that may be copyrighted—so ownership is still being determined in the courts. Here, the term *ownership* is used to attribute AI image creation and nothing more. While AI models are very popular, some companies do not want to share data with AI model creators and prefer to have the AI models deployed in their own infrastructure to ensure full privacy. A **non-fungible token (NFT)** is a unique digital identifier on a blockchain that a user may want to create of their image and possibly sell (i.e., transfer ownership) on a marketplace. A common use case for this app might be:

1. A user logs in to the website and enters a prompt to generate an image.
2. After several rounds of adjusting their prompts, they end up with an image they want.
3. The user pays to have a non-fungible token created of the image, which is stored on a blockchain marketplace for sale.

As you can imagine, running an AI image generator on a blockchain might not perform well. Likewise, creating NFTs without blockchain technologies is counterintuitive. Therefore, the architecture for this solution needs to encompass both Web 2.0 and 3.0 approaches.

Figure 11.13 shows how the APIs will do the heavy lifting of working with the AI model to generate the artwork. Once the user is satisfied, they will interact with aspects of the UI that execute smart contracts to generate and add the NFT to the blockchain. Transactions will happen on the blockchain. APIs may interact directly with the blockchain, looking for similar works.

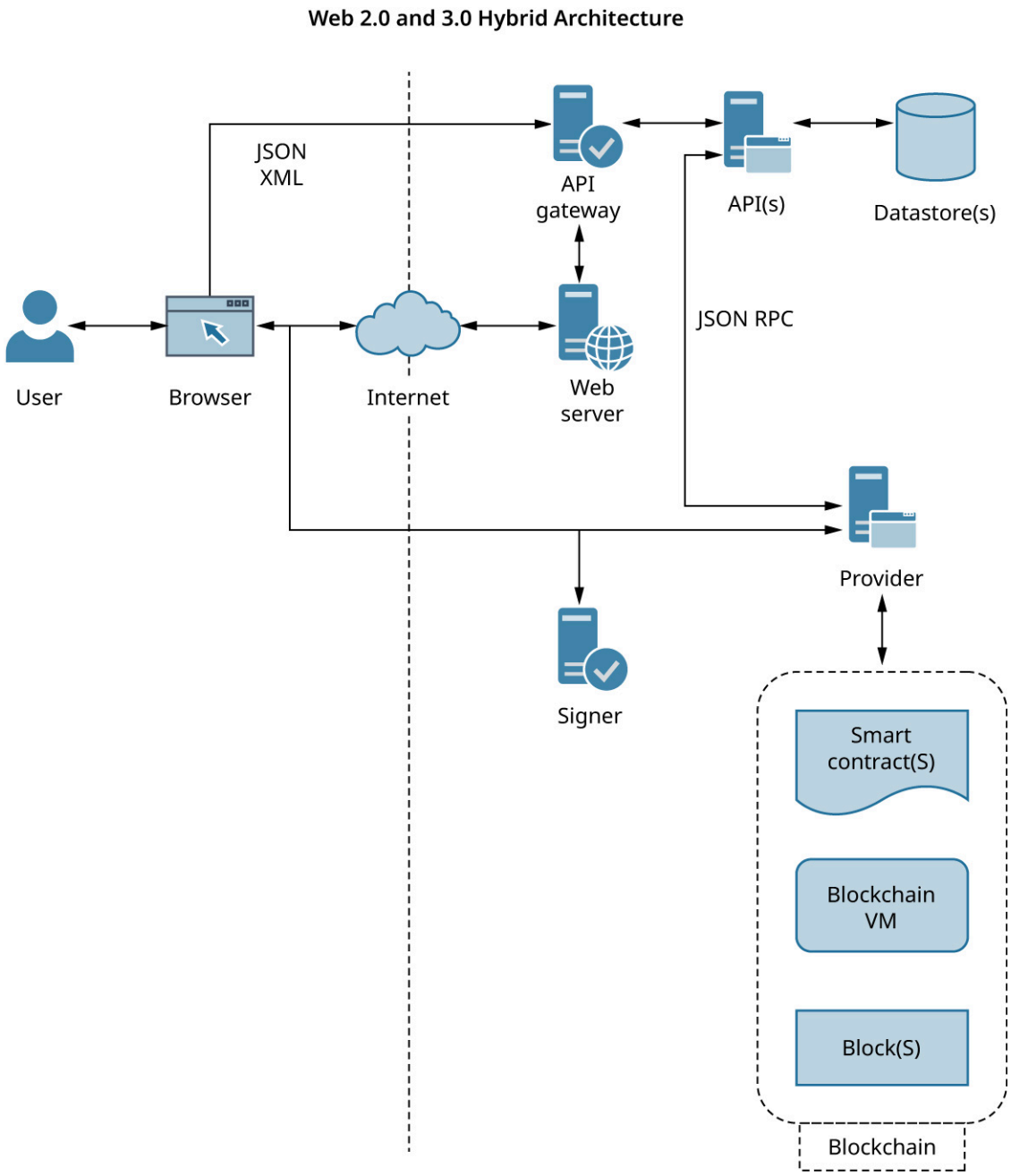


Figure 11.13 This outlines the user's interaction with a hybrid Web 2.0/3.0 application. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

This model still has a centralized Web 2.0 server for artwork generation and account management; however, portions that deal with NFT ownership and selling of that ownership are managed within the Web 3.0 blockchain infrastructure. This approach serves the needs of many businesses that want to take advantage of Web 3.0 features while preserving their original Web 2.0 websites.

LINK TO LEARNING

This timeline showing the history of web frameworks (https://openstax.org/r/76WebFrameHist) shows numerous frameworks as well as some of their updates.

11.2 Sample Responsive WAD with Bootstrap and Django

Learning Objectives

By the end of this section, you will be able to:

- Create a Todo web application with Bootstrap and Django
- Create a Django project
- Create and register a Todo app
- Define the Todo model
- Set up the Django REST APIs
- Create the user interface

In this module, you will create a simple, responsive Todo application. To accomplish this, you will use Bootstrap and Django. Bootstrap is an open-source, responsive web application framework, and Django is a Python-based web application development framework. Both frameworks are highly popular due to their ease of use.

Creating a Todo Web Application with Bootstrap and Django

The Todo web application in this subsection uses Bootstrap and Django. Bootstrap's CSS templates are used for the UI features. Django serves as both the front end and back end. Django templates are part of the user interface to get and set data via HTTP requests. Incoming requests are then handled by an API built using the Django REST Framework.

Prerequisites

To build the Todo application, you must install Python, PIP, Django, Django REST Framework, Bootstrap, and jQuery. The Todo application on the following pages was developed and tested with specific software versions. To avoid errors, please ensure you install the same versions: Python v3.9.4, PIP v21.3.1, Django v4.0.1, Django REST Framework v3.13.1, Bootstrap v4.5.0, and jQuery v3.5.1.

The steps to build the Todo application are as follows:

- Install and set up a local programming environment for Python 3.
- Download and install Python 3.9.4.
- Figure 11.14 shows adding Python and its Scripts subfolder to your path environment data (on Windows, the path updates might be: C:\Users\your_username\AppData\Local\Programs\Python\Python39 and C:\Users\your_username\AppData\Local\Programs\Python\Python39\Scripts).

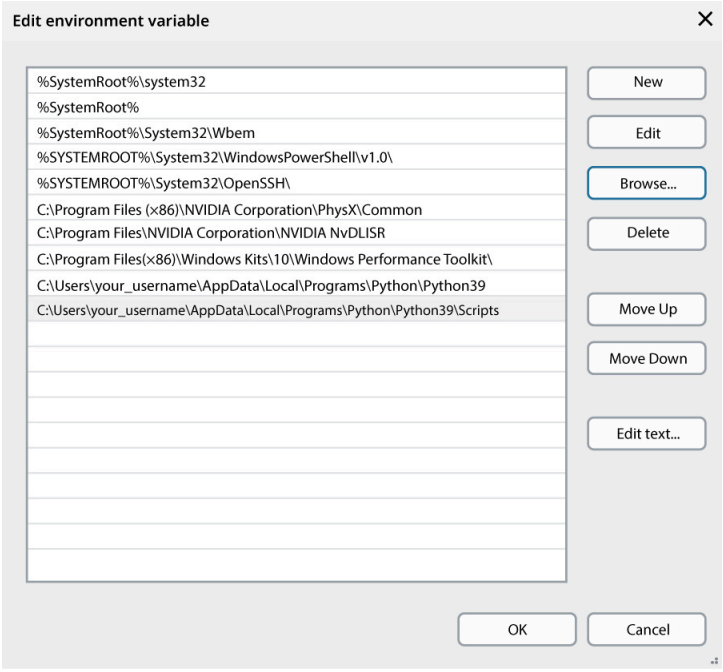


Figure 11.14 This is what appears when adding Python and its Scripts folders to the environment variables path on Windows. (Used with permission from Microsoft.)

- Create a Python venv:
  `$ python -m venv py394venv`
- Activate the venv:
  `$ cd py394venv`
  Windows: `$ .\Scripts\activate.bat`
  macOS: `$ source ./bin/activate`
- Install in the local programming environment Django:
  `$ pip install Django==4.0.1`
- Install in the local programming environment Django REST Framework:
  `$ pip install djangorestframework==3.13.1]`

Figure 11.15 shows the sequence of steps needed to install the Python environment for working with Django and the Django REST Framework.

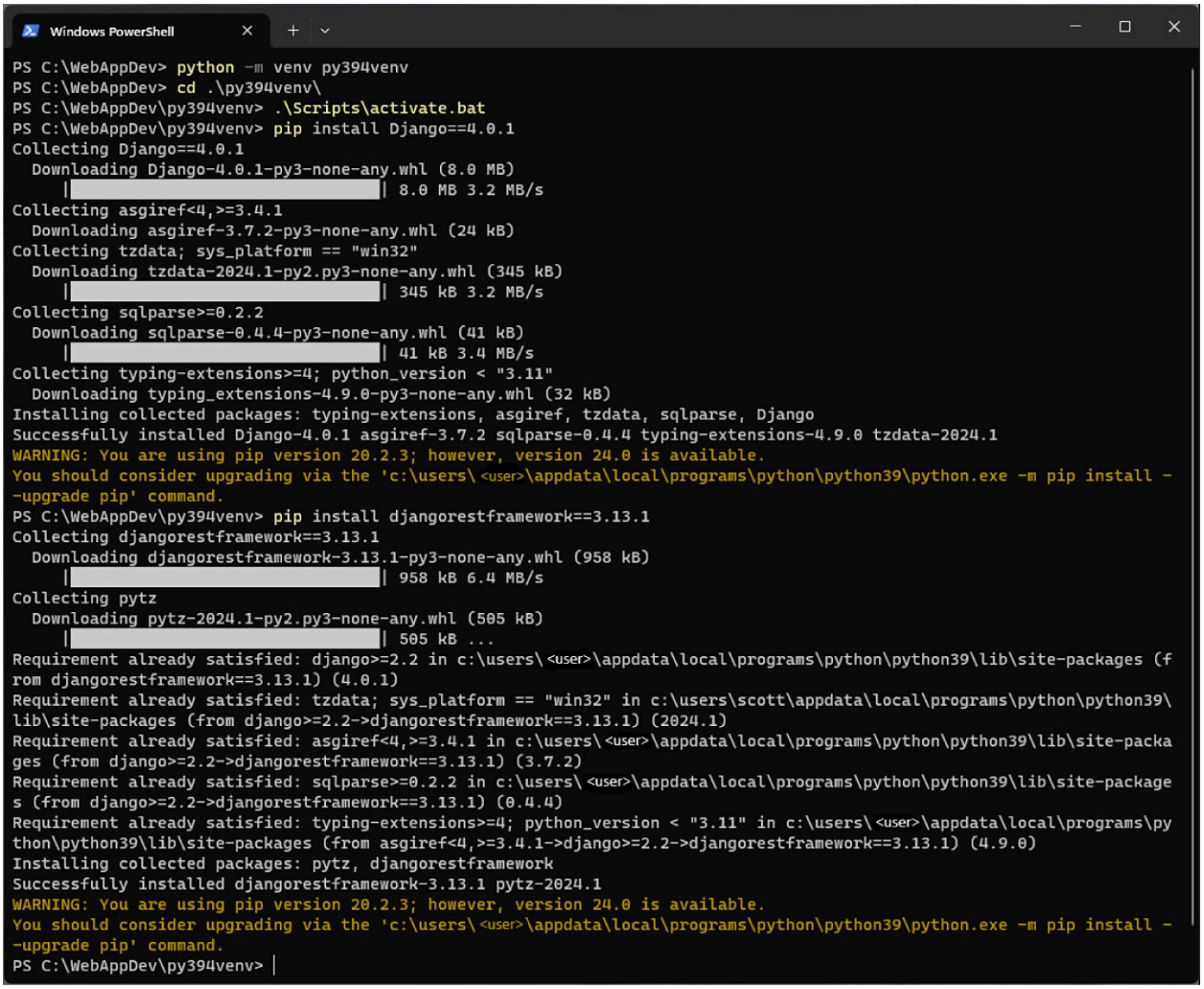


Figure 11.15 This screenshot displays the sequence of steps needed to install the Python environment. (Used with permission from Microsoft)

Creating the Django Project

The first step to building a Django web application is to create a **Django project**, which is a high-level directory used to contain the directories and files necessary to run a Django web application. To create a Django project, run the following commands:

```
$ mkdir BootstrapDjangoToDoApp
$ cd BootstrapDjangoToDoApp
$ django-admin startproject ToDoApp.
```

The period at the end of the last command is very important to ensure that Django-dependent files are generated in the current directory. By following these commands, the directory, BootstrapDjangoToDoApp/, will be created and Django-dependent files will be generated as shown in Figure 11.16.

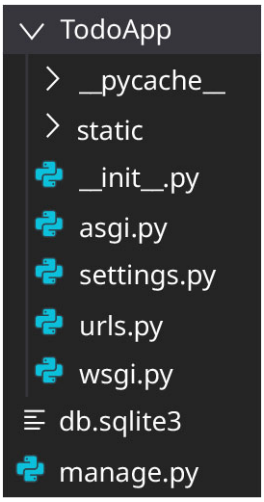


Figure 11.16 The directory BootstrapDjangoToDoApp/ includes these Django-dependent files. (rendered in Django, a registered trademark of the Django Software Foundation; attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Django built-in database tables are used to manage users, groups, migrations, and so forth in a web application. To generate these tables, run the migrate command:

```
$ python manage.py migrate
```

The final step to creating a Django project is to confirm that the setup was completed. To do this, run the runserver command shown here:

```
$ python manage.py runserver
```

To further confirm that the Django project setup was completed, launch a browser and navigate to http://localhost:8000. Figure 11.17 shows the Django page.

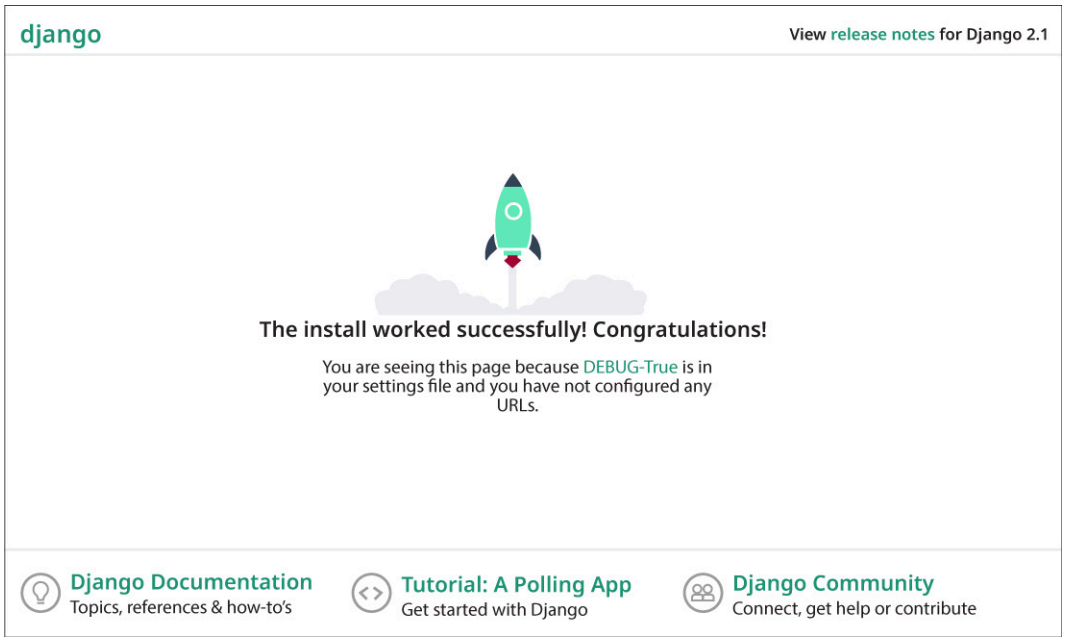


Figure 11.17 Once the Django project setup is successfully completed, this page should appear at http://localhost:8000. (credit: Django is a registered trademark of the Django Software Foundation.)

LINK TO LEARNING

Django (https://openstax.org/r/76Django) is a free and open-source Python web framework that can make web development more efficient and less time-consuming. With an emphasis on streamlining web development and making it easier for web developers to meet deadlines, Django also requires less code.

Creating and Registering the Todo App

Once the Django project is successfully completed and set up, the next step is to create the Todo application and register it in the Django project. To accomplish this, run the following command:

```
$ python manage.py startapp todo
```

The todo/ directory will be generated under the Django project directory, BootstrapDjangoToDoApp/. Files related to the Todo application will be generated as shown in Figure 11.18.

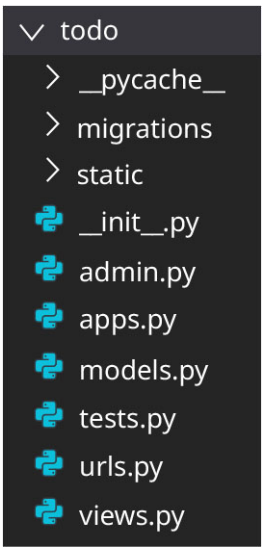


Figure 11.18 This shows the todo/ directory, which is generated under the Django project directory, BootstrapDjangoToDoApp/. (rendered in Django, a registered trademark of the Django Software Foundation; attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Next, the Todo application must be registered in the Django project as an installed app so that Django can recognize it. To do this, open the ToDoApp/settings.py file. Look for the INSTALLED_APPS variable as seen in Figure 11.19. Add 'todo' to the list as shown. Because the Django REST Framework will be used, also add 'rest_framework' to the list.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'todo',
]
```

Figure 11.19 To register the Todo application in the Django project as an installed app, the list of installed apps should include 'rest_framework' and 'todo'. (rendered in Django, a registered trademark of the Django Software Foundation; attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Define the Todo Model

Once you register and install the ToDo web application, you will be able to create todo tasks, which can be assigned categories. The next step is to create the models. Two models will be created for Category and TodoList. Open the todo/models.py file and add the code shown.

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals

from django.db import models
from django.utils import timezone

# Create your models here.
class Category(models.Model):

  name = models.CharField(max_length=100)

  class Meta:
    verbose_name = ("Category")
    verbose_name_plural = ("Categories")

  def __str__(self):
    return self.name

class TodoList(models.Model):
  title = models.CharField(max_length=250)
  content = models.TextField(blank=True)
  created = models.DateField(default=timezone.now().strftime("%Y-%m-%d"))
  due_date = models.DateField(default=timezone.now().strftime("%Y-%m-%d"))
  category = models.ForeignKey(Category, default="General",
on_delete=models.DO_NOTHING)
  class Meta:
    ordering = ["-created"] # order by most recently created
  def __str__(self):
    return self.title
```

The Category and TodoList Python classes describe the properties for the models for Category and TodoList tables, respectively. The TodoList model contains a ForeignKey field to the Category model. Each todo item will be associated with one category. After creating the models, a migration file needs to be generated to create the physical tables in the database. To generate the migration file, run the following command:

```
$ python manage.py makemigrations todo
```

This command will generate a migration file in todo/migrations/, which will look like the following code.

```
# Generated by Django 4.0.1 on 2022-01-30 21:14

from django.db import migrations, models
import django.db.models.deletion


class Migration(migrations.Migration):

  initial = True

  dependencies = [
  ]

  operations = [
     migrations.CreateModel(
        name='Category',
        fields=[
           ('id', models.BigAutoField(auto_created=True, primary_key=True,
serialize=False, verbose_name='ID')),
           ('name', models.CharField(max_length=100)),
        ],
        options={
           'verbose_name': 'Category',
           'verbose_name_plural': 'Categories',
        },
     ),
     migrations.CreateModel(
        name='TodoList',
        fields=[
           ('id', models.BigAutoField(auto_created=True, primary_key=True,
serialize=False, verbose_name='ID')),
           ('title', models.CharField(max_length=250)),
           ('content', models.TextField(blank=True)),
           ('created', models.DateField(default='2022-01-30')),
           ('due_date', models.DateField(default='2022-01-30')),
           ('category', models.ForeignKey(default='General',
on_delete=django.db.models.deletion.DO_NOTHING, to='todo.category')),
        ],
        options={
           'ordering': ['-created'],
```

```
        },
    ),
]
```

The next step is to apply the changes in the migration file to the database, which is accomplished by running the following command:

```
$ python manage.py migrate
```

For this application, Django uses the default sqlite3 (db.sqlite3) database. Please note that Django supports other databases as well, including MySQL and PostgreSQL. To define the Todo model, you can use the default Django admin interface to perform CRUD operations on the database. To use the admin interface, open the todo/admin.py file and register the models as seen in the following code snippet.

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals

from django.contrib import admin
from . import models

# Register your models here.
class TodoListAdmin(admin.ModelAdmin):
  list_display = ("title", "created", "due_date")

class CategoryAdmin(admin.ModelAdmin):
  list_display = ("name",)

admin.site.register(models.TodoList, TodoListAdmin)
admin.site.register(models.Category, CategoryAdmin)
```

The next step is to create a superuser account that allows access to the admin interface. To do this, run the following command and follow the prompts to enter a username, email address, and password for the superuser.

```
$ python manage.py createsuperuser
```

Once this step is complete, restart the server using the following command:

```
$ python manage.py runserver
```

After this is complete, open a browser and navigate to http://localhost:8000/admin/. To access the admin interface, log in with the credentials that you set up for the superuser. When you log in to the admin interface, you should see the following page from Figure 11.20. On this page, you will have the ability to create, edit, and delete categories and todo items.

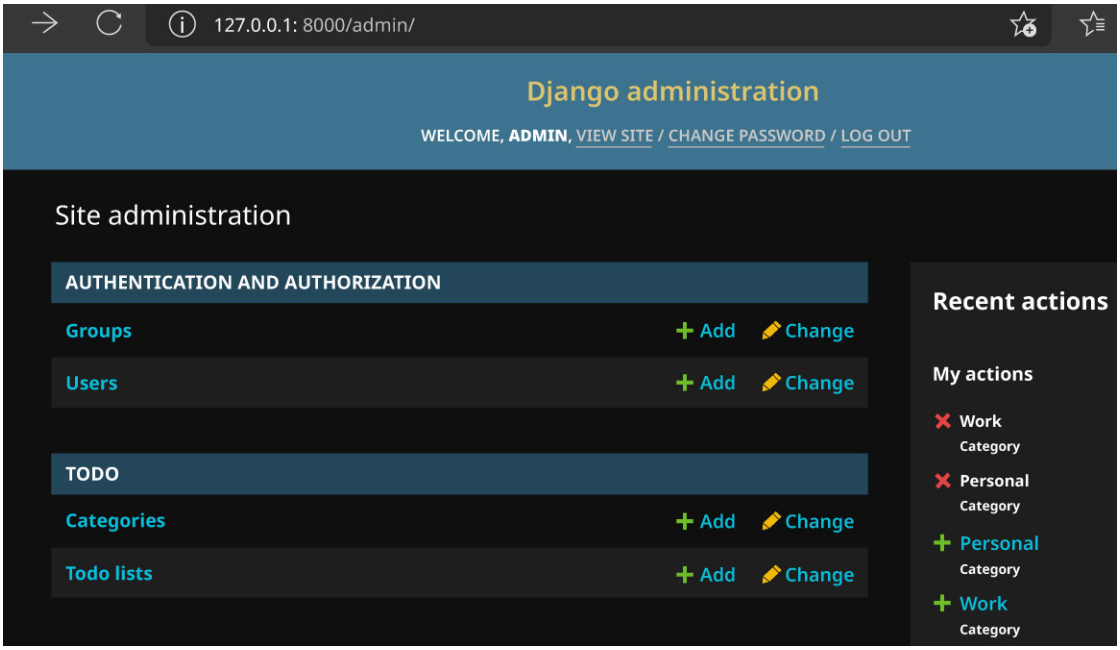


Figure 11.20 The admin interface is where superusers can create, edit, and delete categories and todo items in the Todo web application. (rendered in Django, a registered trademark of the Django Software Foundation; attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Setting Up the Django REST APIs

Previously, you installed the Django REST Framework, which provides a toolkit to build APIs. In this section, you'll create the serializers, View functions, and routers required for the API.

Creating the Serializers

The next step is to implement two serializers, one for the Category model and one for the TodoList model. A **serializer** is a tool to control response outputs and convert complex data into content, such as JSON. In the todo/ directory, create the file, serializers.py, and add the code shown in the code snippet that follows. The serializer classes inherit from serializers.ModelSerializer because this class creates the serializer, with fields corresponding to the Model fields defined earlier. Each serializer specifies the Model to work with and the fields to be included in the JSON object, which are all fields.

```
# todo/serializers.py

from rest_framework import serializers
from .models import TodoList, Category

class CategorySerializer(serializers.ModelSerializer):
  class Meta:
    model = Category
    fields = "__all__"

class TodoSerializer(serializers.ModelSerializer):
  class Meta:
    model = TodoList
    fields = "__all__"
```

Creating the View

After you create each serializer, the next step is to create a corresponding View function for both the Category

serializer and the Todo serializer. To do this, open todo/views.py and add the code shown in the code snippet that follows. The viewsets class, which provides a default implementation of the CRUD operations, is imported from rest_framework. The CategoryView and TodoView classes provide a queryset of categories and todo items, respectively. They also specify the serializer_class defined in the previous section.

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals

from django.shortcuts import render,redirect
from rest_framework import viewsets
from .serializers import TodoSerializer, CategorySerializer
from .models import TodoList, Category
import datetime

# Create your views here.

class CategoryView(viewsets.ModelViewSet):
  queryset = Category.objects.all()
  serializer_class = CategorySerializer

class TodoView(viewsets.ModelViewSet):
  queryset = TodoList.objects.all()
  serializer_class = TodoSerializer
```

Creating the Routers

The next step is to create the routers that provide URL paths for the API. To do this, open the todo/urls.py file and add the code shown in the code snippet that follows:

```
from django.urls import path, include
#from todo import views as todo_views
from rest_framework import routers
from todo.views import *

router = routers.DefaultRouter()
router.register(r'categories', CategoryView, basename='Categories')
router.register(r'todos', TodoView, basename='Todos')

urlpatterns = [
  path('', index, name="TodoList"),
  path(r'api/', include(router.urls)),
]
```

Once you complete this step, launch the Django server using the following command:

```
$ python manage.py runserver
```

To access the API, launch a browser and navigate to http://127.0.0.1:8000/api/. As shown in Figure 11.21, you should see two API paths listed, one for categories and another for todo items.

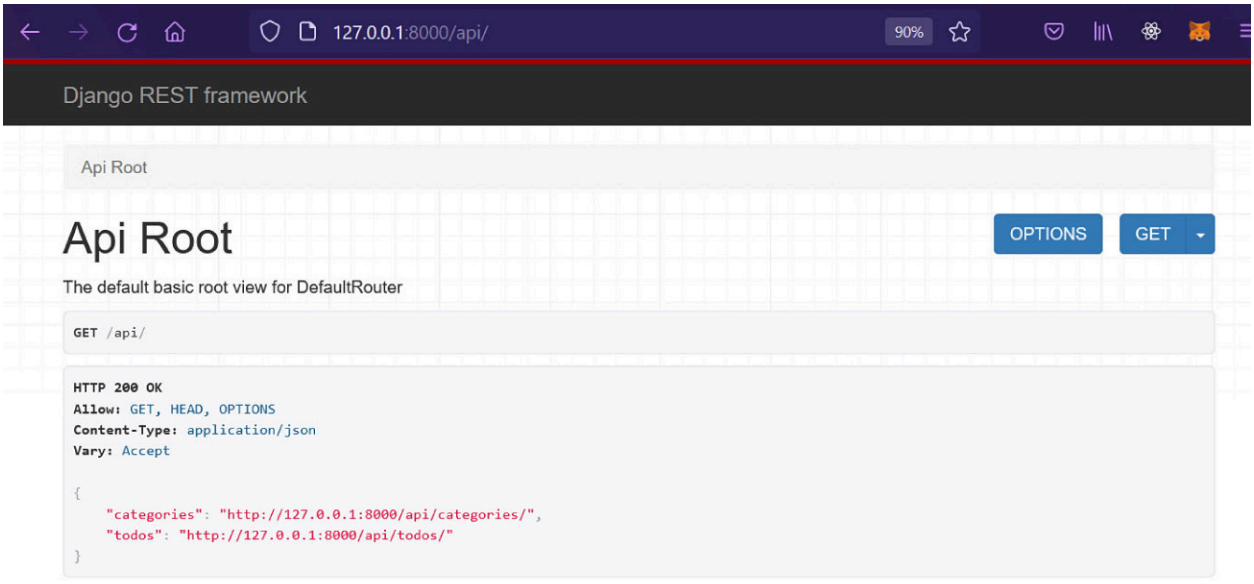


Figure 11.21 Once the URL paths for the API are created, two API paths are listed—one path for categories and the other for todo. (rendered in Django, a registered trademark of the Django Software Foundation; attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Todo items are dependent on categories. To perform CRUD operations on the Category table, click on the categories API path, as shown in Figure 11.22.

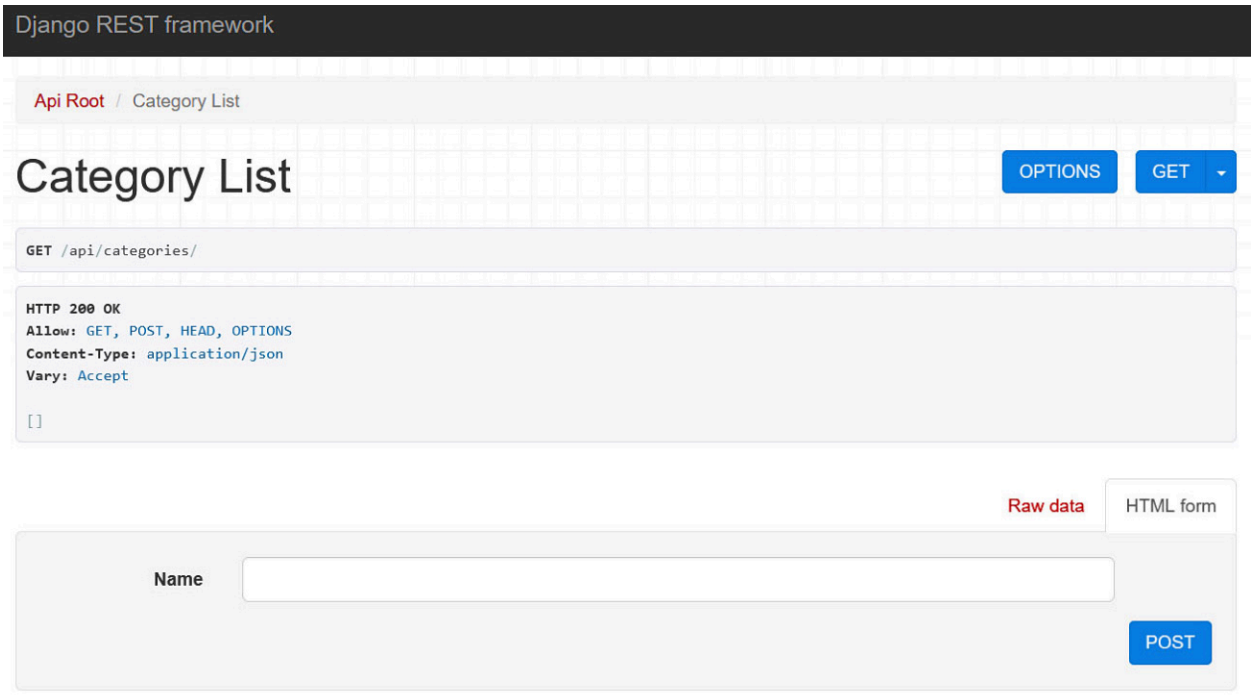


Figure 11.22 The API path can be used to perform CRUD operations on the Category table. (rendered in Django, a registered trademark of the Django Software Foundation; attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Once you access the Category List page, enter a category name in the field near the bottom of the page and click the Post button to save it. You can add additional category names. Once you save each name using the Post button, the categories will appear in JSON format, as illustrated in Figure 11.23.

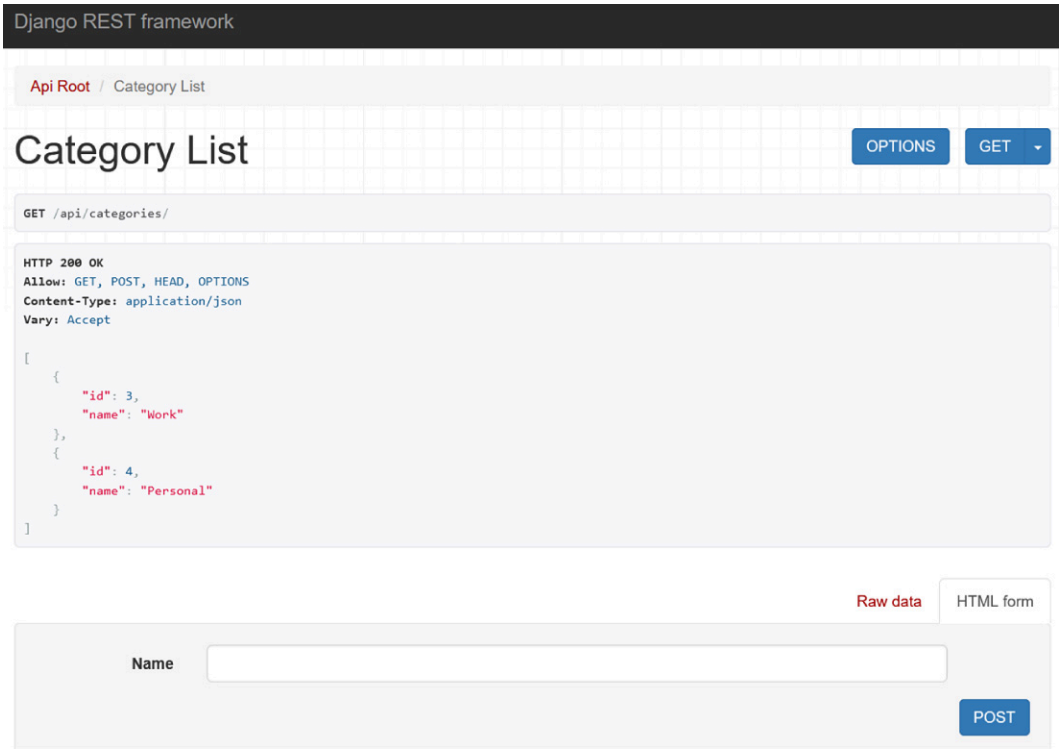


Figure 11.23 The Category List is created in JSON format after saving category names. (rendered in Django, a registered trademark of the Django Software Foundation; attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

To ensure that categories can be updated or deleted as needed, the primary key, which is "id", must be included in the API path (e.g., /api/categories/{id}/). For example, to update or delete the category with id=3, the API path is /api/categories/3/. As shown in Figure 11.24, once this category is pulled up on the Category Instance page, the DELETE button is visible at the top to delete the category. If the category needs to be updated, the PUT button visible near the bottom can be used for updates.

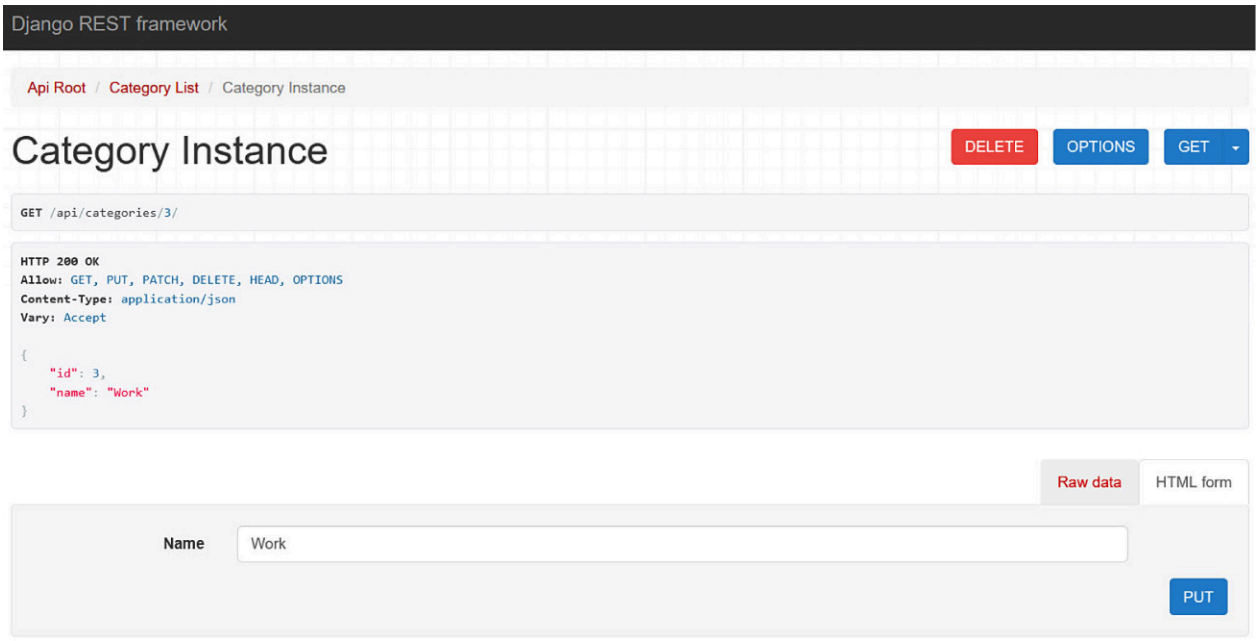


Figure 11.24 The Category Instance page provides DELETE and PUT buttons, which, respectively, can be used to delete or update a category. (rendered in Django, a registered trademark of the Django Software Foundation; attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Next, navigate back to the API root. Click on the todos API path to see, as outlined in Figure 11.25, that a Todo

item has a Category field that appears as a drop-down list of categories. The CRUD operations also can be performed on the TodoList table. This will be done next through the user interface via templates.

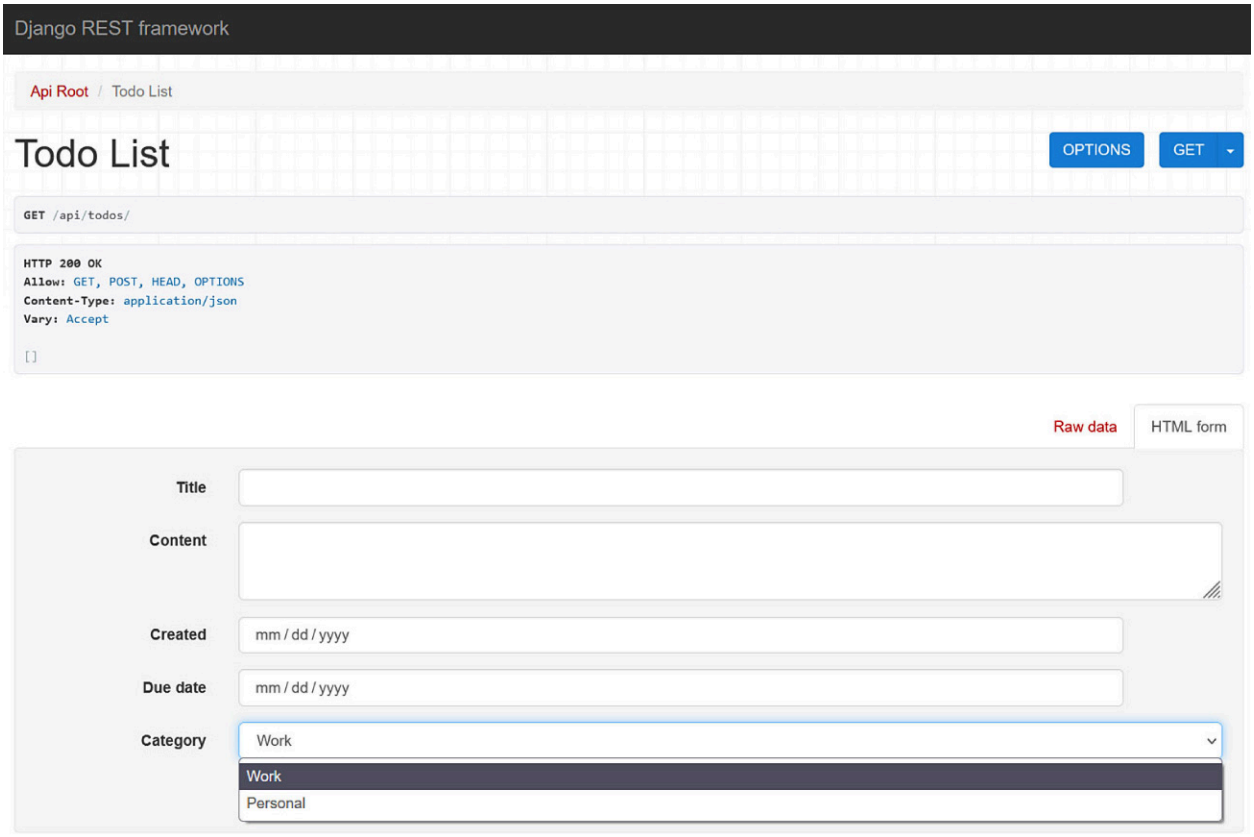


Figure 11.25 After the routers are created, a Todo item has a Category field that appears as a drop-down list of categories, such as "Work" and "Personal." (rendered in Django, a registered trademark of the Django Software Foundation; attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Creating the User Interface

To use and control the Todo application, you must create a user interface (UI). This section will outline the steps to do this.

Installing Bootstrap

The first step to create a UI is to install Bootstrap, which, as an open-source, responsive web application framework, can be used in web applications like Django to create UIs. To install Bootstrap in a Django application, you have several options. In this scenario, an efficient method is to download the Bootstrap CSS and JS files and add them to the static/ directory, as shown in Figure 11.26. To do this, first create the static/ directory in the Django project directory. In addition, to install jQuery, download the JS file and add it to the static/ directory. Finally, add a custom CSS file to include individual style in the Django web application.

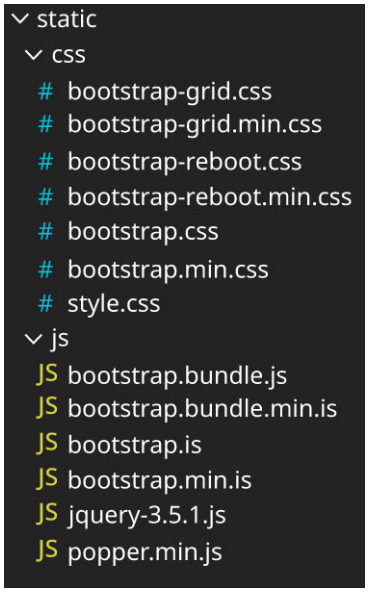


Figure 11.26 An efficient method to install Bootstrap in a Django application is to download the Bootstrap CSS and JS files and add them to the static/ directory. (rendered in Django, a registered trademark of the Django Software Foundation; attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

After adding the CSS and JS files to the static/ directory, the next step is to open the ToDoApp/settings.py file, navigate to the bottom of the file, and add the path variables shown in the following code snippet.

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.0/howto/static-files/

STATIC_URL = 'static/'

PROJECT_ROOT = os.path.dirname(os.path.abspath(__file__))
STATIC_ROOT = os.path.join(PROJECT_ROOT, 'static')
```

Creating the View

The next step is to create the View. The View function is needed for template pages, which are created in the next section. The View function is also needed to interact with the database to both create and delete todo items.

To create the View, open the todo/views.py file and add the code shown in the following code snippet. This code takes an HTTP request object. If the request method is POST, a todo item is either created or deleted, depending on which button is clicked. Otherwise, the request method is GET and the todo items are displayed to the user.

```
def index(request): # index view
  todos = TodoList.objects.all() # query all todos with object manager
  categories = Category.objects.all() # get all categories with object manager
  if request.method == "POST": # check if request method is POST
    if "taskAdd" in request.POST: # check if request is to add a todo item
      title = request.POST["description"] # title
      date = str(request.POST["date"]) # date
      category = request.POST["category_select"] # category
      content = title + " -- " + date + " " + category # content
      Todo = TodoList(title=title, content=content, due_date=date,
```

```
category=Category.objects.get(name=category))
      Todo.save() # save todo item
      return redirect("/") # reload page
    if "taskDelete" in request.POST: # check if request is to delete a todo
      checkedlist = request.POST["checkedbox"] # checked todos to be deleted
      for todo_id in checkedlist:
        todo = TodoList.objects.get(id=int(todo_id)) # get todo id
        todo.delete() # delete todo
  return render(request, "index.html", {"todos": todos, "categories":categories})
```

Creating the Templates

The final step to build a Django web application is to create templates, which are HTML files that can also contain embedded Python code. Create a file called style.css under the /static/css directory and insert the code shown here into that file. The templates discussed in the following sections require this style sheet.

```
/* basic reset */
*,
*:before,
*:after {
  -webkit-box-sizing: border-box;
  -moz-box-sizing: border-box;
  box-sizing: border-box;
}

/* app */
html {
  font-size: 100%;
}

body {
  background: #e6f9ff;
  font-family: "Open Sans", sans-serif;
}
/* super basic grid structure */
.container {
  width: 600px;
  margin: 0 auto;
  background: #ffffff;
  padding: 20px 0;
  -webkit-box-shadow: 0 0 2px rgba(0, 0, 0, 0.2);
  box-shadow: 0 0 2px rgba(0, 0, 0, 0.2);
}

.row {
  display: block;
  padding: 10px;
  text-align: center;
  width: 100%;
  clear: both;
  overflow: hidden;
```

```
}

.half {
  width: 50%;
  float: left;
}

.content {
  background: #fff;
}

/* logo */
h1 {
  font-family: "Rokkitt", sans-serif;
  color: #666;
  text-align: center;
  font-weight: 400;
  margin: 0;
}

.tagline {
  margin-top: -10px;
  text-align: center;
  padding: 5px 20px;
  font-size: 11px;
  font-weight: 600;
  text-transform: uppercase;
  color: #777;
}

/* inputs */
.inputContainer {
  height: 60px;
  border-top: 1px solid #e5e5e5;
  position: relative;
  overflow: hidden;
}

.inputContainer.last {
  border-bottom: 1px solid #e5e5e5;
  margin-bottom: 20px;
}

.inputContainer.half.last.right {
  border-left: 1px solid #efefef;
}

input[type="date"],
input[type="text"],
select {
```

```
  height: 100%;
  width: 100%;
  padding: 0 20px;
  position: absolute;
  top: 0;
  vertical-align: middle;
  display: inline-block;
  border: none;
  border-radius: none;
  font-size: 13px;
  color: #777;
  margin: 0;
  font-family: "Open Sans", sans-serif;
  font-weight: 600;
  letter-spacing: 0.5px;
  -webkit-transition: background 0.3s;
  transition: background 0.3s;
}

input[type="date"] {
  cursor: pointer;
}

input[type="date"]:focus,
input[type="text"]:focus,
select:focus {
  outline: none;
  background: #ecf0f1;
}

::-webkit-input-placeholder {
  color: lightgrey;
  font-weight: normal;
  -webkit-transition: all 0.3s;
  transition: all 0.3s;
}
::-moz-placeholder {
  color: lightgrey;
  font-weight: normal;
  transition: all 0.3s;
}
::-ms-input-placeholder {
  color: lightgrey;
  font-weight: normal;
  transition: all 0.3s;
}
input:-moz-placeholder {
  color: lightgrey;
  font-weight: normal;
  transition: all 0.3s;
```

```
}

input:focus::-webkit-input-placeholder {
  color: #95a5a6;
  font-weight: bold;
}

input:focus::-moz-input-placeholder {
  color: #95a5a6;
  font-weight: bold;
}

.inputContainer label {
  padding: 5px 20px;
  font-size: 11px;
  font-weight: 600;
  text-transform: uppercase;
  color: #777;
  display: block;
  position: absolute;
}

button {
  font-family: "Open Sans", sans-serif;
  background: transparent;
  border-radius: 2px;
  border: none;
  outline: none;
  height: 50px;
  font-size: 14px;
  color: #fff;
  cursor: pointer;
  text-transform: uppercase;
  position: relative;
  -webkit-transition: all 0.3s;
  transition: all 0.3s;
  padding-left: 30px;
  padding-right: 15px;
}

.icon {
  position: absolute;
  top: 30%;
  left: 10px;
  font-size: 20px;
}

.taskAdd {
  background: #444;
  padding-left: 31px;
```

```
}

.taskAdd:hover {
  background: #303030;
}

.taskDelete {
  background: #e74c3c;
  padding-left: 30px;
}

.taskDelete:hover {
  background: #c0392b;
}

/* task styles */
.taskList {
  list-style: none;
  padding: 0 20px;
}

.taskItem {
  border-top: 1px solid #e5e5e5;
  padding: 15px 0;
  color: #777;
  font-weight: 600;
  font-size: 14px;
  letter-spacing: 0.5px;
}

.taskList .taskItem:nth-child(even) {
  background: #fcfcfc;
}

.taskCheckbox {
  margin-right: 1em;
}

.complete-true {
  text-decoration: line-through;
  color: #bebebe;
}

.taskList .taskDate {
  color: #95a5a6;
  font-size: 10px;
  font-weight: bold;
  text-transform: uppercase;
  display: block;
  margin-left: 41px;
```

```
}

.fa-calendar {
  margin-right: 10px;
  font-size: 16px;
}

[class*="category-"] {
  display: inline-block;
  font-size: 10px;
  background: #444;
  vertical-align: middle;
  color: #fff;
  padding: 10px;
  width: 75px;
  text-align: center;
  border-radius: 2px;
  float: right;
  font-weight: normal;
  text-transform: uppercase;
  margin-right: 20px;
}

.category- {
  background: transparent;
}

.category-Personal {
  background: #2980b9;
}

.category-Work {
  background: #8e44ad;
}

.category-School {
  background: #f39c12;
}

.category-Cleaning {
  background: #16a085;
}

.category-Other {
  background: #d35400;
}

footer {
  text-align: center;
  font-size: 11px;
```

```
  font-weight: 600;
  text-transform: uppercase;
  color: #777;
}

footer a {
  color: #f39c12;
}
/* custom checkboxes */
.taskCheckbox {
  -webkit-appearance: none;
  appearance: none;
  -webkit-transition: all 0.3s;
  transition: all 0.3s;
  display: inline-block;
  cursor: pointer;
  width: 19px;
  height: 19px;
  vertical-align: middle;
}

.taskCheckbox:focus {
  outline: none;
}

.taskCheckbox:before,
.taskCheckbox:checked:before {
  font-family: "FontAwesome";
  color: #444;
  font-size: 20px;
  -webkit-transition: all 0.3s;
  transition: all 0.3s;
}

.taskCheckbox:before {
  content: "\f096";
}

.taskCheckbox:checked:before {
  content: "\f14a";
  color: #16a085;
}
/* custom select menu */
.taskCategory {
  -webkit-appearance: none;
  appearance: none;
  cursor: pointer;
  padding-left: 16.5px; /*specific positioning due to difficult behavior of select
element*/
  background: #fff;
```

```
}

.selectArrow {
  position: absolute;
  z-index: 10;
  top: 35%;
  right: 0;
  margin-right: 20px;
  color: #777;
  pointer-events: none;
}

.taskCategory option {
  background: #fff;
  border: none;
  outline: none;
  padding: 0 100px;
}
```

The first template file is base.html, which is the file that includes links to Bootstrap and jQuery. To illustrate these links, a snippet of the base.html file is shown in the code. The Bootstrap navigation bar is implemented in this file.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>TodoApp - Django</title>
    {% load static %}
    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="/static/css/bootstrap.min.css" />
    <link
      rel="stylesheet"
      type="text/css"
      href="https://maxcdn.bootstrapcdn.com/font-awesome/4.7.0/css/font-
awesome.min.css"
    />
    <link
      rel="stylesheet"
      type="text/css"
      href="{% static 'css/style.css' %}"
    />
    <!-- jQuery -->
    <script
      type="text/javascript"
      src="{% static 'js/jquery-3.5.1.min.js' %}"></script>
    <!-- Popper -->
    <script
      type="text/javascript"
      src="{% static 'js/popper.min.js' %}"></script>
```

```
    <!-- Bootstrap Core JavaScript -->
    <script
    type="text/javascript"
    src="{% static 'js/bootstrap.bundle.min.js' %}"></script>
<script
</head>

<body>
  <nav class="navbar navbar-dark bg-dark justify-content-between">
    <a class="navbar-brand">ToDo</a>
    <form class="form-inline">
      <input
        class="form-control mr-sm-2"
        type="search"
        placeholder="Search"
        aria-label="Search"
      />
      <button class="btn btn-outline-info my-2 my-sm-0" type="submit">
        Search
        </button>
      </form>
    </nav>
    <div>{% block content %} {% endblock content %}</div>
  </body>
</html>
```

The second template file is index.html, which lists any existing todo items and includes the form to create or delete a todo item. A snippet of the index.html file is shown in the following code. The index.html file also extends the base.html template, which allows the Bootstrap navigation bar implemented in base.html to be inserted in this page and every page that extends it.

```
{% extends "./base.html" %} {% load static %} {% block content %}
 <div django-app="TaskManager">
  <div class="container mt-5">
   <div class="content">
    <h1>Todo List</h1>
    <!--<p class="tagline">Django Todo App</p>-->
    <form action="" method="post">
     {% csrf_token %}
     <!-- csrf token for basic security -->
     <div class="inputContainer">
       <input         type="text"         id="description"         class="taskName"
        placeholder="What do you need to do?"         name="description"
        required        />
       <label for="description">Description</label>
      </div>
      <div class="inputContainer half last">
       <i class="fa fa-caret-down selectArrow"></emphasis>
       <select id="category" class="taskCategory" name="category_select">
        <option class="disabled" value="">Choose a category</option>
```

```
       {% for category in categories %}
       <option         class=""         value="{{ category.name }}"         name="{{
category.name }}"        >
        {{ category.name }}
       </option>
       {% endfor %}
      </select>
      <label for="category">Category</label>
     </div>
     <div class="inputContainer half last right">
      <input type="date" id="dueDate" class="taskDate" name="date" />
      <label for="dueDate">Due Date</label>
     </div>
     <div class="row">
      <button class="taskAdd" name="taskAdd" type="submit">
       <i class="fa fa-plus icon"></emphasis>Add task
      </button>
      <button        class="taskDelete"        name="taskDelete"
       formnovalidate=""        type="submit"
       onclick="$('input#sublist').click();"       >
       <i class="fa fa-trash-o icon"></emphasis>Delete Tasks
      </button>
     </div>
     <ul class="taskList">
      {% for todo in todos %}
      <!-- django template lang - for loop -->
      <li class="taskItem">
        <input          type="checkbox"          class="taskCheckbox"
         name="checkedbox"          id="{{ todo.id }}"          value="{{ todo.id }}"
        />
        <label for="{{ todo.id }}" ><span class="complete-">{{ todo.title
}}</span></label>
        <span class="category-{{ todo.category }}"          >{{ todo.category
}}</span>
         >{{ todo.category }}</span
<strong class="taskDate" ><i class="fa fa-calendar"></emphasis>Created:
{{todo.created}} - Due:
         >{{ todo.category }}</span
{{todo.due_date}}</strong>
        >{{ todo.category }}</span
       </li>
       {% endfor %}
      </ul>
      <!-- taskList -->
     </form>
    </div>
    <!-- content -->
   </div>
  <!-- container -->
</div>
```

```
{% endblock %}
```

The first step to access the Todo Django web application is to restart the Django server using the following command:

```
$ python manage.py runserver
```

Next, launch a browser and navigate to http://localhost:8000. The page highlighted in Figure 11.27 should appear.

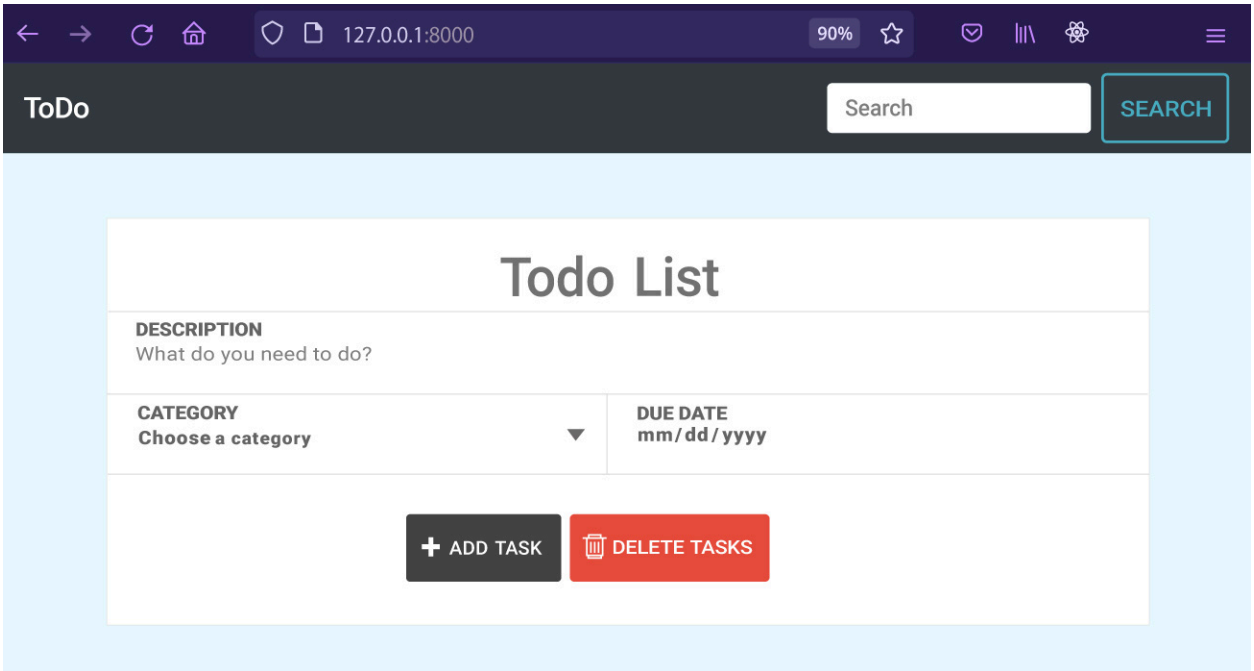


Figure 11.27 Once the Django server is restarted, this page should appear at http://localhost:8000. (rendered in Django, a registered trademark of the Django Software Foundation; attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

To create a todo list, fill out the form and click the Add Task button as shown in Figure 11.28.

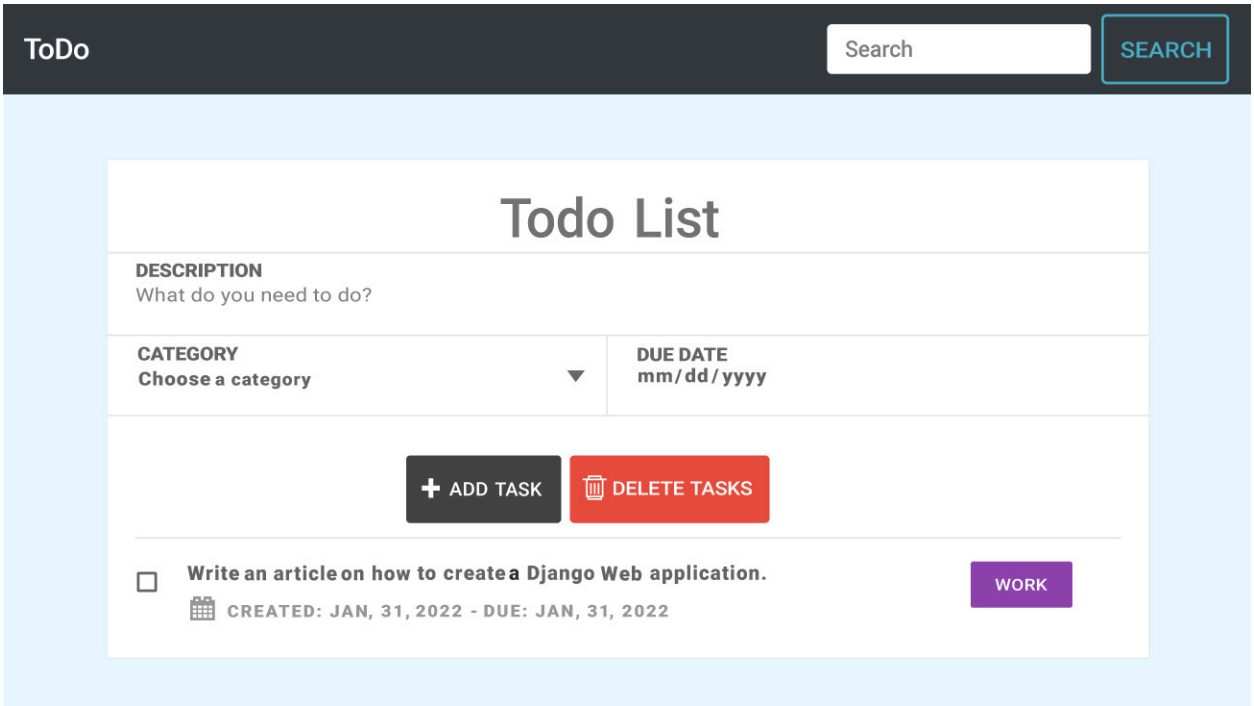


Figure 11.28 This figure shows how the Todo List should appear after using the Add Task button to create a todo list. (rendered in Django, a registered trademark of the Django Software Foundation; attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

In addition, the todo item should also be viewable in the API that was created. This should appear as outlined in Figure 11.29.

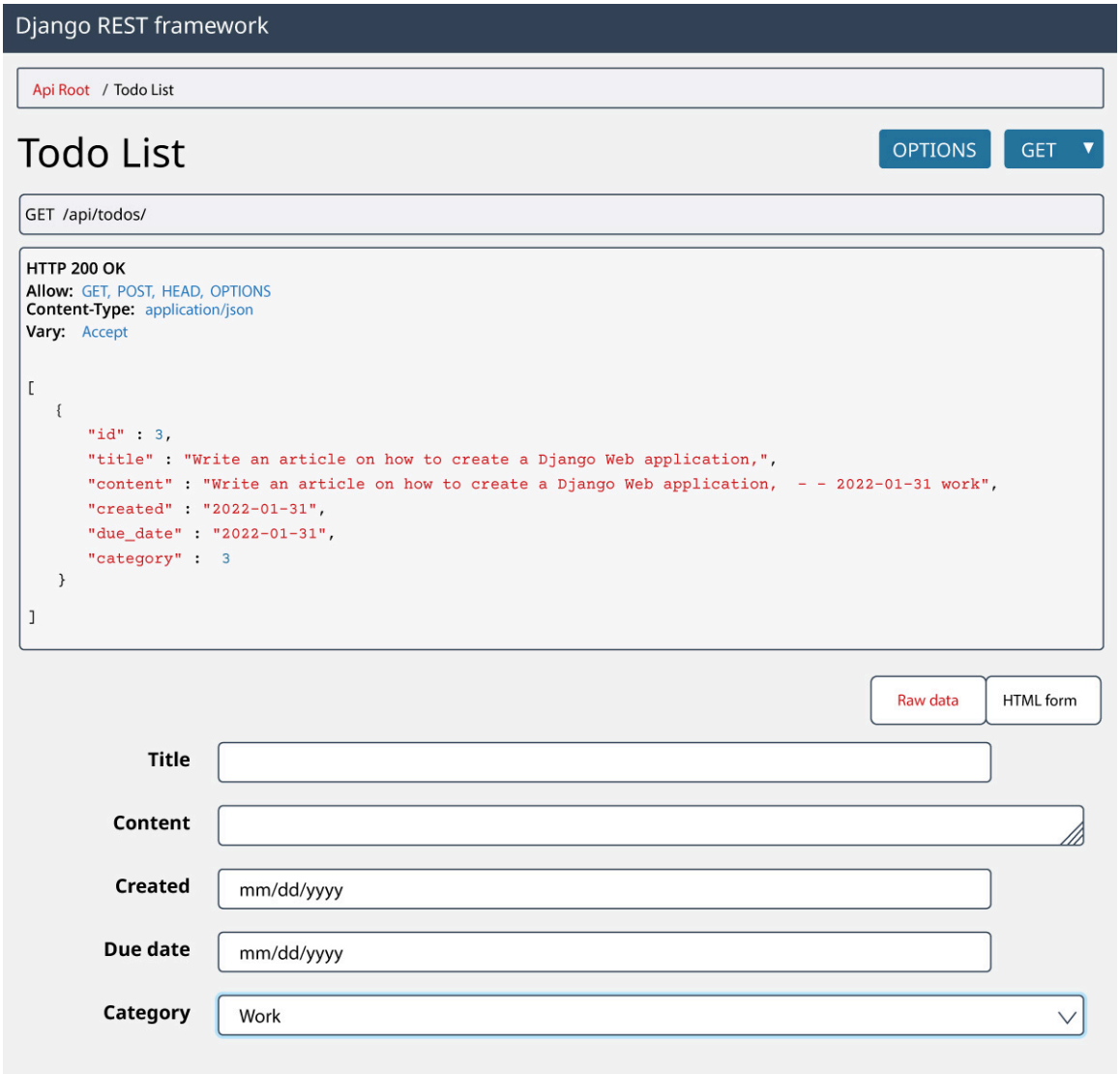


Figure 11.29 Once the todo item is created, it should be viewable in the API. (rendered in Django, a registered trademark of the Django Software Foundation; attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

11.3 Sample Responsive WAD with Bootstrap/React and Node

Learning Objectives

By the end of this section, you will be able to:

- Create a Todo web application with Bootstrap, React, and Node
- Create a Node back end
- Build the controller
- Set up the REST API
- Create the React front end
- Connect the React front end to the Node back end

In the previous section, you created a simple Todo application using Bootstrap and Django. In this section, you will continue to use Bootstrap to create another simple Todo application. But instead of working with Django, you will use React and Node. **React**, or React.js, is a JavaScript library popular to build user interfaces. **Node**, or Node.js, is a JavaScript runtime environment that provides users with the tools to develop web applications, as well as servers, scripts, and command-line tools.

Creating a Todo Web Application with Bootstrap and React and Node

When creating a Todo web application using React and Node, React serves as the front end, handling the user interface, as well as getting and setting data via HTTP requests using Axios. Node serves as the back end, using a REST API built with ExpressJS and the MongooseJS Object Data Modeling (ODM) to interact with a MongoDB database.

Prerequisites

To build the Todo application using Bootstrap, React, and Node, you will need the following software components: React v17.0.2, Bootstrap v4.5.0, Node v14.17.5, ExpressJS v4.17.2, MongooseJS v6.1.9, and Axios v0.21.0. To begin, download and install Node.

Creating the Node Back End

Several steps are needed to build the Node application back end required for the Todo application. This section will explain each of these steps.

LINK TO LEARNING

Node (https://openstax.org/r/76NodeJS) is a JavaScript runtime environment that provides users with the tools to develop web applications, as well as servers, scripts, and command-line tools. Node, which is free, is open-source and cross-platform. It was designed to develop network applications that are scalable, managing many connections simultaneously. Unlike the typical inefficient concurrency model, with Node, a callback is fired with each connection, and Node sleeps unless work needs to be done.

Creating the Node App

Before you can create the Todo application back end, you should create a Node app. To accomplish this, create the directory nodebackend/ and navigate into it. Next, run the following command to initialize the Node application:

```
$ npm init
```

After running this command, follow the prompt, which is highlighted in Figure 11.30.

```
$  npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See 'npm help init' for definitive documentation on these fields
and exactly what they do.

Use  'npm install  <pKg>'   afterwards to install a package and
save it as a dependency in the package.json file.

Press  ^C at any time to quit.
package name: (nodebackend)
version: (1.0.0)
description: Todo web application with Bootstrap, ReactJS and NodeJS
entry point: (index.js) server.js
test command:
git repository:
keywords: bootstrap, reactjs, nodejs, express, mongodb, rest, api
author:
license:  (ISC)
```

Figure 11.30 This prompt appears when the Node application is initialized. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

After the Node application initialization is completed, a package.json file is generated, as shown in the following code.

```
{
  "name": "nodebackend",
  "version": "1.0.0",
  "description": "Todo Web application with Bootstrap, ReactJS and NodeJS",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "bootstrap",
    "reactjs",
    "nodejs",
    "express",
    "mongodb",
    "rest",
    "api"
  ],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.19.1",
    "cors": "^2.8.5",
    "express": "^4.17.2",
    "mongoose": "^6.1.9"
  }
}
```

Setting Up the Express Web Server

Once the Node application is initialized, the next step is to set up the Express web server. To do this, use the following command to install Express, Mongoose (Mongoose is a library for MongoDB that is used to interact with MongoDB by facilitating the modeling of data as objects), and other dependent packages in the nodebackend/ directory:

```
$ npm install express mongoose body-parser cors –save
```

Next, create the Express web server by going to the nodebackend/ directory, create the server.js file, and add the following code.

```
const express = require("express");
const bodyParser = require("body-parser");
const cors = require("cors");

const app = express();

var corsOptions = {
  origin: "http://localhost:8081"
};
```

```
app.use(cors(corsOptions));

// parse requests of content-type - application/json
app.use(bodyParser.json());

// parse requests of content-type - application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({ extended: true }));

// routes
app.get("/", (req, res) => {
  res.json({ message: "Welcome to the Todo Web App." });
});

require("./routes/todo.routes.js")(app);

// set port, listen for requests
const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}.`);
});
```

Once you use the code to import Express, you can build the REST APIs. The body-parser package is used to create and parse the request object. The cors package is used to serve as middleware for Express that enables CORS. The Express web server will run on port 8080 as the default port. Use the following command to start the server:

```
$ node server.js
```

In a browser, navigate to http://localhost:8080/. The following page shown in Figure 11.31 renders.

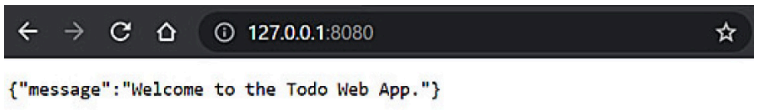


Figure 11.31 Once the Express web server is started, this page should appear at http://localhost:8080/. (rendered in Node; attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Configuring the MongoDB Database and Mongoose

After the Express web server is created, the next step is to configure the MongoDB database and Mongoose. To do this, in the nodebackend/ directory, create the config/ directory. Then, in the config/ directory, create the file db.config.js and add the following code. This specifies the database connection URL to the MongoDB database.

```
module.exports = {
    url: "mongodb://localhost:27017/todo_db"
  };
```

Once the MongoDB connection URL is configured, the next step is to add code to connect to the database

using Mongoose. To do this, in the nodebackend/ directory, create the models/ directory. In the models/ directory, create the index.js file and add the following.

```
const dbConfig = require("../config/db.config.js");

const mongoose = require("mongoose");
mongoose.Promise = global.Promise;

const db = {};
db.mongoose = mongoose;
db.url = dbConfig.url;
db.todos = require("./todo.model.js")(mongoose);

module.exports = db;
```

Next, the nodebackend/server.js file needs to be updated to enable the Express web server to establish a connection with the MongoDB. The code for this is shown in the following snippet.

```
const db = require("./models");
db.mongoose.connect(db.url, {
  useNewUrlParser: true
}).then(() => {
  console.log("Connected to the database successfully!")
}).catch(err => {
  console.log("Cannot connect to the database: " , err);
  process.exit();
});
```

Once the database connection code is completed, the next step is to create the Mongoose model. To do this, in the nodebackend/models/ directory, create the file todo.model.js and add the following code. This code defines a Mongoose schema for the todos model, which results in the creation of a todos collection in the MongoDB database.

```
module.exports = mongoose => {
  var schema = mongoose.Schema({
    title: {
      type: String,
      required: true,
    },
    content: {
      type: String,
      required: false,
    },
    created: {
      type: String,
      default: Date.now(),
      required: true,
    },
    due_date: {
      type: String,
      required: true,
```

```
    },
    category: {
      type: String,
      required: true,
    },
  });

  schema.method("toJSON", function() {
    const { __v, _id, ...object } = this.toObject();
    object.id = _id;
    return object;
  });
  const Todos = mongoose.model("todos", schema);
  return Todos;
};
```

Building the Controller

The next step is to build the controller. The controller contains code that calls the Mongoose CRUD functions to interact with the MongoDB database. To build the controller, in the nodebackend/ directory, create the controllers/ directory. Then, in the controllers/ directory, create the file todo.controller.js and add the code as shown. The code implements the CRUD functions create, findAll, findOne, update, delete, and deleteAll.

```
const db = require("../models");
const Todos = db.todos;

// Create and Save a new Todo item
exports.create = (req, res) => {
  // Validate requred data
  if (!req.body.title) {
    res.status(400).send({ message: "Title can not be empty!" });
    return;
  }
  if (!req.body.category) {
    res.status(400).send({ message: "Category can not be empty!" });
    return;
  }
  if (!req.body.due_date) {
    res.status(400).send({ message: "Due date can not be empty!" });
    return;
  }

  // Create a todo item
  const todo = new Todos({
    title: req.body.title,
    content: req.body.content,
    category: req.body.category,
    due_date: req.body.due_date
  });

  // Save the todo item in the database
```

```
  todo
    .save(todo)
    .then(data => {
      res.send(data);
    })
    .catch(err => {
      res.status(500).send({
        message:
          err.message || "An error occurred while creating the todo item."
      });
    });
};

// Retrieve all todo items
exports.findAll = (req, res) => {
  const title = req.query.title;
  var condition = title ? { title: { $regex: new RegExp(title), $options: "i" } } :
{};

  Todos.find(condition)
    .then(data => {
      res.send(data);
    })
    .catch(err => {
      res.status(500).send({
        message:
          err.message || "An error occurred while retrieving todo items."
      });
    });
};

// Retrieve a todo item by id
exports.findOne = (req, res) => {
  const id = req.params.id;

  Todos.findById(id)
    .then(data => {
      if (!data)
        res.status(404).send({ message: "Error finding todo item with id " + id });
      else res.send(data);
    })
    .catch(err => {
      res
        .status(500)
        .send({ message: "Error retrieving todo item with id=" + id });
    });
};

// Update a todo item by id
exports.update = (req, res) => {
```

```
  if (!req.body) {
    return res.status(400).send({
      message: "Specify a todo item to update. Todo item cannot be empty!"
    });
  }

  const id = req.params.id;

  Todos.findByIdAndUpdate(id, req.body, { useFindAndModify: false })
    .then(data => {
        if (!data) {
        res.status(404).send({
            message: `Cannot update todo item with id=${id}. Todo item was not found!`
        });
        } else res.send({ message: "The todo item was updated successfully." });
    })
    .catch(err => {
        res.status(500).send({
        message: "Error updating todo item with id=" + id
        });
    });
};

// Delete a todo item by id
exports.delete = (req, res) => {
  const id = req.params.id;

  Todos.findByIdAndRemove(id, { useFindAndModify: false })
    .then(data => {
        if (!data) {
            res.status(404).send({
            message: `Cannot delete todo item with id=${id}. Todo item was not found!`
            });
        } else {
            res.send({
            message: "The todo item was deleted successfully!"
            });
        }
        })
        .catch(err => {
        res.status(500).send({
            message: "Error deleting todo item with id=" + id
        });
    });
};

// Delete all todo items from the database
exports.deleteAll = (req, res) => {
    Todos.deleteMany({})
        .then(data => {
```

```
        res.send({
            message: `${data.deletedCount} All todo items were deleted successfully!`
        });
        })
        .catch(err => {
        res.status(500).send({
            message:
            err.message || "An error occurred while deleting all todo items."
        });
        });
};
```

Set Up the REST API

Once the controller is built, the next step is to set up the REST API. A client will send HTTP requests (e.g., GET, POST, PUT, and DELETE) to the REST API endpoints that determine how the server will manage these requests and provide a response. To do this, routes for the REST API are first defined. In the nodebackend/ directory, create the routes/ directory. In the routes/ directory, create the file todo.routes.js and add the following code. In this code, the routes require access to the CRUD functions declared in the controller.

```
module .exports = app => {
  const todos = require("../controllers/todo.controller.js");
  var router = require("express").Router();

  // Create a new todo item
  router.post("/", todos.create);

  // Retrieve all todo items
  router.get("/", todos.findAll);

  // Retrieve a todo item by id
  router.get("/:id", todos.findOne);

  // Update a todo item by id
  router.put("/:id", todos.update);

  // Delete a todo item by id
  router.delete("/:id", todos.delete);

  // Delete all todo items
  router.delete("/", todos.deleteAll);

  app.use("/api/todos", router);

};
```

The next step is to update nodebackend/server.js to import the routes shown in the following code.

```
// routes
app.get("/", (req, res) => {
  res.json({ message: "Welcome to the Todo Web App." });
```

```
});

require("./routes/todo.routes.js")(app);

// set port, listen for requests
const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}.`);
});
```

The next step is to run the Express web server to test the CRUD functions and interact with the MongoDB database. To do this, run the following command:

```
$ node server.js
```

LINK TO LEARNING

As an API platform Postman (https://openstax.org/r/76Postman) can be a useful programming interface to build and use APIs. Postman is simplified, making it easier to build APIs quickly. With Postman, developers can also test and modify APIs to ensure they meet specifications.

To test the REST API, use **Postman**, which is an API platform testing tool that can be used as a client. To accomplish this, follow these steps:

- Install and launch Postman.
- Select POST as the request method.
- Enter the URL http://localhost:8080/api/todos to test creating a todo item.
- In the data input frame, select "Body."
- Select "raw" and "JSON" for the data format.
- Enter a JSON object representing a todo item to be created as shown.
- Click Send.

After you follow these steps, the bottom frame should receive a response with status "200 OK" indicating the request was handled successfully. The body of the created todo item will also display along with a generated id field. You can use this method to test all the CRUD functions.

Creating the React Front End

Once you complete these steps and have the back end of the Todo web application up and running, the next step is to implement the front end using React. This section will outline the steps to accomplish this.

LINK TO LEARNING

React (https://openstax.org/r/76ReactJS) is a JavaScript library popular to build user interfaces. It relies on individual pieces known as components, which are JavaScript functions. React components are created using code and markup, and once created, React components can be combined to develop applications, screens, and web pages. React also has the capability to be added to HTML pages, rather than building an entire page using React.

Creating the React App

To create the React app, the first step is to run the following command. This should be done in a directory that is outside and separate from the nodebackend/ directory.

```
$ npx create-react-app reactfrontend
```

Once you run the command, it will generate the React application files in the reactfrontend/ directory, which is shown in Figure 11.32.

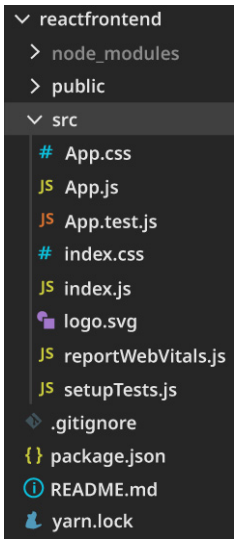


Figure 11.32 This shows the React application files in the reactfrontend/ directory. (rendered in React by Meta Open Source; attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The next step is to navigate into the reactfrontend/ directory and launch the React application to confirm the React front-end application was created successfully. At this point, this application is not connecting to the Node back end, so it launches a default React page. Run the command, which will automatically launch a browser page to http://localhost:3000.

```
$ npm start
```

You will see the following, as illustrated in Figure 11.33.

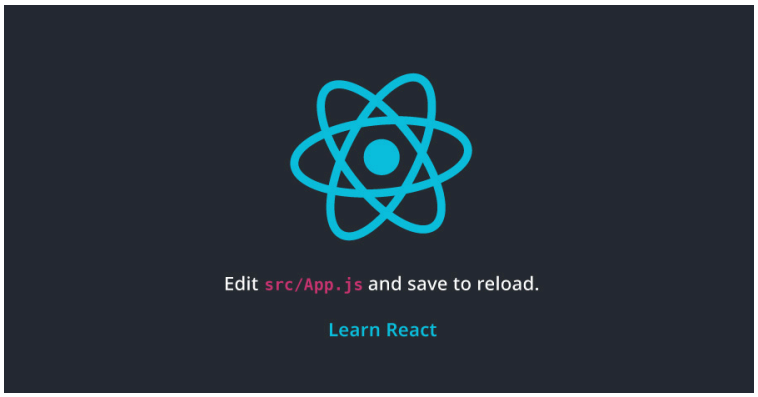


Figure 11.33 When the React front-end application is created successfully, this page launches at http://localhost:3000. (credit: React by Meta Open Source)

Installing Bootstrap and Other Dependencies

The next step is to install the packages for bootstrap and reactstrap, which are necessary to use Bootstrap in a React app. To do this, in the reactfrontend/ directory, run the following commands:

```
$ npm install bootstrap@4.6.0 reactstrap@8.9.0 --legacy-peer-deps
$ npm install react-bootstrap
```

Next, import the Bootstrap[4] CSS file into the React application. To do this, open reactfrontend/src/index.js and add the following Bootstrap import.

```
import React from 'react';
import ReactDOM from 'react-dom';
import 'bootstrap/dist/css/bootstrap.css';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

ReactDOM.render(
  <React.StrictMode>     <App />   </React.StrictMode>,
  document.getElementById('root')
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

Create the React Components

The next step is to create the React components. The React user interface is rendered from the reactfrontend/src/App.js component. The reactfrontend/src/App.js file is generated when the React application is created. This is the main component of the React application, and all other components are added to the reactfrontend/src/App.js component. A few imports are added to the reactfrontend/src/App.js file, as shown in the following code. This includes an import for App.css. Customized CSS rules are added to reactfrontend/src/App.css.

```
//import logo from './logo.svg';
import './App.css';
import React, { Component } from "react";
import Modal from "./components/Modal";
import Nav from "./components/NavComponent";
import axios from "axios";
import { Button, Form, FormGroup, Label, Input } from 'reactstrap';
```

To compare with the Django web application in 11.2 Sample Responsive WAD with Bootstrap and Django, customized CSS rules were added to the style.css file in the static/ directory, as described in Creating the Templates.

Next, look at the following code, which shows App is a class component that extends React's Component class. All state data is added to the this.state variable in the constructor.

```
class App extends Component {
  constructor(props) {
    super(props);
    this.state = {
```

4 Boostrap is a web content framework documented at getbootstrap.com.

```
      todoChecked: false,
      todoList: [],
      categories: [],
      description: "",
      modal: false,
      activeItem: {
        title: "",
        description: "",
        completed: false,
      },
    };
  }
```

The next code snippet shows that every class component must include a `render()` function. This function returns the components that construct the user interface for the Todo web application.

```
render () {
    return(
      <main>         <Nav />         <div className="container mt-5 pl-3">
          <h1>Todo List</h1>           <Form>             <div
className="inputContainer">                  <FormGroup>                   <Label
htmlFor="description">Description</Label>                    <Input type="text"
id="description" name="description"                         placeholder="Description"
                    value={this.state.description}                        />
                </FormGroup>             </div>             <div
className="inputContainer half last">                 <FormGroup>
                  <Label htmlFor="category">Category</Label>                    <Input
id="category" className="taskCategory" type="select" name="category_select"
                    value={this.state.contactType}                       >
                    <option className="disabled" value="">Choose a category</option>
                    <option>Work</option>
                    <option>Personal</option>                   </Input>
                </FormGroup>             </div>             <div
className="inputContainer half last right">               <FormGroup>
                  <Label htmlFor="description">Due Date</Label>
                  <Input type="text" id="description" name="description"
                      placeholder="Due Date (mm/dd/yyyy)"
                      value={this.state.description}                        />
                </FormGroup>             </div>             <div className="row">
                <Button                   className="taskAdd"
                  name="taskAdd" type="submit"                 >
                  Edit                  </Button>                 <Button
                  className="taskDelete ml-1"                  name="taskDelete"
                  type="submit"
                  onclick="$('input#sublist').click();"                 >
                  Delete                  </Button>             </div>
          </Form>           <ul className="taskList">{this.renderItems()}</ul>
        </div>       </main>
    )
  }
```

Connecting the React Front End to the Node Back End

The final step to create the Todo web application is to configure the React application so it can make requests to the API endpoints of the Node application. The React application uses Axios to fetch data by making requests to a given endpoint. To install Axios, run the following command in the reactfrontend/ directory:

```
$ npm install axios@0.21.1
```

Next, add a proxy to the Node application. The proxy will help tunnel API requests from the React application to http://localhost:8080 where the Node application will receive and handle the requests. To do this, open the reactfrontend/package.json file and add the following proxy.

```
{
  "name": "reactfrontend",
  "version": "0.1.0",
  "private": true,
  "proxy": "http://localhost:8080",
  "dependencies": {
```

The next step is to create a service on the front end to send HTTP requests to the back end. This process uses Axios and is similar to how routes were created on the back-end side. The service will export CRUD functions and a finder method to interact with the MongoDB database. To do this, in the reactfrontend/ directory, create the services/ directory. Then, in the services/ directory, create the tile TodoService.js and add the following code.

```
import axios from "axios";

const getAll = () => {
  return axios.get("/api/todos/");
};

const get = id => {
  return axios.get(`/api/todos/${id}`);
};

const create = data => {
  return axios.post("/api/todos", data);
};

const update = (id, data) => {
  return axios.put(`/api/todos/${id}`, data);
};

const remove = id => {
  return axios.delete(`/api/todos/${id}`);
};

const removeAll = () => {
  return axios.delete(`/api/todos`);
```

```
};

export default {
  getAll,
  get,
  create,
  update,
  remove,
  removeAll
};
```

Finally, to complete this step, update reactfrontend/App.js to the following to call the services.

```
handleSubmit = (item) => {
  var data = {
    id: item.id,
    title: item.title,
    content: item.content,
    due_date: item.due_date,
    category: item.category
  };

  TodoService.create(data)
    .then((res) => this.refreshList())
    .catch((err) => console.log(err));
};
handleUpdate = (item) => {
  var data = {
    id: item.id,
    title: item.title,
    content: item.content,
    due_date: item.due_date,
    category: item.category
  };
  TodoService.update(item.id, data)
    .then((res) => this.refreshList())
    .catch((err) => console.log(err));
};
handleDelete = (item) => {
  TodoService.remove(item.id)
    .then((res) => this.refreshList())
    .catch((err) => console.log(err));
};
refreshList = () => {
  TodoService.getAll()
  .then((res) => this.setState({ todoList: res.data }))
  .catch((err) => console.log(err));
}
```

Once this is completed, run both the Express web server and the React app using the following commands:

```
$ node server.js
$ npm start
```

When this is done, use the form shown in Figure 11.34 to create a new todo item.

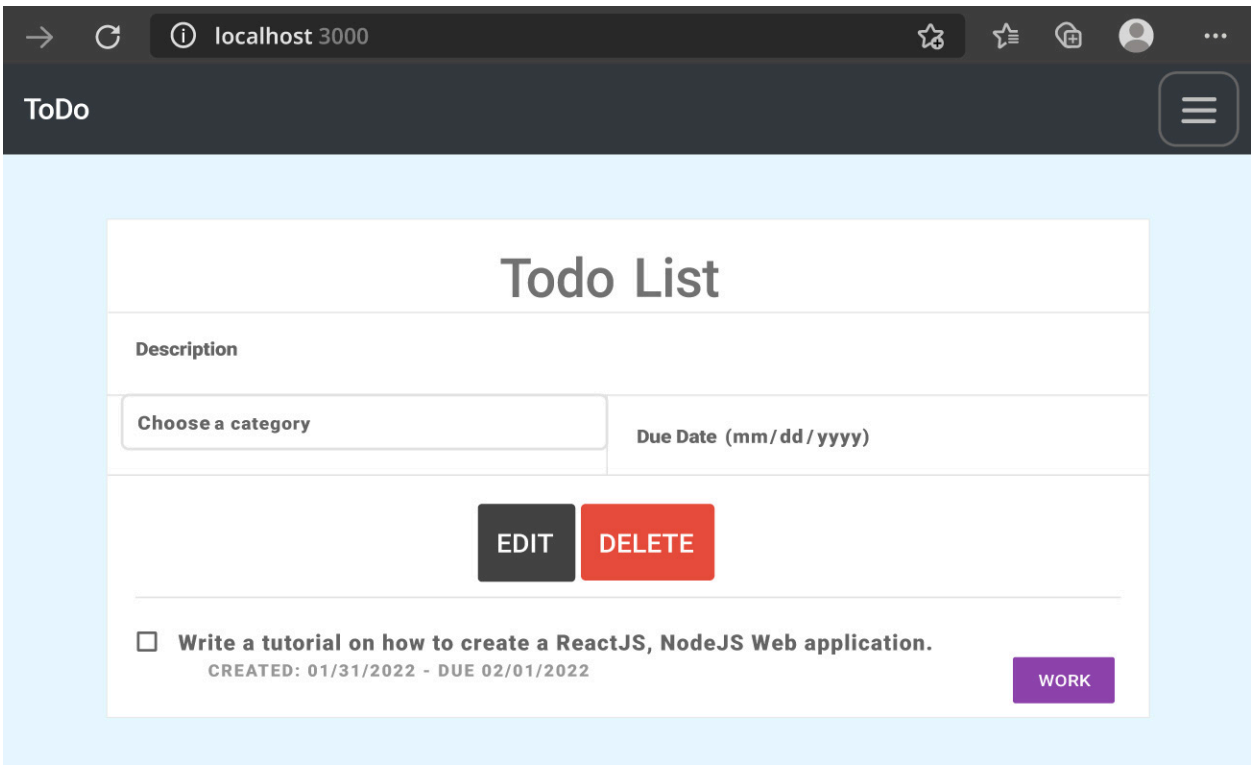


Figure 11.34 Once the Todo web application is created using Bootstrap with React and Node, this form can be used to create todo items. (rendered using Bootstrap, under MIT license copyrighted 2018 Twitter, with React by Meta Open Source and Node; attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

11.4 Sample Responsive WAD with Bootstrap/React and Django

Learning Objectives

By the end of this section, you will be able to:

- Update the Todo web application with Bootstrap, React, and Django
- Extend the Django back end to support the React front end
- Connect the React front end to the Django back end

Previously, you learned how to build a Todo web application using Bootstrap and Django and then using Bootstrap with React and Node. This section will review the steps required to update the Todo web application using Bootstrap with React and Django.

Updating the Todo Web Application with Bootstrap, React, and Django

In this section, the Todo web application implemented will build on the Django application covered in 11.2 Sample Responsive WAD with Bootstrap and Django, as well as the React application explored in 11.3 Sample Responsive WAD with Bootstrap/React and Node. For this version of the Todo web application, React serves as the front end handling the user interface to get and set data via HTTP requests. Django serves as the back end.

Prerequisites

To build this version of the Todo application, you need Python v3.9.4, PIP v21.3.1, Django v4.0.1, Django REST Framework v3.13.1, Bootstrap v4.5.0, Django-cors-headers v3.11.0, React v17.0.2, and Axios v0.21.0. To begin, complete the following steps:

- Activate the venv used in section 11A.2:
  `$ cd py394venv`
  Windows: `$ source ./Scripts/activate`
  macOS: `$ source ./bin/activate`
- Install in the local programming environment Django Cors Headers
  `$ pip install django-cors-headers==3.11.0 ]`

LINK TO LEARNING

When using Django, Cross-Origin Resource Sharing (CORS) headers (https://openstax.org/r/76CORSheaders) can be a valuable tool to allow a Django application to accept in-browser requests that come from other origins. With CORS headers, other domains can access your resources, but only if permitted. When used appropriately, this makes CORS an important tool to boost a website's security.

Extending the Django Back End to Support the React Front End

To create a Todo web application using React as a front end to the Django back end, the Django project requires a couple of configurations. Open TodoApp/settings.py and add 'corsheaders' to INSTALLED_APPS as in the following code.

```
INSTALLED _APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'corsheaders',
    'rest_framework',
    'todo',
]
```

By configuring the Django project with CORS, the Django application will be allowed to accept in-browser requests that come from other origins. To add CORS, add the CORS middleware to MIDDLEWARE, as shown in the following code.

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'corsheaders.middleware.CorsMiddleware',
]
```

After completing this step, scroll to the bottom of the settings.py file and add the following variable.

```
# add CORS whitelist for localhost:3000 since the React frontend will be served on
port 3000
```

```
CORS_ORIGIN_WHITELIST = [
    'http://localhost:3000'
]
```

LINK TO LEARNING

Axios is an HTTP client for Node that manages asynchronous HTTP requests. Axios is free and open-source, with built-in security measures. Axios uses clean, efficient syntax to manage promises, and works well with Node, as well as browser environments. Visit this page on Axios (https://openstax.org/r/76Axios) to learn more.

Connecting the React Front End to the Django Back End

Next, the React application must be configured so it can make requests to the API endpoints of the Django application. The React application uses Axios to fetch data by making requests to a given endpoint. To install Axios, run the following command in the reactfrontend/ directory:

```
$ npm install axios@0.21.1
```

Next, add a proxy to the Django application. The proxy will help tunnel API requests from the React application to http://localhost:8000, where the Django application will receive and handle the requests. To add the proxy, open the reactfrontend/package.json file and add the following code.

```
{
  "name": "reactfrontend",
  "version": "0.1.0",
  "private": true,
  "proxy": "http://localhost:8080",
  "dependencies": {
```

After completing this step, open the reactfrontend/src/App.js file and import Axios, as shown in the following code.

```
import './App.css';
import React, { Component } from "react";
import Modal from "./components/Modal";
import Nav from "./components/NavComponent";
import axios from "axios";
import { Button, Form, FormGroup, Label, Input } from 'reactstrap';
```

To update the reactfrontend/src/App.js file, add the following code. The `handleSubmit()` function will use Axios to make requests to the Django API endpoints to create and delete todo items.

```
handleSubmit = (item) => {
  this.toggle();

  if (item.id) {
    axios
      .put(`/api/todos/${item.id}/`, item)
      .then((res) => this.refreshList());
    return;
```

```
  }
  axios
    .post("/api/todos/", item)
    .then((res) => this.refreshList());
};

handleDelete = (item) => {
  axios
    .delete(`/api/todos/${item.id}/`)
    .then((res) => this.refreshList());
};
```

When these steps are complete, start up the Django server and then start up the React application, using the following commands, respectively.

```
$ python manage.py runserver
$ npm start
```

To access the Django REST API, navigate to http://localhost:8000/api/. This is similar to the steps completed in 11.2 Sample Responsive WAD with Bootstrap and Django. Click on the categories API path to enter two categories, as seen in Figure 11.35.

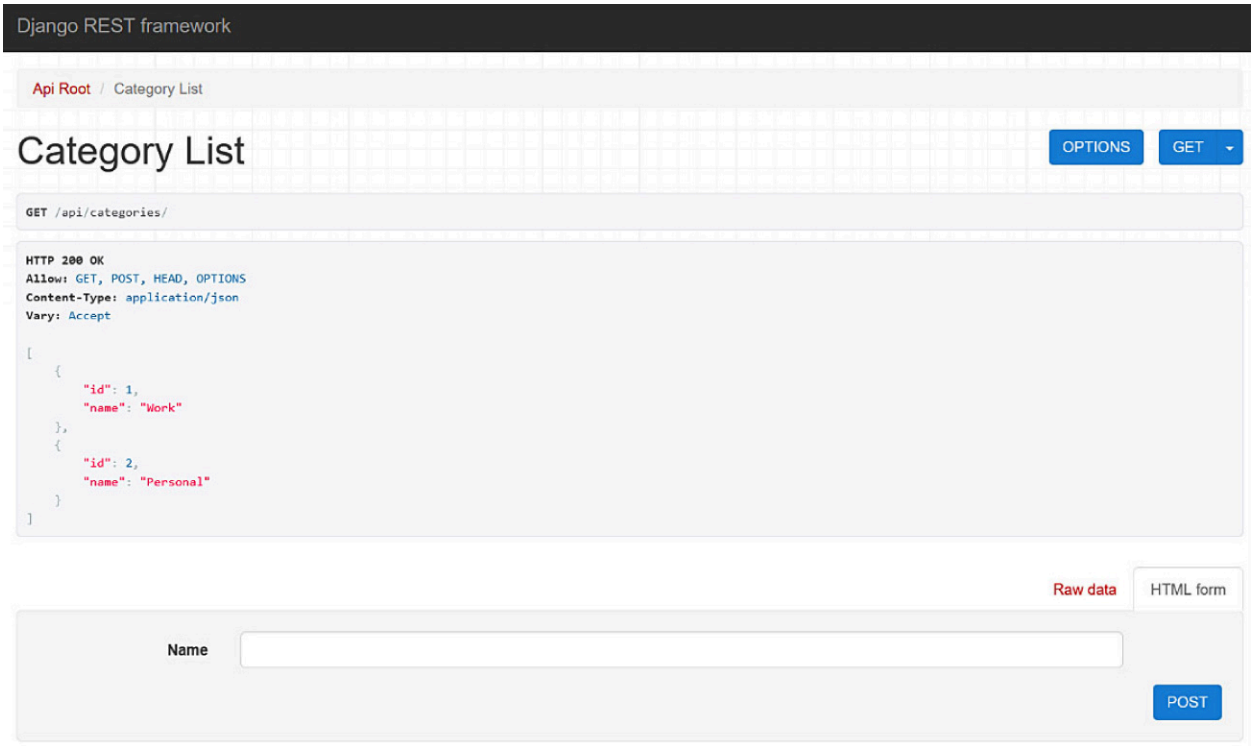


Figure 11.35 After navigating to http://localhost:8000/api/ and following the instructions provided, this page will appear to allow access to the Django REST API. (rendered in React by Meta Open Source; attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

When the React application starts up, a browser page will automatically launch for navigating to http://localhost:3000 and the user interface should render. Figure 11.36 shows the page on which to create a todo item.

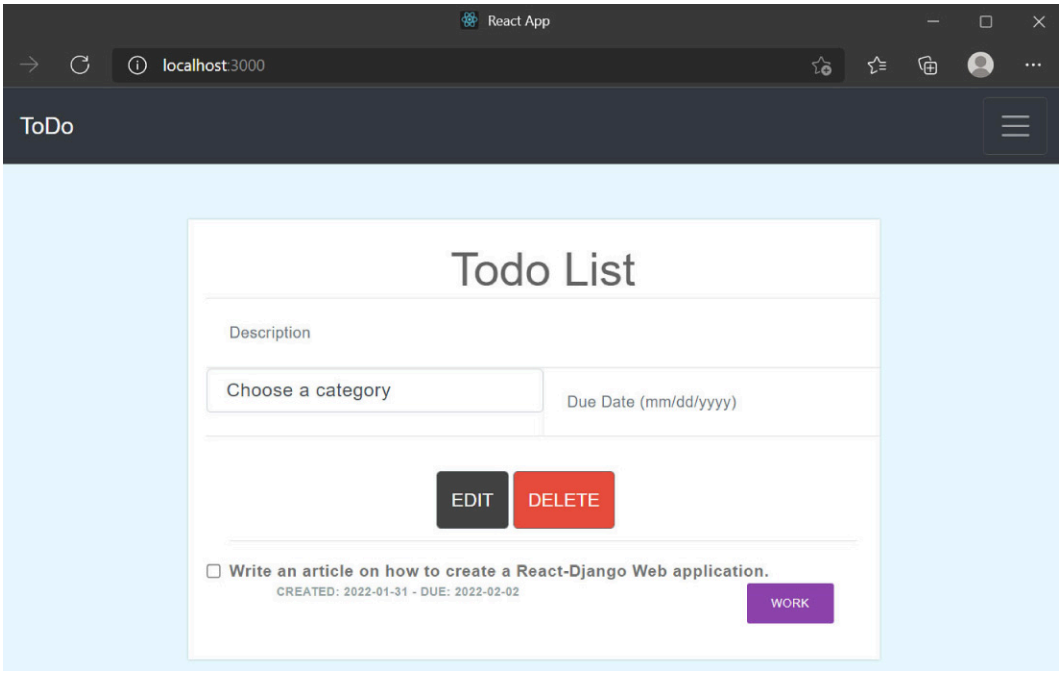


Figure 11.36 After the React application starts up, this user interface will appear. (rendered in React by Meta Open Source; attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

At this point, revisit the Django REST API. Figure 11.37 shows that the todo item should be accessible via the todos API path.

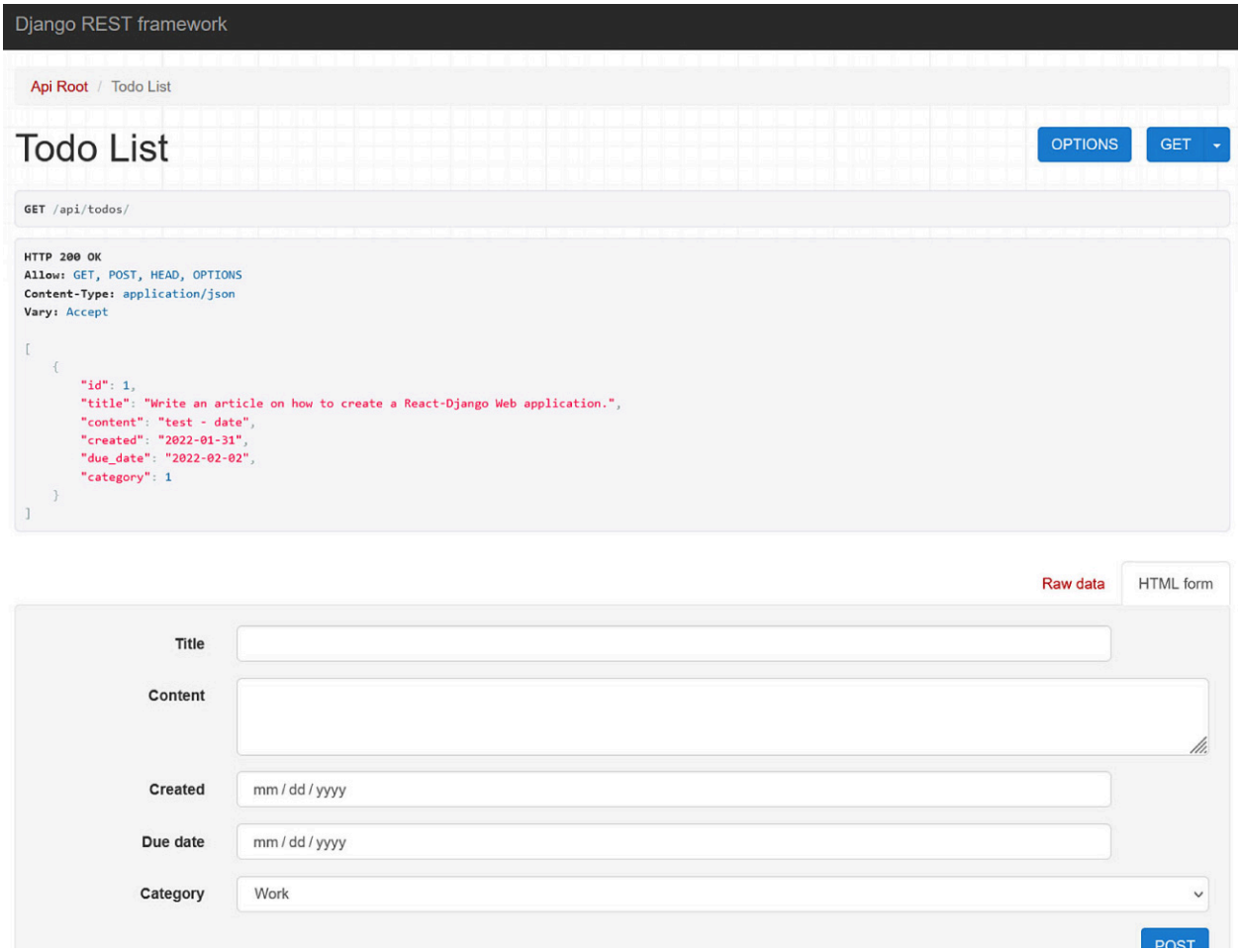


Figure 11.37 Once the Todo web application is updated using Bootstrap with Django and React, this page should appear to allow users to create a Todo List. (rendered using Bootstrap under MIT license copyrighted 2018 Twitter; with Django, a registered trademark of the Django Software Foundation; and React by Meta Open Source; attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

11.5 Sample Native WAD with React Native and Node or Django

Learning Objectives

By the end of this section, you will be able to:

- Create a Todo native mobile application with React Native or Node
- Create a React Native app and its components
- Connect the front-end Native app with the back-end Node app

In the previous sections, you worked with Bootstrap, Django, React, and Node to build versions of a Todo web application. In this section, you will learn how to use React Native and Node to develop a Todo application for mobile devices. You are already familiar with Node. Like React, **React Native** is an open-source JavaScript framework used to build native applications for mobile devices.

LINK TO LEARNING

React Native (https://openstax.org/r/76ReactNative) uses the React JavaScript library to enable native development and build user interfaces for mobile devices. React Native has the flexibility to work with Xcode, which is the IDE for various platforms. Released by Facebook (now Meta) in 2015, React Native has become increasingly popular among developers.

Creating a Todo Web Application with React Native and Node

This application uses React Native and Node to implement a simple Todo mobile application. This application builds on top of the React and Node application discussed in 11.3 Sample Responsive WAD with Bootstrap/ React and Node. For this version of the application, React Native serves as the front end handling the user interface and getting and setting data via HTTP requests. Node serves as the back end that makes use of the API built using the Django REST Framework in 11.2 Sample Responsive WAD with Bootstrap and Django.

Prerequisites

To build the Todo mobile application using React Native and Node, you need React Native v0.67, Node v14.17.5, ExpressJS v4.17.2, MongooseJS v6.1.9, and Axios v0.21.0. This Todo application will run using the Android emulator and will use Android Studio v2021.1.1.

LINK TO LEARNING

Xcode (https://openstax.org/r/76Xcode) is the IDE for the Apple platform. For Apple, React Native provides developers with the tools needed for cross-platform development that creates user-friendly apps on mobile devices.

- To begin, download and install an emulator for your intended platform (iOS or Android).
- If you plan to run the native app on an iOS device or emulator, download and install **Xcode**, which is Apple's IDE that enables application development for Apple's platforms.
- If you plan to run the native app on an Android device or emulator, download and install Android Studio, which enables application development for Android mobile operating systems.
- Figure 11.38 shows how to set up the Android emulator. Launch Android Studio. From the top navigation, select Tools > Device Manager. Click on Create device. In the Select Hardware pop-up, select Pixel 5 and click Next. On the next page, select Pie Download. Click on the Download link to obtain an image as shown. Click Next followed by Finish.

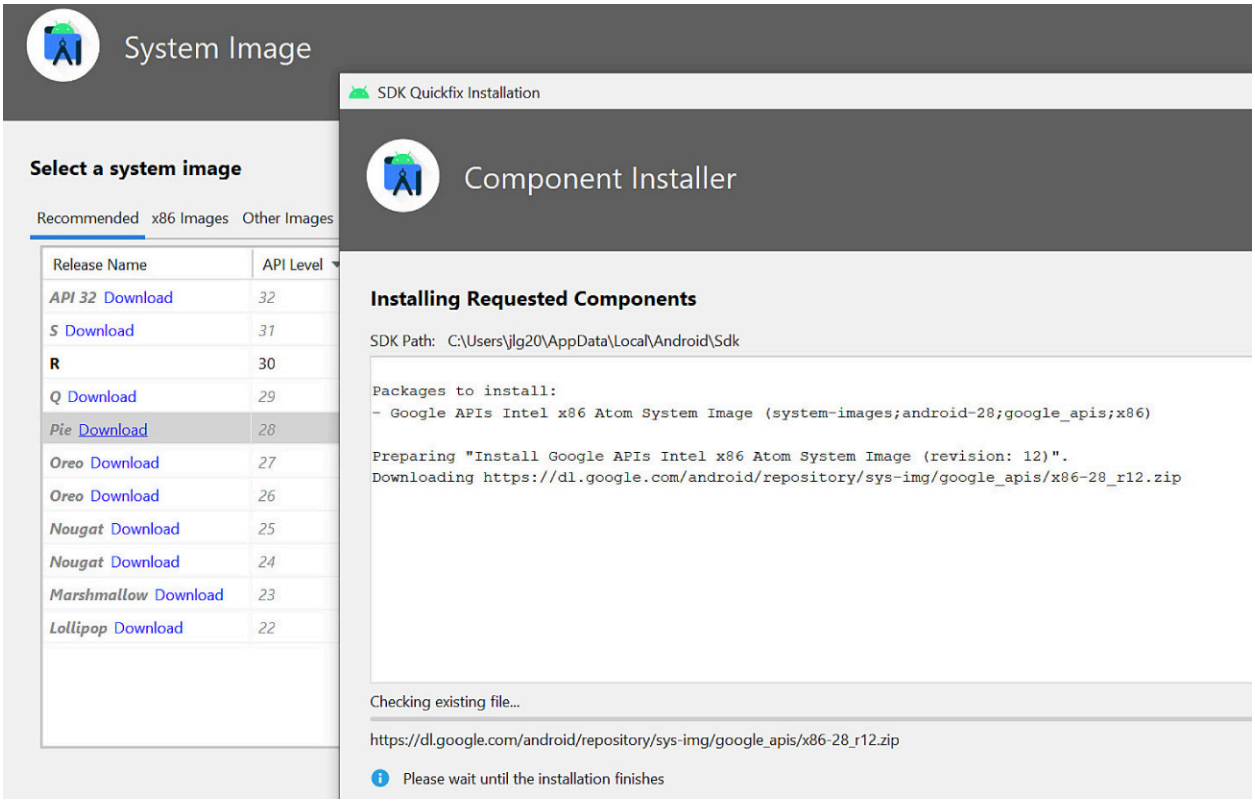


Figure 11.38 This shows how to set up the Android emulator. (Android Studio is a trademark of Google LLC.)

Once the image has been created, it will appear in the Device Manager. Next, click on the Start button to launch the emulator. When the emulator is launched, it will turn on, as shown in Figure 11.39.

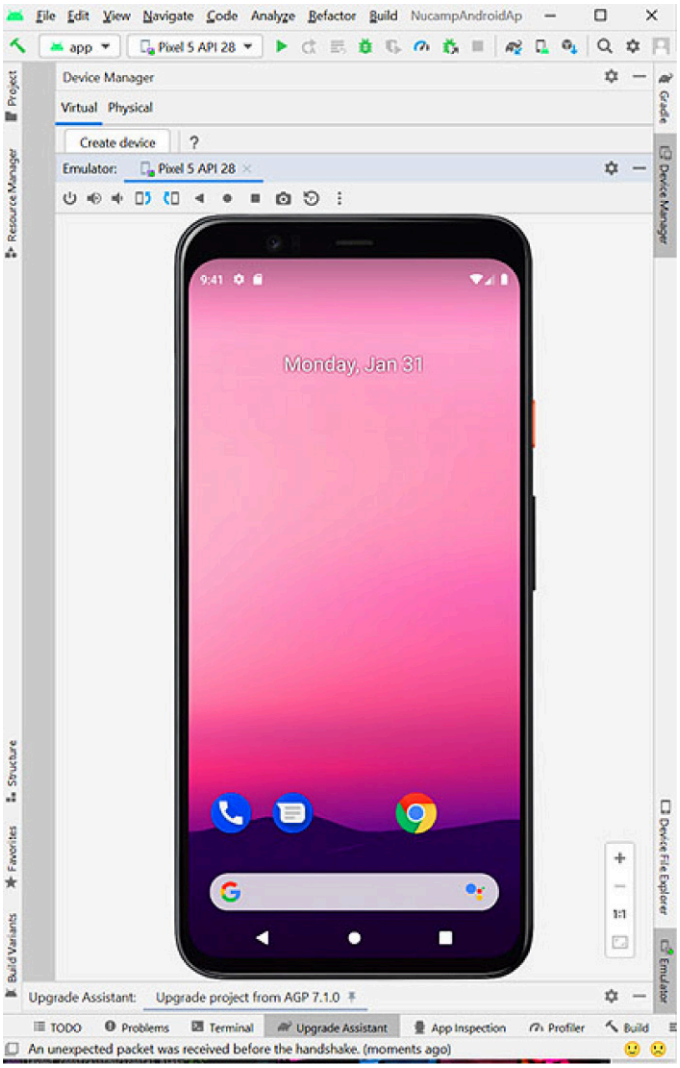


Figure 11.39 This is how the emulator will appear after it is launched. (Android Studio is a trademark of Google LLC.)

LINK TO LEARNING

Xcode also integrates well with Android Studio (https://openstax.org/r/76AndroidStud) for developing Android apps. Android Studio is the IDE for Android devices.

Creating the React Native App

After launching the emulator, create the React Native app by running the following command in a directory outside and separate from the nodebackend/ directory. Figure 11.40 shows the React Native application files in the reactnativefrontend/ directory.

```
$ npx react-native init reactnativefrontend
```

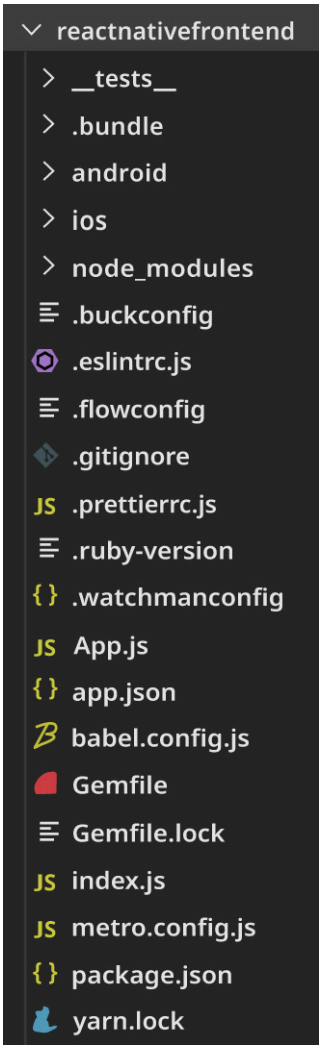


Figure 11.40 When the React Native app is created, it will generate the React Native application files in the reactnativefrontend/ directory. (rendered in React Native, under MIT license; attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Next, navigate into the reactnativefrontend/ directory and launch the React Native application to confirm that the React Native front-end application has been created successfully. At this point, the application is not connected to the Node back end and will launch the **Metro bundler**, which bundles the JavaScript code that is deployed on the mobile device or emulator when the React Native front-end application is successfully completed. When this is done, run the command. Figure 11.41 shows the page in the terminal.

```
$ npx react-native start
```

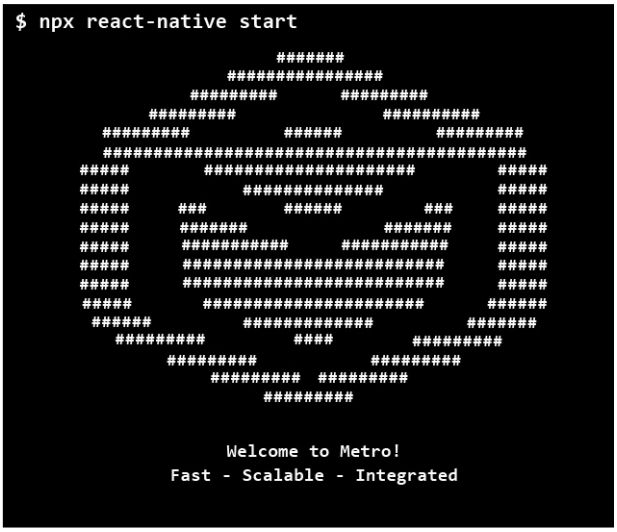


Figure 11.41 This page will appear when the react-native start command is run. (credit: React Native, under MIT license)

The next step is to open another terminal, navigate to the reactnativefrontend/ directory, and run the following command. This will build the front-end code and deploy it on the emulator, which may take a few minutes.

```
$ npx react-native run-android
```

When the application is successfully built, Figure 11.42 shows what should appear in the second terminal.

```
$ npx react-native run-android
info Running jetifier to migrate libraries to Androidx. You can disable it using
"--no-jetifier" flag.
Jetifier found 863 file(s) to forward-jetify. Using 12 workers...
info JS server already running.
info Installing the app...
Downloading https://services.gradle.org/distributions/gradle-7.2-all.zip
.............10%.............20%.............30%.............40%.............50%..
...........60%.............70%.............80%.............90%.............100%

Welcome to Gradle 7.2!

Here are the highlights of this release:
  - Toolchain support for Scala
  - More cache hits when Java source files have platform-specific line endings
  - More resilient remote HTTP build cache behavior

For more details see https://docs.gradle.org/7.2/release-notes.html

Starting a Gradle Daemon, 1 incompatible Daemon could not be reused, use --status
for details

> Task :app:compileDebugJavaWithJavac

> Task :app:stripDebugDebugSymbols
Unable to strip the following libraries, packaging them as they are: libbetter.so
, libc++_shared.so, libevent-2.1.so, libevent_core-2.1.so, libevent_extra-2.1.so,
libfabricjni.so, libfb.so, libfbjni.so, libflipper.so, libfolly_futures.so, libfo
lly_json.so, libglog.so, libglog_init.so, libimagepipeline.so, libjsc.so, libjsce
xecutor.so, libjsi.so, liblibjsijniprofiler.so, libjsinspector.so, liblogger.so,
libmapbufferjni.so, libnative-filters.so, libnative-imagetranscoder.so, libreact_
codegen_rncore.so, libreact_debug.so, libtract_nativemodule_core.co, libreact_ren
der_animation.so, libreact_render_attributedstring.so, libreact_render_componentr
egistry.so, libreact_render_core.so, libreact_render_debug.so, libreact_render_gr
aphics.so, libreact_render_imagemanager.so, libreact_render_leakchecker.so, libre
act_render_mapbuffer.so, libract_render_mounting.so, libreact_render_runtimesched
uler.so, libreact_render_scheduler.so, libreact_render_telemetry.so, libreact_ren
der_templateprocessor.so, libreact_render_textlayoutmanager.so, libreact_render_u
imanager.so, libreact_utils.so, libreactconfig.so, libreactnativeblob.so, libreac
tnativejni.so, libreactnativeutilsj

> Task :app:installDebug
Installing APK 'app-debug.apk' on 'Pixel_s_API_28(AVD) - 9' for app:debug
Installed on 1 device.

BUILD SUCCESSFUL in 1m 29s
31 actionable tasks: 31 executed
info Connecting to the development server...
info Starting the app on "emulator-5554"...
Starting: Intent { cmp=com.reactnativefrontend/.MainActivity }
```

Figure 11.42 This code should appear in the second terminal when the React Native application is successfully built. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Figure 11.43 shows how the native application should appear in the emulator.

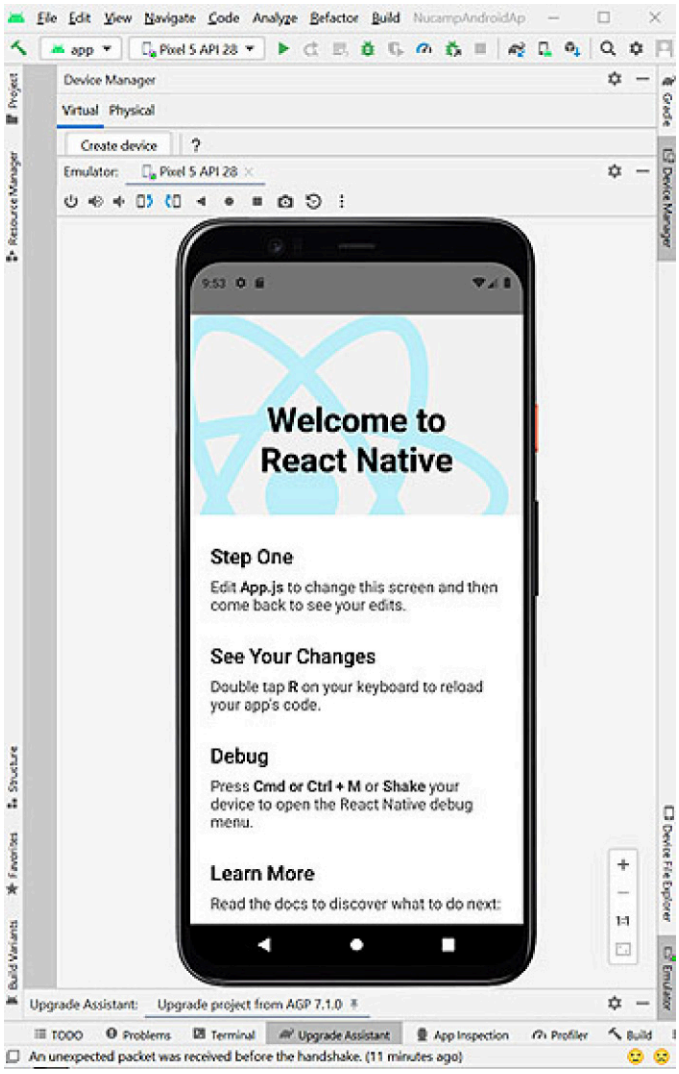


Figure 11.43 Once the React Native application is successfully built, this is how the native application should appear in the emulator. (Android Studio is a trademark of Google LLC.)

Creating the React Native App Components

The packages used by the React Native app for navigation and other features are different from the packages used by React. To install the required packages, in the reactnativefrontend/ directory, run the following list of commands:

```
$ npm install @react-navigation/native@6.0.7
$ npm install @react-navigation/native-stack@6.3.0
$ npm install axios@0.25.0
$ npm install moment@2.29.1
$ npm install react-native-modal@13.0.0
$ npm install react-native-safe-area-context@3.3.2
$ npm install react-native-screens@3.10.2
$ npm install react-native-snackbar@2.4.0
$ npm install react-native-vector-icons@9.0.0
```

The first step is to create the Todo component. In the reactnativefrontend/ directory, create the directory src/Screens/. In the src/Screens/ directory, create the file TodoList.js. The following code shows the imports required to create the screen components.

```
import React, {useState, useEffect} from 'react';
import {
  View,
  Text,
  StyleSheet,
  TextInput,
  FlatList,
  TouchableOpacity,
  Dimensions,
  StatusBar,
  Alert,
} from 'react-native';
import axios from 'axios';
import {COLORS} from '../utils/colors';
import MaterialIcon from 'react-native-vector-icons/MaterialIcons';
import MaterialCommunityIcon from 'react-native-vector-icons/MaterialCommunityIcons';
import Modal from 'react-native-modal';
import Snackbar from 'react-native-snackbar';
import {duration} from 'moment';
```

Once this is done, the following code includes the `return()` function that renders the screen components.

```
return (
  <View style={styles.container}>     <StatusBar backgroundColor={COLORS.DARKALT} />
    <Text style={styles.heading}>Welcome to Todo List App!</Text>     <Text
style={styles.heading}>Your Tasks</Text>     {todos.length == 0 ? (       <View
style={styles.center}>         <MaterialCommunityIcon           name="note-multiple"
          size={90}           color={COLORS.LIGHTALT}         />         <Text
style={styles.noTasksText}>No Tasks Added</Text>       </View>     ) : (       <>
        <View style={styles.todosContainer}>           <FlatList
            data={todos}             renderItem={({item}) => (               <View
style={styles.todo}>                 <Text                   style={item.complete ?
styles.completedStyle : styles.text}>                   {item.title}
               </Text>                 <TouchableOpacity
                 style={styles.deleteTodo}                   onPress={() =>
deleteTodo(item._id)}>                   <Text style={{color: COLORS.WHITE}}>X</Text>
               </TouchableOpacity>               </View>             )}           />
        </View>         <TouchableOpacity           style={styles.addButton}
          onPress={() => {             setModalActive(true);           }}>
          <MaterialIcon name="add" size={32} color={COLORS.LIGHT} />
        </TouchableOpacity>       </>
    )}

    <Modal
      isVisible={modalActive}
      animationIn={'slideInUp'}
      animationOut={'slideInDown'}>
      <View style={styles.modalView}>         <TouchableOpacity
          style={styles.modalCloseBtn}           onPress={() =>
setModalActive(false)}>           <Text style={styles.modalCloseBtnText}>X</Text>
```

```
        </TouchableOpacity>         <Text style={styles.addTaskHeading}>Add
Task</Text>         <TextInput           style={styles.taskInput}
          placeholder="Enter task here.."           onChangeText={text =>
setNewTodo(text)}           value={newTodo}         />         <TouchableOpacity
style={styles.createBtn} onPress={() => addTodo()}>           <Text
style={styles.createBtnText}>Create Task</Text>         </TouchableOpacity>
      </View>
    </Modal>
  </View>
);
```

The following code creates the modal pop-up screen that is used to create a Todo list.

```
<Modal
  isVisible={modalActive}
  animationIn={'slideInUp'}
  animationOut={'slideInDown'}>
  <View style={styles.modalView}>     <TouchableOpacity
      style={styles.modalCloseBtn}       onPress={() => setModalActive(false)}>
      <Text style={styles.modalCloseBtnText}>X</Text>     </TouchableOpacity>
    <Text style={styles.addTaskHeading}>Add Task</Text>     <TextInput
      style={styles.taskInput}       placeholder="Enter task here.."
      onChangeText={text => setNewTodo(text)}       value={newTodo}     />
<TouchableOpacity style={styles.createBtn} onPress={() => addTodo()}>       <Text
style={styles.createBtnText}>Create Task</Text>     </TouchableOpacity>   </View>
</Modal>
```

Next, update the reactnativefrontend/App.js file, as shown in the following code, to render the screen components declared in the TodoList.js file.

```
import React from 'react';
import {NavigationContainer} from '@react-navigation/native';
import {createNativeStackNavigator} from '@react-navigation/native-stack';
import TodoList from './src/Screens/TodoList';

const Stack = createNativeStackNavigator();

const App = () => {
  return (
    <NavigationContainer>       <Stack.Navigator>         <Stack.Screen
          name="TodoList"           component={TodoList}
          options={{headerShown: false}}         />       </Stack.Navigator>
    </NavigationContainer>
  );
};
export default App;
```

When this is done, relaunch the Metro bundler and deploy the native application by running the following commands:

```
$ npx react-native start
```

```
$ npx react-native run-android
```

Next, create a Todo item (Figure 11.44).

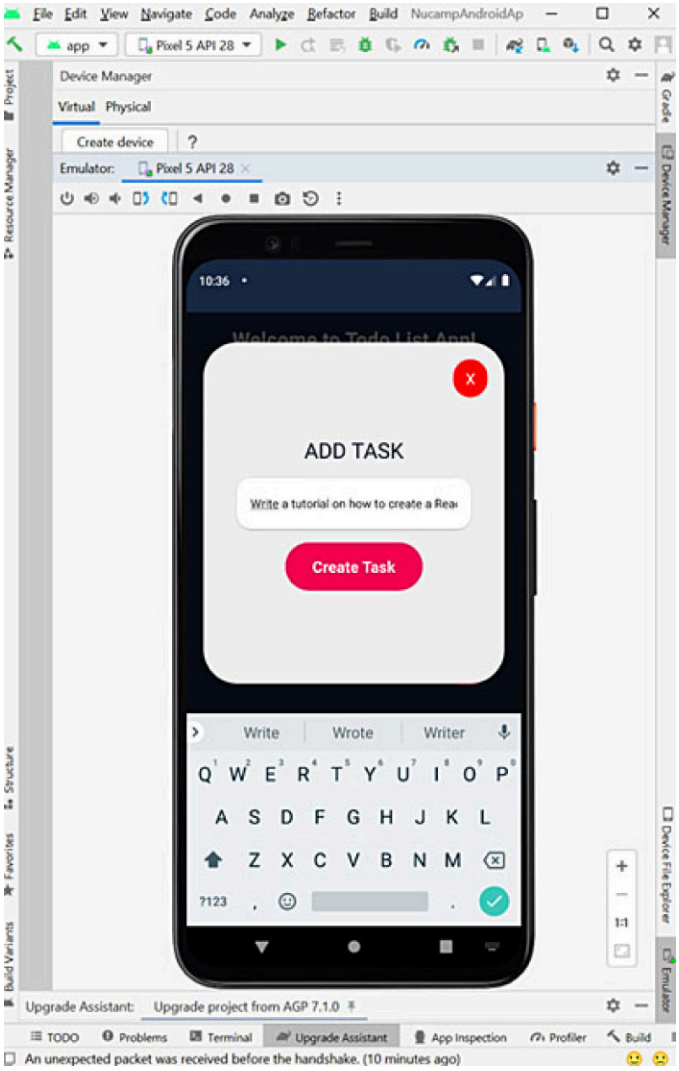


Figure 11.44 After relaunching the Metro bundler and deploying the native application, this screen allows users to create a Todo item. (Android studio is a trademark of Google LLC.)

After the Todo item is created, it should appear on the main screen as seen in Figure 11.45.

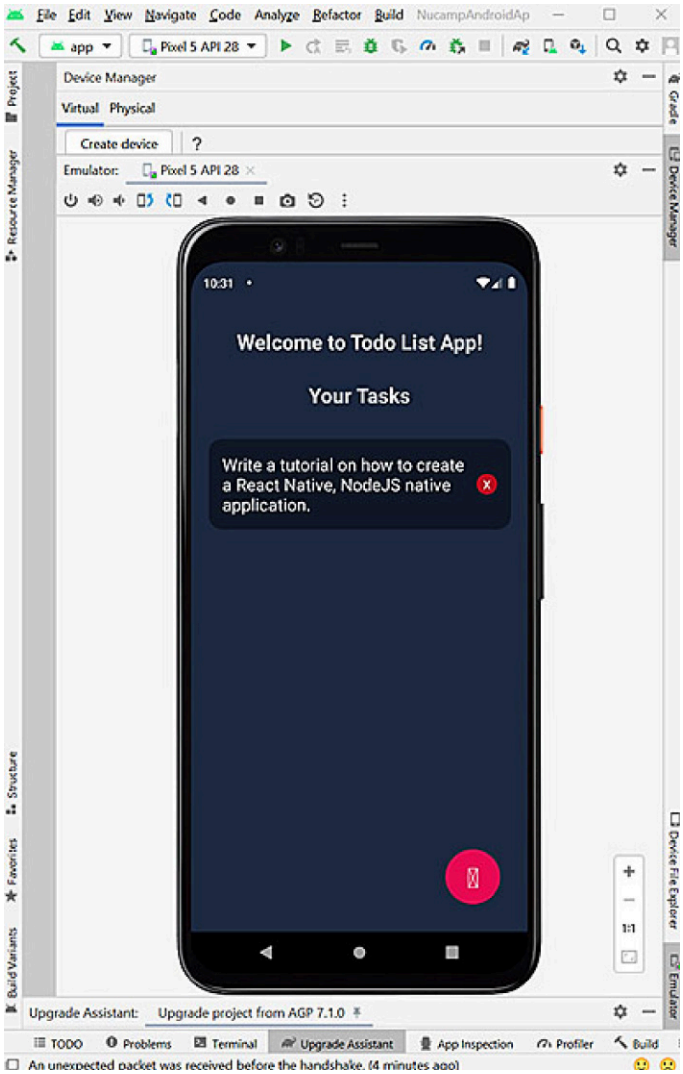


Figure 11.45 Here is how the Todo item should appear on the main screen. (Android studio is a trademark of Google LLC.)

Connecting the Front-End Native App with the Back-End Node App

The next step is to connect the front-end React Native app with the back-end Node app. To do this, open reactnativefrontend/src/Screens/Todolist.js. Add the API_BASE variable and configure the IP address to the local computer, running the Express web server, as shown in the following code.

```
import axios from 'axios';
import {COLORS} from '../utils/colors';
import MaterialIcon from 'react-native-vector-icons/MaterialIcons';
import MaterialCommunityIcon from 'react-native-vector-icons/MaterialCommunityIcons';
import Modal from 'react-native-modal';
import Snackbar from 'react-native-snackbar';
import {duration} from 'moment';

const width = Dimensions.get('window').width;

const API_BASE = 'http://192.168.1.183:8080';
```

Next, add the Axios calls to interact with the REST API provided via the Express web server to interact with the MongoDB database.

```
const [todos, setTodos] = useState([]);
  const [modalActive, setModalActive] = useState(false);
  const [newTodo, setNewTodo] = useState('');

  useEffect(() => {
    GetTodos();
  }, [todos]);

  const GetTodos = () => {
    axios
      .get(`${API_BASE}/api/todos`)
      .then(response => {
        setTodos(response.data);
      })
      .catch(err => {
        console.error('Error: ', err);
        Snackbar.show({
          text: '' + err,
          duration: Snackbar.LENGTH_LONG,
          backgroundColor: 'red',
          textColor: COLORS.WHITE,
        });
      });
  };

  const completeTodo = async id => {
    const data = await axios.put(`${API_BASE}/api/todos/${id}`);

    setTodos(todos =>
      todos.map(todo => {
        if (todo._id === data._id) {
          todo.complete = data.complete;
        }

        return todo;
      }),
    );
  };

  const deleteTodo = async id => {
    const data = await axios.delete(`${API_BASE}/api/todos/${id}`);

    setTodos(todos => todos.filter(todo => todo._id !== data._id));
  };

  const addTodo = async () => {
    if (newTodo == '') {
      Alert.alert('Error!', 'Please enter a task first!');
      return;
    } else {
```

```
      await axios
        .post(`${API_BASE}/todo/new`, {
          text: newTodo,
        })
        .then(function (response) {
          const data = response.data;
          setTodos([...todos, data]);
          setModalActive(false);
          setNewTodo('');
        })
        .catch(function (error) {
          console.log('Error: ', error);
        });
    }
  };
```

11.6 Sample Ethereum Blockchain Web 2.0/Web 3.0 Application

Learning Objectives

By the end of this section, you will be able to:

- Build a hybrid Ethereum blockchain Web 3.0 application
- Create the React app and install dependencies
- Create the smart contract
- Create the front-end React components
- Add Web3 to the React app
- Configure the React app to communicate with the smart contract

So far, this chapter has explored how to develop a Todo web application, as well as a Todo mobile application, using Bootstrap, Django, React, React Native, and Node. In this final section, you will learn how to create a simple Todo application using React with Web 3.0 powered by Ethereum smart contracts on the blockchain.

Creating a Todo Ethereum Blockchain Web 3.0 Application

As you have learned, Todo applications can be created using different tools. In this section, you will explore the **Ethereum blockchain**, which creates a secure peer-to-peer network through the use of a smart contract, which is a secure digital agreement that enables users to transact directly with each other via the blockchain. You will use React to create a Todo Ethereum blockchain Web 3.0 application.

LINK TO LEARNING

Ganache (https://openstax.org/r/76Ganache) is a personal Ethereum blockchain environment that provides developers with the means to use a private blockchain for development and testing. Ganache can emulate the Ethereum blockchain, and because it is a private blockchain, it allows developers control over development and testing processes.

Prerequisites

To build the Todo Ethereum blockchain Web 3.0 application, you should use React v17.0.2, Bootstrap v4.5.0, Node v14.17.5, Web 3.js v1.2.2, Truffle v5.0.2, and Solidity v0.8.11. In addition, Ganache is used as the personal blockchain for development. To begin, do the following:

- Download and install Ganache. Launch Ganache and choose the quick start Ethereum option as seen in Figure 11.46.

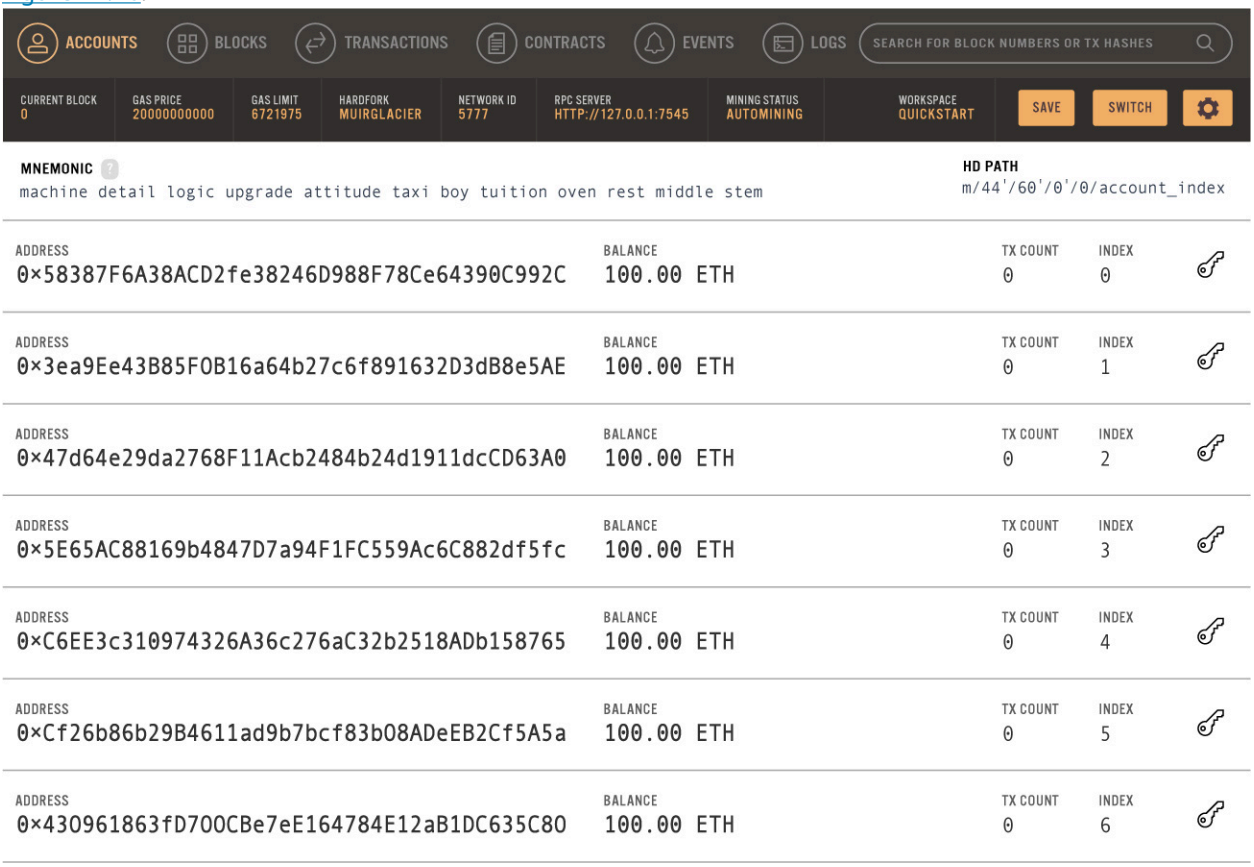


Figure 11.46 This is the quick start Ethereum option that is available after installing Ganache. (credit: Ganache, under MIT license)

- The next step is to install the MetaMask Chrome plug-in. Configure the MetaMask account and log in.

Creating the React App

Previously in this chapter, you created React apps. For this application, you need to create a new React app.

To create the new React app, run the following command:

```
$ npx create-react-app ethreact
```

Install Bootstrap, Web3, and Other Dependencies

To use Bootstrap and Web3 in a React app, corresponding packages must be installed. To do this, in the ethreact/ directory, run the following commands:

```
$ npm install bootstrap@4.6.0 reactstrap@8.9.0 --legacy-peer-deps
$ npm install react-bootstrap
$ npm install web3@1.2.2
```

Next, import the Bootstrap CSS file into the React application. Open ethreact/src/index.js and add the following Bootstrap import.

```
import React from 'react';
import ReactDOM from 'react-dom';
```

```
import 'bootstrap/dist/css/bootstrap.css';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

ReactDOM.render(
  <React.StrictMode>     <App />   </React.StrictMode>,
  document.getElementById('root')
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

LINK TO LEARNING

Truffle (https://openstax.org/r/76Truffle) provides a suite of tools that can be used to develop smart contracts. Truffle offers end-to-end development that includes the ability to develop, test, and implement smart contracts, while using Truffle to manage the workflow.

Creating the Smart Contract

To develop the Ethereum smart contract, you will use the **Truffle Framework**, which is a popular suite of tools used to develop smart contracts. Implement the Truffle Framework using **Solidity**, which is a high-level, object-oriented language that is focused on the implementation of smart contracts. To use the Truffle Framework, in the ethreact/ directory, run the installation following command:

```
$ npm install -g truffle@5.0.2
```

To set up the React app to use Truffle, in the ethreact/ directory, run the following command, which may take several minutes.

```
$ truffle init
```

When the initialization is completed, the contracts/ directory and Migrations.sol file will be generated. In addition, the truffle-config.js file will be generated, as highlighted in Figure 11.47.



Figure 11.47 Here is the truffle-config.js file. (rendered with Truffle by Truffle Security Co., under MIT license; attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

To create the smart contract, in the ethreact/contracts/ directory, create the file TodoList.sol and add the following code.

```
pragma solidity ^0.5.0;
contract TodoList {
  uint public taskCount = 0;

  struct Task {
    uint id;
    string content;
    bool completed;
  }

  mapping(uint => Task) public tasks;

  event TaskCreated(
    uint id,
    string content,
    bool completed
  );

  event TaskCompleted(
    uint id,
    bool completed
```

```
  );

  constructor () public {
    createTask("Check out dappuniversity.com");
  }

  function createTask(string memory _content) public {
    taskCount ++;
    tasks[taskCount] = Task(taskCount, _content, false);
    emit TaskCreated(taskCount, _content, false);
  }
  function toggleCompleted(uint _id) public {
    Task memory _task = tasks[_id];
    _task.completed = !_task.completed;
    tasks[_id] = _task;
    emit TaskCompleted(_id, _task.completed);
  }

}
```

Next, compile the smart contract. In the ethreact/ directory, run the following command:

```
$ truffle compile
```

This command should provide a status confirming that the smart contract has been successfully compiled as seen in the following code:

```
> Compiled successfully using:
  - solc: 0.5.16+commit.9c3226ce.Emscription.clang
```

Figure 11.48 shows how this process will generate a few files, including two JSON files in the build/contracts/ directory and a migration file in the migrations/ directory.

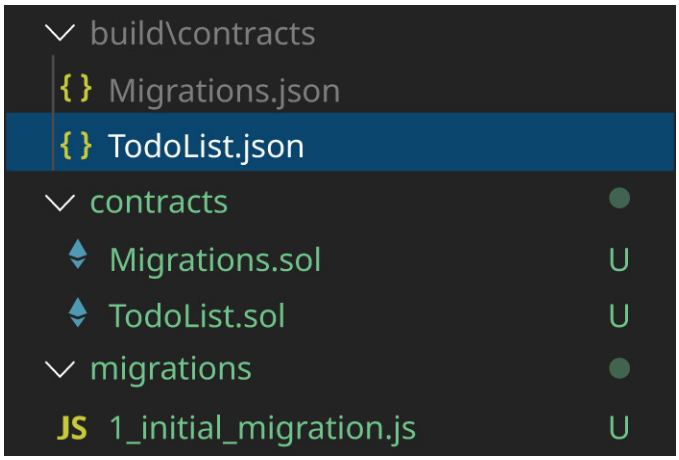


Figure 11.48 These are the JSON and migration files generated when the smart contract is compiled. (rendered using JSON; attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The TodoList.json file in the build/contracts/ directory is the smart contract Abstract Binary Interface (ABI) file. This file contains the following:

- compiled bytecode from the Solidity smart contract code that can run on the Ethereum Virtual Machine (EVM)
- a JSON representation of the smart contract.

Next, configure the React app to connect to the Ganache blockchain network. To do this, open the ethreact/truffle-config.js file and uncomment the two sections shown. Under “networks,” enable the connection host and port to the Ganache blockchain network. Ensure that the host and port are in sync with the following settings in Ganache.

```
development: {
  host: "127.0.0.1",     // Localhost (default: none)
  port: 7545,            // Standard Ethereum port (default: none)
  network_id: "*",       // Any network (default: none)
},
```

Next, under “compilers,” enable the solc optimizer.

```
// Configure your compilers
compilers: {
  solc: {
    // version: "0.5.1",    // Fetch exact version from solc-bin (default: truffle's
version)
    // docker: true,        // Use "0.5.1" you've installed locally with docker
(default: false)
    // settings: {          // See the solidity docs for advice about optimization and
evmVersion
     optimizer: {
       enabled: true,
       runs: 200
     },
    //  evmVersion: "byzantium"
    // }
  }
},
```

Next, create a migration script to deploy the smart contract to the Ganache blockchain network. In the ethreact/migrations/ directory, create the file 2_deploy_contracts.js and add the following code.

```
var TodoList = artifacts.require("./TodoList.sol");

module.exports = function(deployer) {
  deployer.deploy(TodoList);
};
```

The next step is to migrate the contract. In the ethreact/ directory, run this command.

```
$ truffle migrate
```

Figure 11.49 displays how the contract will be migrated and provides the transaction details. You should make a note of the contract address in the output because it will be added to the config.js file later.

```
2_deploy_contracts.js
=====================

   Deploying 'TodoList'
   --------------------
   > transaction hash:    0x11c7eff2b2697b933933b306e4799345f54faab03b37ccf1f0282489e3918ae7
- Blocks: 0            Seconds: 0
   > Blocks: 0            Seconds: 0
   > contract address:    0x9A93A9951c1cAcF0Cecc320Fa6aD4e05d55a2d02
   > block number:        3
   > block timestamp:     1643697826
   > account:             0x68387F6A38ACD2fe38246D9B8F78Ce64390C992C
   > balance:             99.98513256
   > gas used:            509091 (0x7c4a3)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.01018182 ETH


- Saving migration to chain.
   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.01018182 ETH
```

Figure 11.49 Here are the transaction details for migrating the contract. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

LINK TO LEARNING

Solidity is a high-level, object-oriented language used to implement smart contracts. Influenced by JavaScript, C++, and Python, Solidity is focused on the Ethereum Virtual Machine. Smart contracts implemented by Solidity are used for purposes like blind auctions, voting, and crowdfunding. Visit this page on Solidity (https://openstax.org/r/76Solidity) to learn more.

Creating the Front-End React Components

The TodoList component renders the user interface for the Todo app. In the ethreact/src/ directory, create the components/ directory. In the components/ directory, create the file TodoList.js and then create the TodoList component.

```
import React, { Component } from 'react'

class TodoList extends Component {

  render() {
    return (
      <div id="content">         <form onSubmit={(event) => {
          event.preventDefault()           this.props.createTask(this.task.value)
        }}>           <input             id="newTask"             ref={(input) => {
              this.task = input             }}             type="text"
            className="form-control"             placeholder="Add task..."
            required />           <input type="submit" hidden={true} />
        </form>         <ul id="taskList" className="list-unstyled">           {
this.props.tasks.map((task, key) => {             return(               <div
className="taskTemplate" className="checkbox" key={key}>                 <label>
                <input                   type="checkbox"
                name={task.id}                   defaultChecked={task.completed}
```

```
                  ref={(input) => {                    this.checkbox = input
                  }}                  onClick={(event) => {
                    this.props.toggleCompleted(this.checkbox.name) }}/>
                  <span className="content">{task.content}</span>
                </label>               </div>             )           })}
        </ul>         <ul id="completedTaskList" className="list-unstyled">
        </ul>       </div>
    );
  }
}
export default TodoList;
```

Update the ethreact/src/App.js to import and render the TodoList component.

```
render() {
  return (
    <div>       <nav className="navbar navbar-dark fixed-top bg-dark flex-md-nowrap
p-0 shadow">         <a className="navbar-brand col-sm-3 col-md-2 mr-0"
href="#">ToDo</a>         <ul className="navbar-nav px-3">           <li
className="nav-item text-nowrap d-none d-sm-none d-sm-block">             <small><a
className="nav-link" href="#"><span id="account"></span></a></small>           </li>
        </ul>       </nav>       <div className="container-fluid">         <div
className="row">           <main role="main" className="col-lg-12 d-flex justify-
content-center">             { this.state.loading               ? <div id="loader"
className="text-center"><p className="text-center">Loading...</p></div>
              : <TodoList                 tasks={this.state.tasks}
                createTask={this.createTask}
                toggleCompleted={this.toggleCompleted} />             }
          </main>         </div>       </div>     </div>
  );
}
```

Adding Web3 to the React App

The next step is to add Web3 and configure it to connect to Ganache. To do this, open ethreact/src/App.js, import the Web3 package, and add the code shown here. In this code, a web3 connection is instantiated providing the Ganache URL (http://localhost:8545). This enables the React app to connect to the blockchain. The blockchain account information is then accessed and added to the React app's state object. In the ethreact/src/ directory, create the file getWeb3.js and add the code shown. This code creates a Web3 object via different connection approaches depending on the approach used (e.g., type of DApp browser, using localhost).

```
import Web3 from "web3";

const getWeb3 = () =>
  new Promise((resolve, reject) => {
    // Wait for loading completion to avoid race conditions with web3 injection
timing.
    window.addEventListener("load", async () => {
      // Modern dapp browsers...
      if (window.ethereum) {
```

```
        const web3 = new Web3(window.ethereum);
        try {
          // Request account access if needed
          await window.ethereum.enable();
          // Accounts now exposed
          resolve(web3);
        } catch (error) {
          reject(error);
        }
      }
      // Legacy dapp browsers...
      else if (window.web3) {
        // Use Mist/MetaMask's provider.
        //const web3 = window.web3; // ORIG
        const web3 = window.web3.currentProvider.enable()
        console.log("Injected web3 detected.");
        resolve(web3);
      }
      // Fallback to localhost; use dev console port by default...
      else {
        const provider = new Web3.providers.HttpProvider(
          "http://127.0.0.1:8545"
        );
        const web3 = new Web3(provider);
        console.log("No web3 instance injected, using Local web3.");
        resolve(web3);
      }
    });
  });

export default getWeb3;
```

Configuring the React App to Communicate with the Smart Contract

To make the tasks from the smart contract available to the React app, create a config file, along with the contract address and ABI. To do this, in the ethreact/src/ directory, create the file config.js. Then, create two variables: TODO_LIST_ADDRESS and TODO_LIST_ABI. Grab the contract address that was provided when the smart contract was deployed to the blockchain and assign it to TODO_LIST_ADDRESS. Open the ethreact/build/contracts/TodoList.json file and copy the ABI portion only and assign it to TODO_LIST_ABI.

```
export const TODO_LIST_ADDRESS = '0x9A93A9951c1cAcF0Cecc320Fa6aD4e05d55a2d02'

export const TODO_LIST_ABI = [
  {
    "inputs": [],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "constructor"
  },
  {
    "anonymous": false,
```

```
    "inputs": [
      {
        "indexed": false,
        "internalType": "uint256",
        "name": "id",
        "type": "uint256"
      },
      {
        "indexed": false,
        "internalType": "bool",
        "name": "completed",
        "type": "bool"
      }
    ],
    "name": "TaskCompleted",
    "type": "event"
  },
  {
    "anonymous": false,
    "inputs": [
      {
        "indexed": false,
        "internalType": "uint256",
        "name": "id",
        "type": "uint256"
      },
      {
        "indexed": false,
        "internalType": "string",
        "name": "content",
        "type": "string"
      },
      {
        "indexed": false,
        "internalType": "bool",
        "name": "completed",
        "type": "bool"
      }
    ],
    "name": "TaskCreated",
    "type": "event"
  },
  {
    "constant": true,
    "inputs": [],
    "name": "taskCount",
    "outputs": [
      {
        "internalType": "uint256",
        "name": "",
```

```
      "type": "uint256"
    }
  ],
  "payable": false,
  "stateMutability": "view",
  "type": "function"
},
{
  "constant": true,
  "inputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "name": "tasks",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "id",
      "type": "uint256"
    },
    {
      "internalType": "string",
      "name": "content",
      "type": "string"
    },
    {
      "internalType": "bool",
      "name": "completed",
      "type": "bool"
    }
  ],
  "payable": false,
  "stateMutability": "view",
  "type": "function"
},
{
  "constant": false,
  "inputs": [
    {
      "internalType": "string",
      "name": "_content",
      "type": "string"
    }
  ],
  "name": "createTask",
  "outputs": [],
  "payable": false,
```

```
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "constant": false,
    "inputs": [
      {
        "internalType": "uint256",
        "name": "_id",
        "type": "uint256"
      }
    ],
    "name": "toggleCompleted",
    "outputs": [],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "function"
  }
]
```

Next, update `ethreact/src/App.js`. Import getWeb3.js and add the following code to the `componentDidMount()` life cycle method. This will connect to the blockchain network and load the contract before the React components renders in the browser.

```
componentDidMount = async () => {
  try {
    // Get network provider and web3 instance.
    const web3 = await getWeb3();

    // Use web3 to get the user's accounts.
    const accounts = await web3.eth.getAccounts();

    // Get the contract instance.
    const networkId = await web3.eth.net.getId();
    let deployedNetwork = TodoListABI.networks[networkId];
    const todoListInstance = new web3.eth.Contract(
      TODO_LIST_ABI,
      deployedNetwork && deployedNetwork.address
    );

    // Set web3, accounts, and contract to the state, and then proceed with an
    // example of interacting with the contract's methods.
    this.setState({ web3, accounts, todoListContract: todoListInstance },
this.runExample);
  } catch (error) {
    // Catch any errors for any of the above operations.
    alert(
      'Failed to load web3, accounts, or contract. Check console for details.',
    );
    console.error(error);
```

```
  }
};
```

Launch the React app by running the following command:

```
$npm start
```

Launch a browser with the MetaMask plug-in installed and navigate to http://localhost:3000. Create a task.

Chapter Review

Key Terms

Android Studio official IDE for Android development
Asynchronous JavaScript and XML (AJAX) exchanges small amounts of data between a client and server
cascading style sheets (CSS) standard style sheet language used to alter the presentation style of the web data content found in HTML
decentralized Apps (DApps) applications that execute smart contracts and run over distributed ledger technology
Django project high-level directory used to contain the directories and files necessary to run a Django web application
Ethereum blockchain creates a secure peer-to-peer network through the use of smart contracts
Firebase app development platform and a collection of services for authenticating users, integrating ads, running A/B tests, and more
full node computer that maintains a copy of the blockchain and runs blockchain software
gas price cost of validating transactions and updating ledgers
GraphQL open-source query and manipulation language
hypertext markup language (HTML) standard markup language used to describe the structure and content of web data
JavaScript (JS) scripting language that adds interactivity to web content and server-side functionality
JavaScript Object Notation (JSON) file format that represents data as text-based attribute-value information
Jetpack Compose toolkit for building native user interfaces
loose coupling component in a software system that has a weak association with the other components
Metro bundler bundles the JavaScript code that is deployed on the mobile device or emulator when the React Native front-end application is successfully created
Moore's law states that the number of transistors on an integrated circuit doubles roughly every two years
Node JavaScript runtime environment that provides users with the tools to develop web applications, as well as servers, scripts, and command-line tools
non-fungible token (NFT) unique digital identifier on a blockchain
optimistic rollup protocol that increases transaction output by bundling multiple transactions into batches, which are processed off-chain
peer-to-peer (P2P) network one in which devices connect and can share data and processing without needing a centralized server
Postman API platform testing tool that can be used as a client
React JavaScript library popular to build user interfaces
React Native open-source JavaScript framework used to build native applications for mobile devices
Semantic Web system of autonomous agents, which are software programs that respond to events
serializer tool to control response outputs and convert complex data into content, such as JSON
sidechain secondary blockchain that aggregates blocks back to the main blockchain
Solidity high-level, object-oriented language focused on the implementation of smart contracts
stateful application software and system that maintains the state of an application over time
stateless application state is not maintained by the system and previous actions do not impact future ones
Truffle Framework popular suite of tools used to develop smart contracts
View presentation layer that handles the user interface
Web 1.0 phase of the Web where the user's interaction was limited primarily to reading and selecting web pages
web page document commonly written in HTML and viewed in a browser
web publishing also called *online publishing*; is for publishing content on the Web
web server software application that runs at a known URL and responds to HTTP requests

World Wide Web (the Web) started as a way to link content (primarily text and images) stored on different servers or machines

Xcode Apple's IDE that enables application development for Apple's platforms

zero-knowledge rollup (zk-rollup) protocol that bundles transactions into batches that are executed off the mainnet

Summary

11.1 Modern Web Applications Architectures

- The Web is defined by phases, each denoted by its principal usage—Web 1.0 for reading content, Web 2.0 regarding social interaction, and Web 3.0 as an open, decentralized, and trusted web.
- Traditional web architectures rendered web pages on the server, with each client request resulting in a new HTML page being sent from the server to the browser. The Model-View-Controller (MVC) pattern was often used for server-side development.
- Responsive web applications render effectively in a browser regardless of the user's device screen or orientation.
- Single-page applications (SPAs) load one web page in the browser, manipulating the UI and fetching data through APIs. This is often done with JavaScript making AJAX calls to REST APIs.
- Native mobile applications are designed to run on a specific device. They have the advantage of performance and access to data and functionality on the device but suffer from a lack of cross-platform support.
- Web 3.0 is the next phase of the Web. Technologies that will be needed to support it will likely include distributed ledger technologies and AI.
- Future web architectures will likely see a combination of Web 2.0 and 3.0 architectural models to support decentralization, openness, trust, cost, and performance effectively. This hybrid approach solves some problems while not fully addressing the goals of Web 3.0.

11.2 Sample Responsive WAD with Bootstrap and Django

- Bootstrap is an open-source, responsive web application framework, and Django is a Python-based web application development framework. Both frameworks are highly popular due to their ease of use.
- To build the Todo application, you must install Python, PIP, Django, Django REST Framework, Bootstrap, and jQuery.
- The steps to build the Todo application are install and set up a local programming environment for Python 3, download and install Python 3.9.4, and add Python and its Scripts subfolder to your path environment data.
- The first step to build a Django web application is to create a Django project.
- Once the Django project is successfully completed and set up, the next step is to create the Todo application and register it in the Django project.
- After registering and installing the Todo web application, the next step is to create the models for Category and TodoList.
- After the Category and TodoList models are created, the next step is to create the serializers, the View, routers, user interface, and templates.

11.3 Sample Responsive WAD with Bootstrap/React and Node

- React is a JavaScript library popular to build user interfaces.
- Node is a JavaScript runtime environment that provides developers with the tools to develop web applications, as well as servers, scripts, and command-line tools.
- When creating a Todo web application using React and Node, React serves as the front end, handling the user interface, as well as getting and setting data via HTTP requests using Axios. Node serves as the back end, using a REST API built with ExpressJS and the MongooseJS ODM to interact with a MongoDB database.
- To build the Todo application using Bootstrap, React, and Node, you need React v17.0.2, Bootstrap v4.5.0,

Node v14.17.5, ExpressJS v4.17.2, MongooseJS v6.1.9, and Axios v0.21.0.

- The steps to create a Todo web application using Bootstrap, React, and Node include:
  - create the Node back end
  - create the Node app
  - set up the Express web server
  - configure the MongoDB database and Mongoose
  - build the controller
  - set up the REST API
  - create the React front end
  - create the React app
  - install Bootstrap and other dependencies
  - create the React components
  - connect the React front end to the Node back end

11.4 Sample Responsive WAD with Bootstrap/React and Django

- The Todo web application can be updated using Bootstrap with React and Django.
- For this version of the Todo web application, React serves as the front end handling the user interface to get and set data via HTTP requests. Django serves as the back end.
- To build the Todo application using Bootstrap with React and Django, you need Python v3.9.4, PIP v21.3.1, Django v4.0.1, Django REST Framework v3.13.1, Bootstrap v4.5.0, Django-cors-headers v3.11.0, React v17.0.2, and Axios v0.21.0.
- The React application uses Axios to fetch data by making requests to a given endpoint.
- A proxy to the Django application helps tunnel API requests from the React application to http://localhost:8000, where the Django application will receive and handle the requests.

11.5 Sample Native WAD with React Native and Node or Django

- A Todo application for mobile devices can be developed using React Native and Node.
- React Native is an open-source JavaScript framework used to build user interfaces and native applications for mobile devices.
- To build a Todo application for mobile devices, React Native serves as the front end handling the user interface and getting and setting data via HTTP requests. Node serves as the back end that makes use of the API built using the Django REST Framework in 11.2 Sample Responsive WAD with Bootstrap and Django.
- Building the Todo mobile application using React Native and Node requires React Native v0.67, Node v14.17.5, ExpressJS v4.17.2, MongooseJS v6.1.9, and Axios v0.21.0.

11.6 Sample Ethereum Blockchain Web 2.0/Web 3.0 Application

- A simple Todo application can be created using React with Web 3.0 powered by Ethereum smart contracts on the blockchain.
- The Ethereum blockchain creates a secure peer-to-peer network through the use of smart contracts, which are secure digital agreements that enable users to transact directly with each other via the Web.
- Building the Todo Ethereum blockchain Web 3.0 application requires React v17.0.2, Bootstrap v4.5.0, Node v14.17.5, Web 3.js v1.2.2, Truffle v5.0.2, Solidity v0.8.11, and Ganache.
- The Truffle Framework, which is a suite of tools popular to develop smart contracts, can be used to develop the Ethereum smart contract.
- The Truffle Framework can be implemented using Solidity, which is a high-level, object-oriented language that is focused on the implementation of smart contracts.

Review Questions

1. What is MVC?
   a. the primary design pattern used for SPA applications

b. a software architecture pattern that separates a system's presentation, business logic, and data
c. a software pattern for loose coupling and high cohesion
d. stands for Multi-Vector Chain, a principal technology used in Web 3.0 blockchain applications

2. What is a web application framework?
a. software designed to aid in developing web applications
b. software that restricts the boundaries and edges of the network for the application
c. software that provides cybersecurity for web applications
d. software that increases the performance of a web application when networking traffic is heavy

3. What is a native application framework?
a. software designed to support development and execution targeted toward a specific platform (e.g., Android, iOS)
b. program for writing assembly language for a given device
c. software tools created by a company that also creates the targeted device
d. software designed for responsive Web 2.0 SPA applications

4. List examples of web and native application frameworks.

5. Why is it difficult to implement server-side rendering using MVC?

6. What is a responsive web application?
a. a web application that runs very quickly
b. the web application runs as an SPA
c. a web application that changes the look and feel based on the user's credentials
d. a web application that is effective regardless of a user's device constraints, such as screen size or orientation

7. What is jQuery?
a. a web application framework like Angular
b. the official name for JavaScript
c. an open-source JavaScript library used for browser-based functionality
d. a Java-based implementation for server-side rendering

8. What is the difference between the first and current generations of web frameworks?
a. The names were changed; however, they have no differences.
b. Current web frameworks require blockchain, while previous generations did not.
c. First-generation web frameworks only used HTML and CSS, while current versions use JavaScript.
d. Current generation web frameworks adhere to updated web standards and resolve issues with the initial implementations.

9. What are the differences between Web 2.0 and Web 3.0 applications?

10. What is Bootstrap?
a. a Python-based web application development framework
b. a tool to control response outputs and convert complex data into content, such as JSON
c. an open-source, responsive web application framework
d. a high-level directory used to contain the directories and files necessary to run a Django web application

11. Using Bootstrap and Django, what should you do to define the Todo model?
a. create serializers
b. use the default Django admin interface to perform CRUD operations on the database
c. enter category names and click the Post button
d. install Bootstrap

12. Why is the View function important?
- **a.** The View function enables users to create user interfaces.
- **b.** The View function creates a Django project, which is needed to contain directories and files.
- **c.** The View function is required to generate the todo/ directory.
- **d.** The View function is needed to interact with the database to both create and delete todo items.

13. When a Mongoose schema is defined for the todos model, what happens?
- **a.** A todos collection is created in the MongoDB database.
- **b.** The REST APIs are built in the MongoDB database.
- **c.** A todos collection is created in the Express web server.
- **d.** The REST APIs are created in the Express web server.

14. What does the controller do?
- **a.** The controller contains code that configures the MongoDB database and Mongoose.
- **b.** The controller contains code that builds the REST APIs.
- **c.** The controller contains code that creates the React components.
- **d.** The controller contains code that calls the Mongoose CRUD functions to interact with the MongoDB database.

15. What is Postman?
- **a.** the port that runs the Express web server
- **b.** an API platform testing tool
- **c.** the database connection URL to the Mongo DB database
- **d.** the connection that allows the CRUD functions to interact with the Express web server

16. What is the purpose of Axios?
- **a.** Axios runs the Express web server.
- **b.** Axios is used by the Node application to fetch data by making requests to a given endpoint.
- **c.** Axios runs the MongoDB database.
- **d.** Axios is used by the React application to fetch data by making requests to a given endpoint.

17. When using Bootstrap with React and Django to update the Todo web application, what will the React application use to fetch data by making requests to a given endpoint?
- **a.** Node
- **b.** Axios
- **c.** Postman
- **d.** Django API

18. What is needed to configure the Django application so it is allowed to accept in-browser requests that come from React?
- **a.** CORS
- **b.** Node
- **c.** Axios
- **d.** Proxy

19. When using Bootstrap with React and Django to build the Todo web application, how are API requests tunneled from the React application to http://localhost:8000, where they can be received and handled by the Django application?
- **a.** through an Express web server
- **b.** through Node
- **c.** through Postman
- **d.** through a proxy

20. To develop a Todo application for Apple platforms, what IDE should you use?

a. Android Studio
b. Django project
c. Xcode
d. Axios

21. What is a Metro bundler?
a. a JavaScript bundler that bundles code into a single JavaScript file
b. a Django bundler that bundles code into a single Django file
c. an Express web bundler that bundles code into a single Express web file
d. a Postman bundler that bundles code into a single Postman file

22. What must you run to connect the front-end app with the back-end app to create the mobile Todo application?
a. React Native app
b. Node app
c. Express web server
d. Mongo DB

23. What are smart contracts?
a. agreements between Bootstrap and the Ethereum blockchain to share information
b. secure digital agreements that enable applications to be used on mobile devices, as well as computers
c. agreements between React and Node to share information
d. secure digital agreements that enable users to transact directly with each other via the Web

24. When building a Todo Ethereum blockchain Web 3.0 application, what is the purpose of Ganache?
a. Ganache enables Truffle and Solidity to interact.
b. Ganache serves as the personal blockchain for development.
c. Ganache is a tool to develop smart contracts.
d. Ganache creates the front-end React components.

25. What does the Ethereum blockchain do?
a. creates a secure peer-to-peer network through the use of smart contracts
b. provides the high-level, object-oriented language needed to implement smart contracts
c. connects React to the MetaMask plug-in to create smart contracts
d. creates a migration script to deploy smart contracts to Ganache

Conceptual Questions

1. Why did it take a long time to create web frameworks that enforce a good architecture and a design able to evolve with web standards while sustaining operational stability and scalability? Provide examples to illustrate your answer.

2. How is it possible to leverage Web 3.0 applications to ensure data privacy and software openness? Provide examples to illustrate your answer.

3. Why are hybrid Web 2.0/3.0 applications needed?

4. The Ethereum blockchain relies on ether, a cryptocurrency that is the key currency on the network. For each transaction carried out on the Ethereum blockchain, a small fee in ether is applied. What is the need for such cost?

5. Explain the difference between a Django project and a Django web application.

6. Explain the difference between the Delete and Put buttons.

7. Explain the difference between React and Node.

8. Explain how React and Node are used to create a Todo web application.
9. When updating the Todo web application using Bootstrap with React and Django, explain how React and Django are used.
10. What are some of the benefits and drawbacks of native application development?

Practice Exercises

1. Explain how server-side rendering works.
2. How do SPA frameworks work as compared to traditional server-side rendering?
3. What is the difference between a P2P and a decentralized application architecture?
4. Using Bootstrap and Django to create a Todo application, to enable Django to recognize the Todo application, it must be registered in the Django project as an installed app. How is this done?
5. Using Bootstrap and Django, what is required to build the API needed for the Todo web application?
6. What is the purpose of the user interface and, using Bootstrap and Django, what is the first step to create a UI for Todo?
7. Follow the steps provided in this subsection to build a sample responsive web application with Bootstrap, React, and Node. Explain what each technology is used for in the app.
8. Install Postman and call your APIs to test them with Postman. How does Postman allow you to call your APIs?
9. Follow the steps provided in this subsection to build a sample responsive web application with Bootstrap, React, and Django. Identify each technology used in the responsive web application.
10. Follow the steps provided in this subsection to build a sample native application with React Native, Node, and Django. Explain the use of each technology.

Problem Set A

1. Classify the web application frameworks mentioned in this timeline (https://openstax.org/r/76WebFrameTime) according to the web application development chronology covered earlier in this subsection.
2. Perform some research on the Internet and identify the top three web application frameworks according to popularity and their relative pros and cons.
3. Explain how single-page applications (SPAs) are implemented.
4. After installing the Todo web application using Bootstrap and Django, you realize that you do not have a physical table for the categories that can be assigned to todo tasks. What steps should you follow to correct this?
5. Using Bootstrap and Django to create the Todo web application, how do you allow access to the admin interface?
6. Research some industry standard formats for the REST endpoint payloads. Provide at least two formats that REST supports.
7. In our sample web application, we implemented REST APIs in React. Research other languages that REST endpoints can be created in. Write a one- to two-sentence summary that explains supported languages.
8. Explain how React Native differs from React.
9. Explain the difference between Truffle Framework and Solidity.

Problem Set B

1. Implement a simple website using the Flask framework. How does Flask rank as compared to the latest web and web application frameworks?

   See the Flask documentation (https://openstax.org/r/76Flask) for more information.

2. Follow the Django tutorial (https://openstax.org/r/76DjangoTutor) to build a simple Django application.
3. Using Bootstrap and Django, if the categories in the Todo web application cannot be updated or deleted, what is the likely problem?
4. Using Bootstrap and Django, why is the View function vital for the Todo web application?
5. Another technology that is often paired with Postman is OpenAPI's Swagger UI. Research what OpenAPI's Swagger can do and how it can be paired with Postman.
6. Research how we can automate API testing with Postman. Explain why automated API testing is useful.
7. Explain how React Native and Node are used to build a Todo mobile application.
8. Develop a sample hello world Web 3.0 application and deploy it on Ethereum. Follow the steps described earlier in this subsection to optimize your application using the various technologies suggested in addition to the standard blockchain platform.

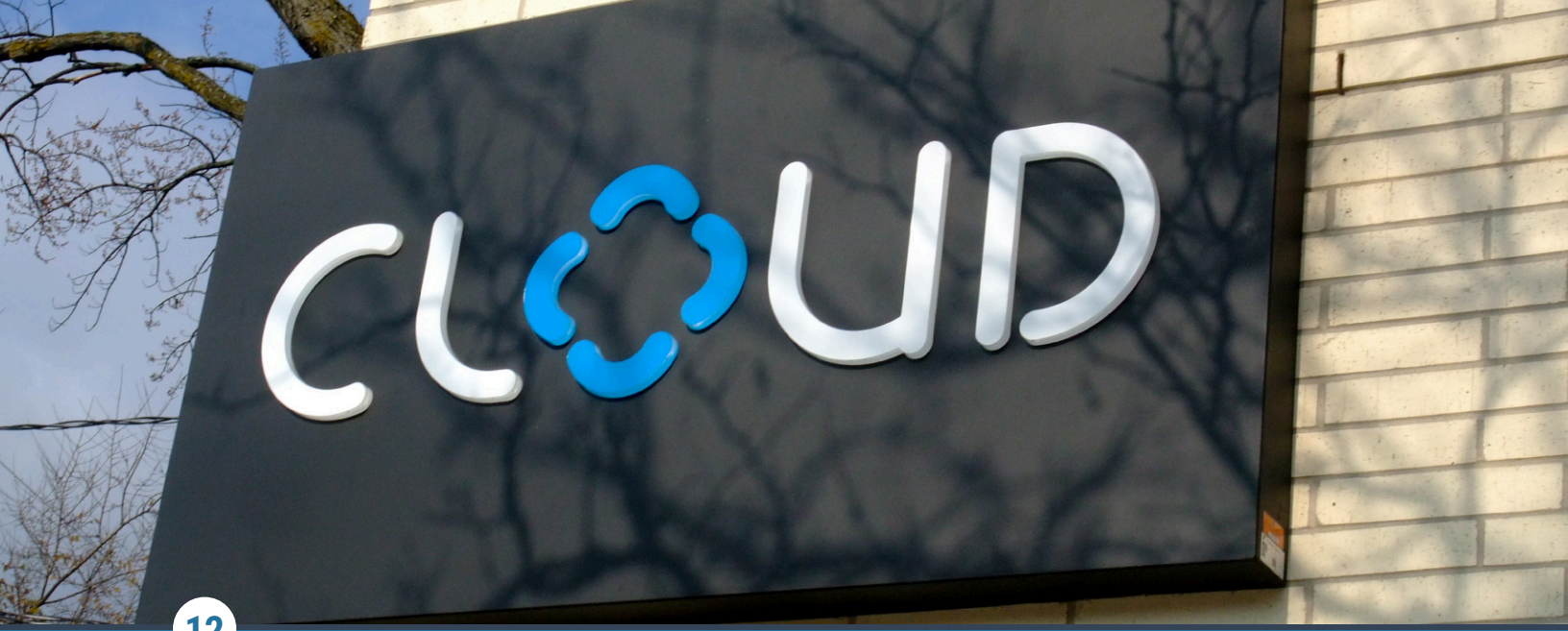
Thought Provokers

1. Consider our start-up company, which is looking to develop a mobile application for monitoring sleep disruptions (e.g., snoring, restless sleep) using a user's phone microphone. Users will be prompted to complete a mindfulness questionnaire on their phones each morning and night, allowing them to correlate sleep disruptions and mood. Users can also view reports on a website. The start-up company wants to use the data from users to point them toward support services, and they are looking to partner with health companies. Based on this, how would you recommend the solution be architected? Provide a sketch of the architecture. Create a list of the technologies you'd consider using, where they would be used, and why you chose that technology.
2. Your colleagues need an organized way to prioritize tasks for their team and would like to install a Todo web application. But they are concerned about the amount of work required. Based on your experience installing this Todo web application using Bootstrap and Django, what advice would you give them?
3. Consider our start-up company, which is looking to create a collection of web applications with a scalable back-end application that will feature multiple front-end applications. How can we create automated testing that will scale our back-end application?
4. Consider our start-up company and our goal of growth. Our new clients say that they will only use Apple devices. How can we accelerate development of their front-end web application?
5. Consider our start-up company and our goal of growth. Our client has indicated that their business does a lot of international business transactions. How could we utilize the Ethereum blockchain to save our customer in transaction fees?

Labs

1. Follow the steps provided in this subsection to build a sample responsive web application with Bootstrap and Django.
2. Download and install OpenAPI and the Swagger UI tool. Create a sample API and call it through Swagger UI and compare it to Postman.

3. Follow the steps provided in this subsection to build a sample Web 3.0 application using the Ethereum blockchain.



12

Cloud-Native Applications Development

Figure 12.1 Cloud-native development and optimization is critical to delivering software applications securely, more rapidly, and continuously. (credit: modification of "Cloud" by James Cridland/Flickr, CC BY 2.0)

Chapter Outline

12.1 Introduction to Cloud-Native Applications
12.2 Cloud-Based and Cloud-Native Applications Deployment Technologies
12.3 Example PaaS and FaaS Deployments of Cloud-Native Applications

Introduction

Organizations are facing more pressure today to retain customers who depend on updated or new business capabilities. This requires organizations to deliver software applications securely, more rapidly, and continuously. For organizations to adapt existing applications or build new applications using cloud-native application development, they must abide by a different set of architectural constraints to leverage cloud infrastructure in comparison to traditional on-premises infrastructure.

The focus should be on how to optimize these applications to leverage the capabilities that cloud platforms offer. There are four key principles of cloud-native development that help with designing applications for the cloud. First, applications should adopt a microservices architecture by breaking down the software into small components or services. Second, containerization must be applied to package microservices and isolate them for development and deployment. This not only speeds up development, but also makes it easier to quickly move isolated containers from one deployment environment to another. Containerization platforms such as Docker (an open-source platform used to deploy and run containerized applications) and container orchestration systems such as Kubernetes are instrumental in achieving this. Third, microservices are also more easily managed through continuous integration and continuous delivery. This makes it easier to automate the development and scale the deployment of highly resilient, manageable, and observable cloud-native applications. Fourth, DevOps facilitates the collaborative development of cloud-native applications using continuous delivery practices and results in shorter application development and deployment life cycles. DevOps allows development teams to adapt to changing user requirements more quickly while promoting business agility.

By adopting these four key principles, organizations can speed up the process of cloud-native development

and deployment. Cloud-native applications are critical to facilitate the creation of modern software solutions that interface with next-generation secure super society intelligent autonomous solutions (e.g., advanced robotics, autonomous cars, and drones, or other autonomous systems). Organizations in many industries rely on the use of cloud-native applications to conduct business daily.

12.1 Introduction to Cloud-Native Applications

Learning Objectives

By the end of this section, you will be able to:

- Analyze the differences between monolithic and microservices architectures
- Understand the architecture of cloud-native applications
- Learn the features of cloud-native applications
- Relate to the benefits of cloud-native applications
- Understand best practices and tools used to develop cloud-native applications
- Apply the tools used in cloud-native application development
- Envision the road ahead for cloud-native applications

Cloud computing makes it possible for organizations to offer solutions in the cloud and use cloud services to streamline development and deployment. As organizations look to the cloud to solve today's business problems, they may opt to create, deploy, and manage cloud-based or cloud-native applications. As noted earlier, cloud-native development relies on four key principles that help with designing applications for the cloud, as illustrated in Figure 12.2.

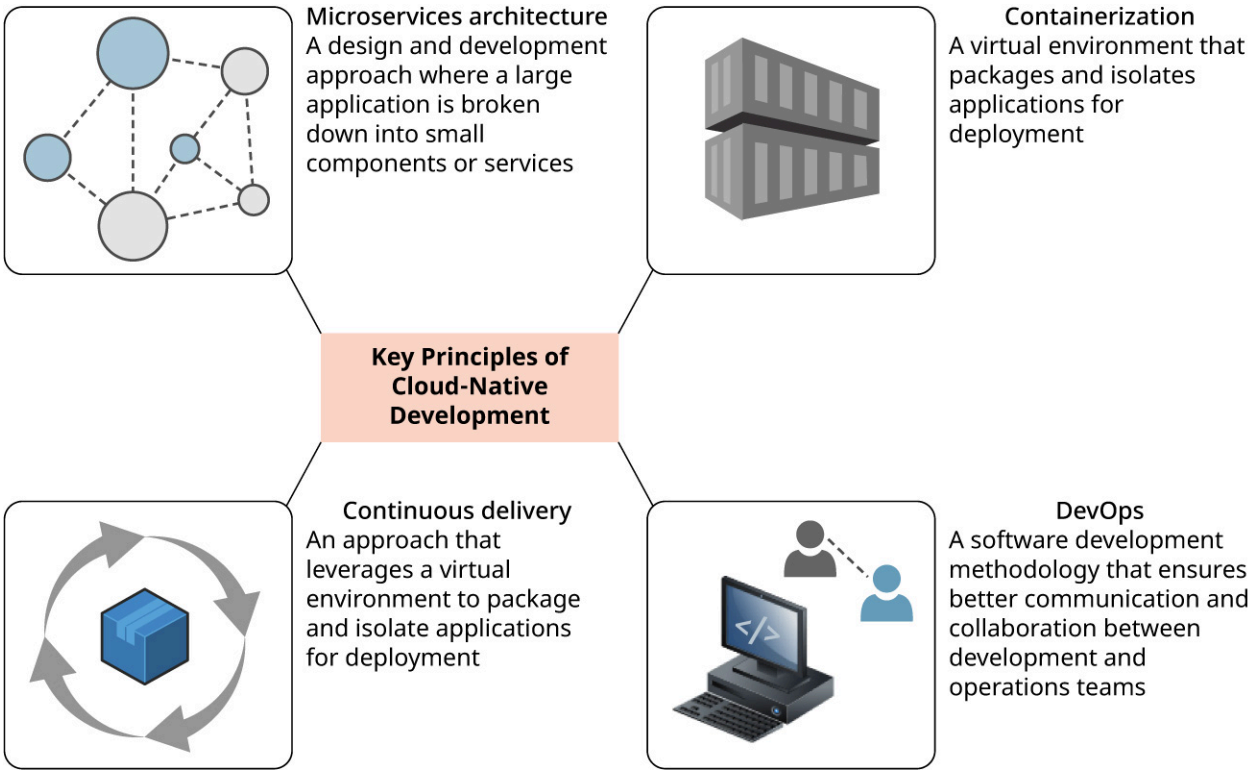


Figure 12.2 The four key principles of cloud-native development are microservices, containerization, continuous delivery, and DevOps. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

In a nutshell, a **cloud-native application** uses a microservices architecture that takes full advantage of the benefits of the cloud such as scalability and on-demand services. Cloud-based applications are adaptations of legacy applications, or monoliths, migrated to a cloud provider. They are designed to leverage cloud platforms to facilitate the interoperation of local and cloud components. Although cloud-based applications do leverage

cloud platforms to some degree, they are generally deployed as a single monolithic component, which makes them inflexible as full-stack upgrades are likely necessary and require downtime.

Different from cloud-based applications, cloud-native applications leverage microservices architectures that structure applications as coordinated collections of small, independent services that are self-contained and are deployed in software containers managed by a container orchestrator.

Both cloud-based and cloud-native applications can run on public, private, or hybrid cloud infrastructures. A **hybrid cloud** combines private, public clouds, and bare metal or virtual environments. What distinguishes cloud-native applications from cloud-based applications is the fact that they take full advantage of inherent characteristics of the cloud, such as resource pooling, on-demand self-managed services, automation, scalability, and rapid elasticity, to name a few.

Monolithic vs. Microservices Architecture

A **monolithic architecture** consists of software units that are united through a single codebase. In a monolithic architecture, all the different components of the application, including the user interface, business logic, and data access layer are tightly integrated and deployed as a single software unit. Mainframe applications or legacy e-commerce websites were based on monolithic architectures.

A microservice is a self-contained software unit that typically implements a single business capability. Because microservices are independent and loosely coupled, changes to an individual microservice can be deployed separately and do not require updating the entire application. This in turn streamlines the application development and minimizes downtime.

Monolithic architectures typically consist of a user interface, business logic, and data access layers that are tightly coupled and generally dependent on one another to operate properly. An application that conforms to a monolithic architecture (also called a **monolith**) implements all the business capabilities together, and data is shared across all these capabilities. Monoliths are typically easier to develop and deploy, but they may become difficult to manage and scale over time. The basic difference between a traditional monolithic architecture and a microservices architecture is illustrated in Figure 12.3.

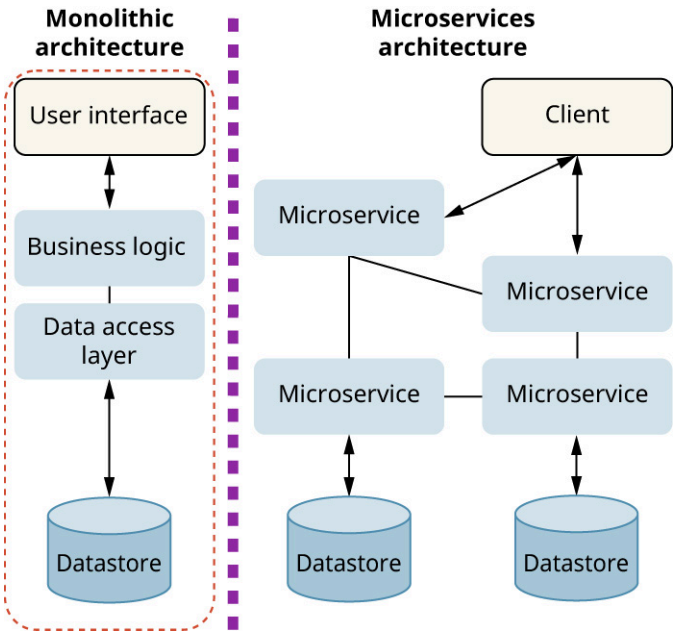


Figure 12.3 Monolithic architectures consist of highly dependent components and do not scale well compared to microservices architectures with self-contained components that are more easily scalable. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

As an example of a monolith, consider an application that provides a catalog of products that can be ordered

online. An application designed using a monolith architecture is shown in Figure 12.4. The application includes a user interface. The business logic layer includes various components such as an inventory system for the product catalog, a basket service to order products, a payment system to purchase the products, and a reporting system that generates reports on sales and customer interests. A component is a software unit that encapsulates a set of related functions and data through a well-defined interface and specified dependencies. A single database is typically used as part of the data access layer for the entire application.

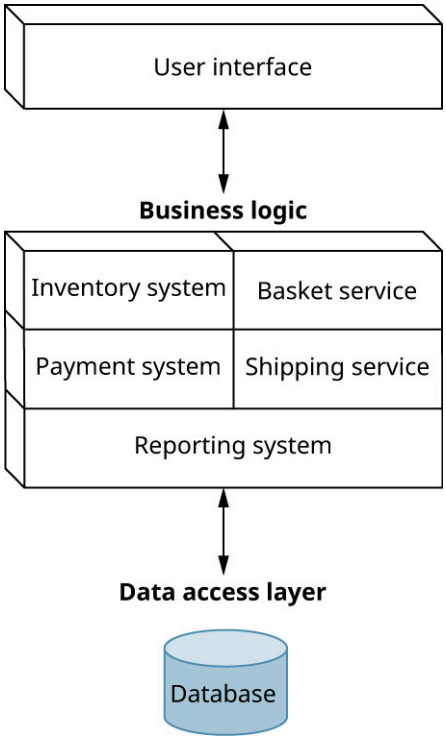


Figure 12.4 An application is designed using a monolithic architecture. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

These are highly dependent software components that use shared libraries. Any changes to one component require understanding what other components need from these shared libraries. Such changes can cause dependencies between components to break over time. The various components are also programming language and framework dependent. New components need to be implemented in the same programming language or use the same framework as the existing components.

Other challenges with monoliths are managing growth and ensuring scalability. Over time, as new functionality is added to meet the needs of end users, the monolith grows and becomes more difficult to manage. Deploying the monolith then becomes challenging as it requires more effort to stabilize it and get it to run smoothly. Scaling a monolith also becomes more difficult over time. For example, during a holiday season with a rapid increase in sales, the payment system component may require additional capabilities to support a sudden increase in transactions. To add the necessary capabilities to just the payment system component, the entire monolith application needs to be duplicated. The problem with this is that deploying a second instance of the entire application and making it available to end users may take a lot of time. Before this gets done, the holiday season may have ended already and the version of the application that was available was only able to support a limited number of transactions. The second instance of the application is useless at this point despite the effort it took to deploy it. A scaled version of the monolith application is shown in Figure 12.5.

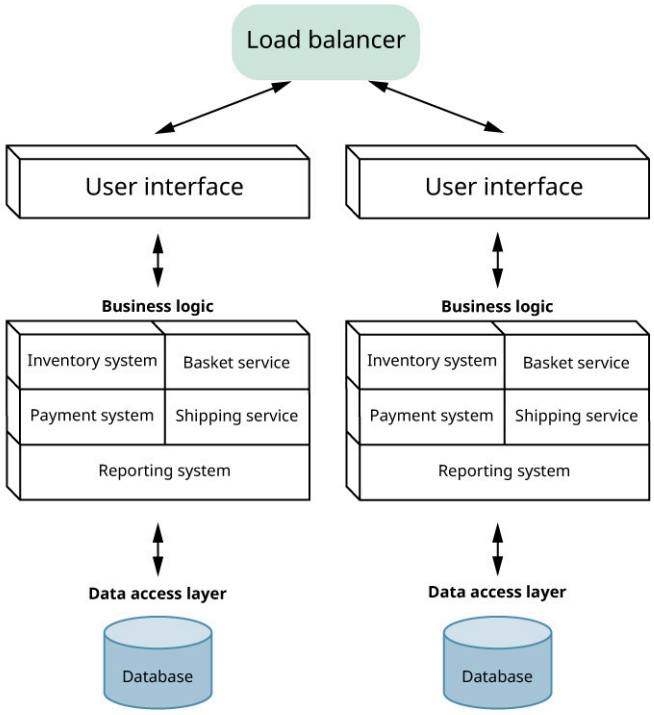


Figure 12.5 An application that was designed to use a monolithic architecture is scaled to handle an increase in use. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

In contrast, microservices architectures consist of small loosely coupled services that run independently and connect via a network. Applications that conform to a microservices architecture include a collection of modular components that run independently and provide a well-defined API to facilitate communication with other services. Instead of using a shared database across all business capabilities, each microservice that implements a single business capability contains its own datastore, the type of which depends on the type of data needed.

If the sample monolith application discussed earlier is implemented using a microservices architecture, each one of its components (i.e., the inventory system, basket service, payment system, and reporting system) becomes an individual microservice. These microservices each run in separate containers and communicate via APIs. An application designed using a microservices architecture is shown in Figure 12.6.

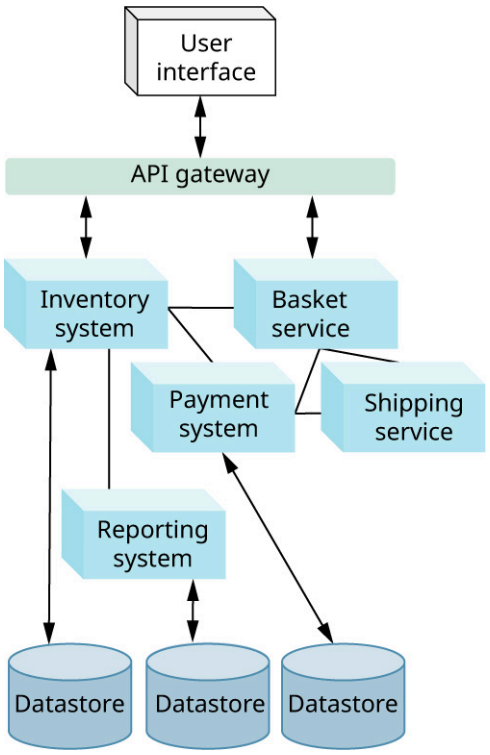


Figure 12.6 An application is designed using a microservices architecture. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Components in a microservices architecture are independent of each other. Individual microservices can be changed without affecting the other microservices. If a microservice fails, the rest of the application is not affected.

Microservices are also programming language and framework independent. New microservices can use different programming languages or frameworks. For example, a team responsible for the inventory system can use a programming language or framework that is different from the ones used to implement the payment system. It is also easier to grow or scale a microservices-based architecture.

Additional functionality can be added to individual microservices independently from the other microservices. This allows teams to iterate faster to bring value to end users. Microservices can also scale independently, as shown in Figure 12.7. During a holiday season with a rapid increase in sales, additional capabilities can be added to the payment system microservice quickly without affecting the other microservices. These additional capabilities can be scaled down as needed when sales decline after the holiday season ends.

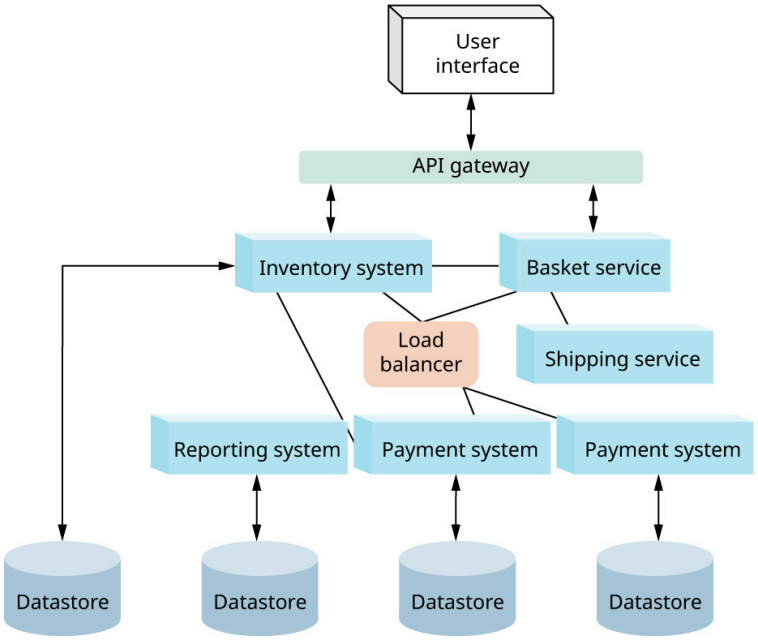


Figure 12.7 An application that was designed to use a microservices architecture is scaled to handle an increase in use. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

In summary, a monolith bundles all the capabilities together in a highly dependent way and requires them to use the same programming language and framework. Monoliths are more difficult to grow and scale because they require that the entire application be deployed as a second instance. In contrast, a microservices-centric application places individual capabilities in separate microservices that are deployed via containers and communicate via APIs. Individual capabilities are programming language and framework independent. They are easier to grow and scale independently.

LINK TO LEARNING

The article Microservices (https://openstax.org/r/76microservic) discusses the characteristics of a microservices architecture in contrast to a monolith architecture.

Components vs. Services

Building applications using components is a desirable approach. In the case of monoliths, this is typically achieved by using libraries as components and compiling or linking them into a single program or process that accesses them using “in-process” function calls. In contrast, “services” in a microservices-centric application are “out-of-process” components that communicate with each other using a web service request, or a remote procedure call. Figure 12.8 illustrates how components are accessed in a monolithic architecture as compared to a microservices architecture.

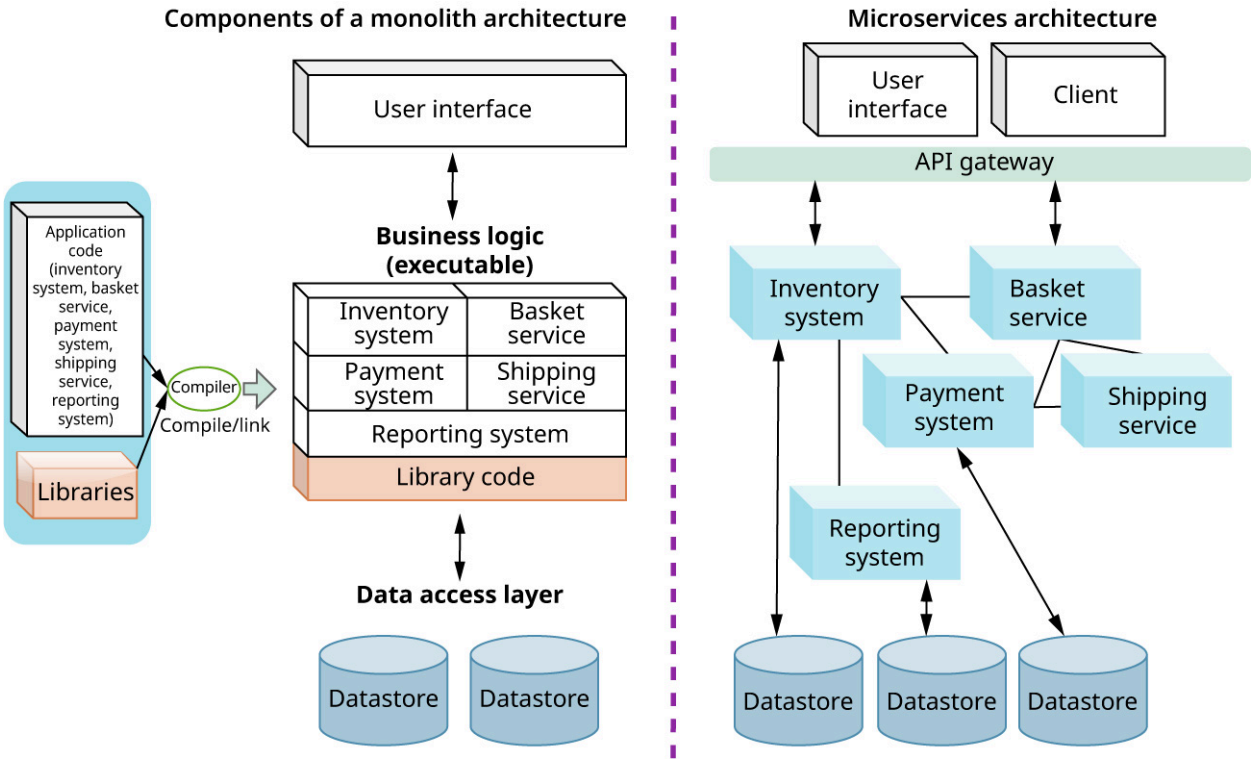


Figure 12.8 Components in a monolith architecture are accessed differently compared to components in a microservices architecture. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

As noted earlier, services used by microservices-centric applications are independently deployable so that changes to any service do not affect other services. This assumes services are well encapsulated and designed with minimally cohesive service boundaries. The use of web services calls and RPC makes this easier to achieve. A **web service** is a unit of software that is available as a resource over the Internet. A **remote procedure call (RPC)** is a request-response protocol used by a program to access services from other programs over a network. However, there are downsides to this approach. RPCs are more expensive than in-process calls, and thus remote APIs need to be coarser-grained (i.e., coarse-grained APIs expose more methods in their interfaces as opposed to fine-grained APIs).

One challenge to using services is changing the allocation of responsibilities between components. This can be challenging especially if the service consists of multiple processes deployed together such as an application process and a datastore process that are only used by that service. Another challenge is that it may be difficult to get the components' service boundaries right. In other words, the difficulty is to fit software into a component that can be used as a service. Services use APIs, thus any changes to the API must be coordinated with other components. API changes must be backward compatible so components that do not use the latest updated version of the API can still function. Finally, testing is also more complicated.

SOA vs. Microservices

The microservices architecture is not the first architectural style to make use of self-contained components and implement individual business capabilities. The service-oriented architectural style, a well-established style of software design, commands the same approach. A **service-oriented architecture (SOA)** requires that conforming applications be structured as discrete, reusable, and interoperable services that communicate through an enterprise service bus (ESB) via message protocols such as SOAP,[1] ActiveMQ,[2] or Apache Thrift.[3]

An **enterprise service bus (ESB)** implements a bus-like communication system between interacting service

1 For information on the SOAP Messaging Protocol, see https://www.w3.org/TR/2007/REC-soap12-part1-20070427/
2 For information on ActiveMQ, see https://activemq.apache.org/
3 For information on Apache Thrift, see https://thrift.apache.org/

providers and service consumers. The services, ESB, and messaging protocols collectively make up an SOA application. In Figure 12.9, the architecture shown on the left illustrates a typical SOA compared to the typical microservices architecture illustrated on the right. As is the case for services in a microservices architecture, SOA services can be updated independently. However, the ESB represents a single point of failure for the entire system.

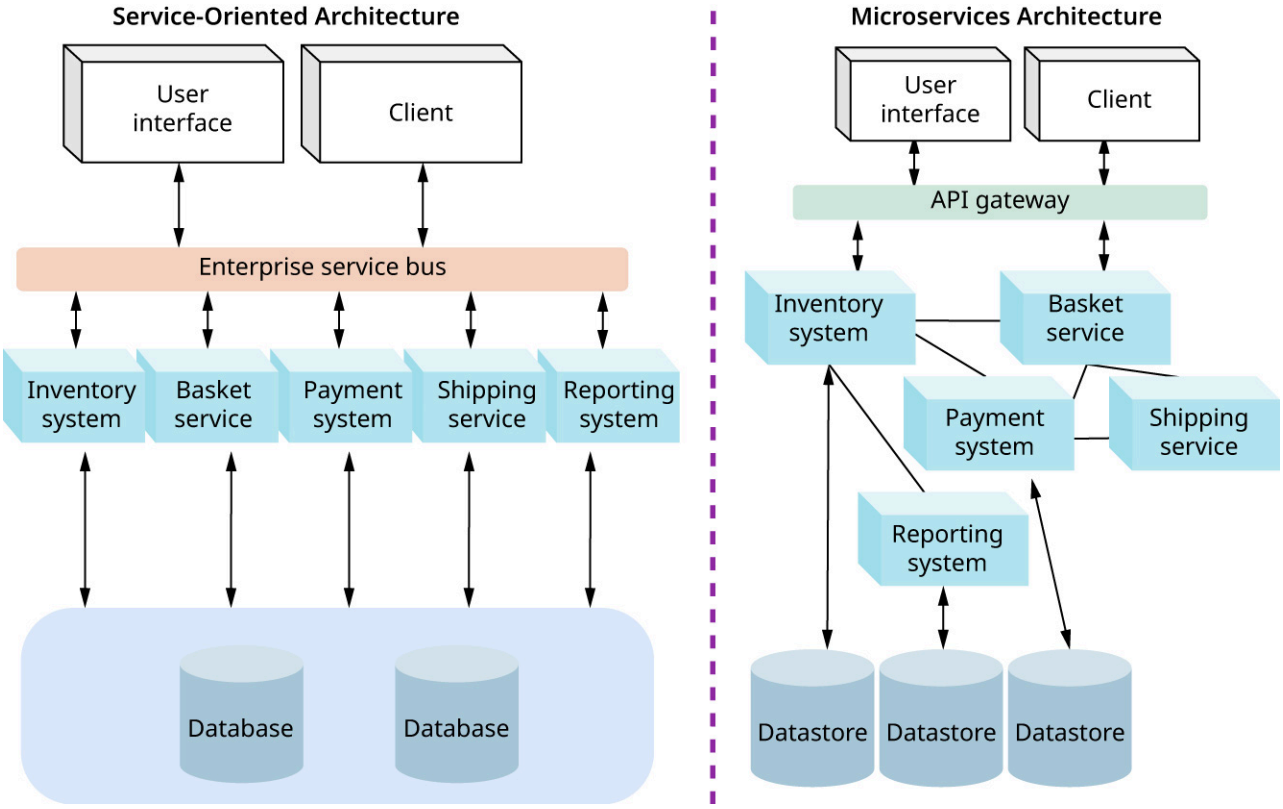


Figure 12.9 In an SOA, services communicate via an ESB compared to a microservices architecture where services typically communicate via REST protocols. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Different from services in an SOA, microservices usually communicate with each other statelessly using lightweight REST protocols implemented via HTTP. Furthermore, microservices can be implemented using different programming languages and frameworks because they can communicate through programming language-agnostic APIs. Advancements in containerization technologies make it possible to deploy, upgrade, scale, and restart microservices independently, better controlling their life cycles. In case a microservice fails, the rest of the application is unaffected. Thus, applications using microservices can be more fault tolerant because they do not rely on a single ESB as is the case for SOA applications.

CONCEPTS IN PRACTICE

Microservices Architecture and Cloud-Native Applications

As discussed in Chapter 9 Software Engineering and Chapter 10 Enterprise and Solution Architectures Management, architectural styles, architectural patterns, and design patterns are typically used to enforce the quality attributes of software solutions. The microservices architectural style is used to break monolithic applications into smaller services based on business capabilities. It is partially based on the SOA style but does not rely on an enterprise service bus to enable communication between services.

One example of a microservices design pattern is the Backend for Frontend (BFF) Microservice Design Pattern.[4] This design pattern involves creating dedicated back-end services tailored to the specific needs of front-end clients. The goal of this pattern is to ensure an efficient and streamlined interaction between the front-end and back-end components. As an example of how to use this pattern, consider an e-commerce platform like the online products catalog sample described above. The e-commerce platform may serve both web and mobile client applications. Implementing the BFF pattern means creating dedicated back-end components that serve each client application. Because the user interactions between these two client applications are different, the APIs between these clients and the dedicated back-end services can be optimized to best suit those client types.

In addition to optimizing the API gateway, the microservices architecture utilized also efficiently allocates resources to each service that a cloud-native application uses, making the application flexible and adaptable to a cloud architecture. Containers are an architectural pattern that is used to deploy microservices on any platform that operates a container engine runtime. Combining microservices and containers with another architectural pattern referred to as container orchestration makes it possible to scale, manage, and automate the deployment of containerized applications based on microservices. This is an example of how a combination of architectural styles and related patterns can drastically improve the quality of enterprise applications. However, as noted earlier, there are challenges associated with the use of microservices and other architectural styles that also must be considered when designing an enterprise solution.

Microservices Challenges

Microservices enhance a development team's ability to leverage distributed development. Development time can be reduced as microservices can be developed concurrently. However, there are known challenges to shifting to a microservices architecture. In addition to complexity and efficiency being two major challenges of a microservices-based architecture, the following eight challenge categories have been identified.

Building Microservices

Some microservices may need to access other microservices to provide certain application functionality. This causes dependencies between microservices. Time must be spent identifying these dependencies. Added dependencies between microservices can result in completing one build that triggers several subsequent builds.

Another type of dependency is data sharing where one microservice may need access to data managed by another microservice. Some concerns with data sharing include scaling. As microservices are added, to provide data consistency and redundancy, a microservice that requires a schema change in a database must be shared with other microservices. These concerns should also be taken into consideration when addressing dependencies.

Microservices Connectivity

Microservices-based applications consist of several microservices that need to communicate with each other. Microservices run in containerized environments where the number of microservices instances and locations change dynamically. Service discovery can be used to locate each microservice instance. Without service discovery, a microservices-based application can be difficult to maintain.

Versioning of Microservices

A microservice might be updated to satisfy new requirements or address some design issues. Other microservices that depend on an older version of the updated microservice could fail. Dependencies between

4 https://aws.amazon.com/blogs/mobile/backends-for-frontends-pattern/

microservices when updating a microservice can lead to breaking backward compatibility. One way to address this is to build in conditional logic that either appends the version number or tracks versioning, which can become difficult to manage. Alternatively, multiple live versions can be provided for different clients, but they can also become difficult to manage.

Testing Microservices

Testing microservices can become more challenging, particularly with integration testing and end-to-end testing. A single transaction carried out in an application can span across multiple microservices. A failure in one microservice can lead to failures in other microservices. In addition, failures can occur in the microservice itself, in its container, or in the network interconnecting the microservices, making it more challenging to design integration tests. For these reasons, an integration test plan should factor in interdependencies between microservices.

LINK TO LEARNING

Various tools and/or frameworks can help automate testing in a microservices testing (https://openstax.org/r/76microtest) such as Selenium automation testing.

Logging Microservices

Microservices are distributed systems and therefore traditional logging is ineffective when determining which microservices failed. As more microservices are added, the complexity of logging microservices activities grows exponentially and becomes more difficult to manage. In particular, logging aggregation is needed to track errors that might span across several microservices, as shown in Figure 12.10.

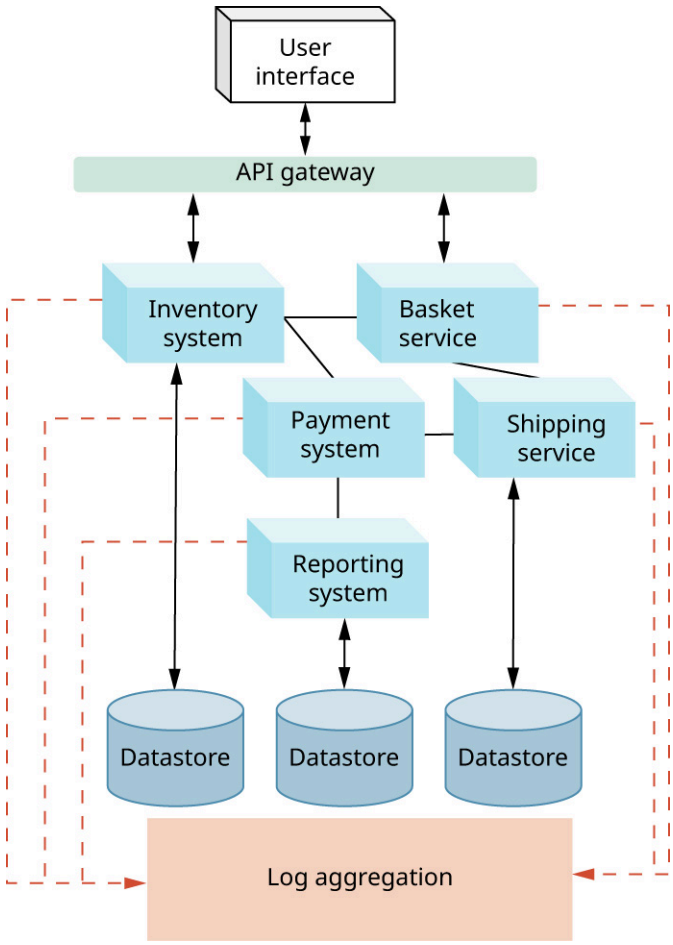


Figure 12.10 A microservices architecture is configured to use log aggregation and distributed tracing, which are typically used in a microservices-based distributed system. Log aggregation refers to storing various levels of warning and errors for a combined set of microservices. Distributed tracing helps interrelate information logged by multiple services so it can be followed as a single thread. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Debugging Microservices

Remote debugging through a local integrated development environment will not work across several microservices. An IDE provides developers access to resources (e.g., source code editor, build automation tools, debuggers) to design, develop, debug, and test programs. Debugging a microservices-based application is challenging because errors are not propagated in a useful manner. In addition, logging formats can also differ across microservices. Debugging requires tracing through various error messages and status codes across the network, which is inefficient and makes it difficult to acquire the required information to debug the errors. Currently, there is no easy solution to debugging such applications effectively; however, tools (e.g., Helios,[5] Rookout,[6] Lightrun,[7]) do exist today to address this problem.

Deployment of Microservices

A monolithic application's deployment may seem less complex because it is limited to a single deployment unit. In contrast, a microservices-based application is more complex to deploy because the interconnections between the microservices are more visible. In addition, there are more deployable units and the dependencies between them require more complex configurations. This increase in the number of deployable units imposes an order in which microservices need to be deployed. It also requires more investment in automation to practically manage them and help ensure that the whole microservices-based application is

5 For information about Helios, see https://snyk.io/blog/welcoming-helios-to-snyk/
6 For information about Rookout, see https://www.rookout.com/
7 For information about Lightrun, see https://lightrun.com/

resilient and limits the risks of failovers.

Monitoring Microservices

Because of the interdependencies between microservices, any downtime of a microservice resulting from updates or failures can cause cascading failures to other microservices. Monitoring a microservices-based application is challenging because it requires knowledge of the performance and availability of API calls for all the microservices in the application. In addition to monitoring the containers hosting the microservices, it is crucial to have a centralized view of the entire microservices-based application. This holistic view helps pinpoint any potential issues and ensures effective problem identification.

Cloud-Native Application Architecture

A cloud-native application takes advantage of cloud computing frameworks to implement a collection of loosely coupled microservices in the cloud. The cloud-native application architecture makes it possible to efficiently allocate resources to individual microservices that the cloud-native application uses. An example of a cloud-native application architecture is illustrated in Figure 12.11.

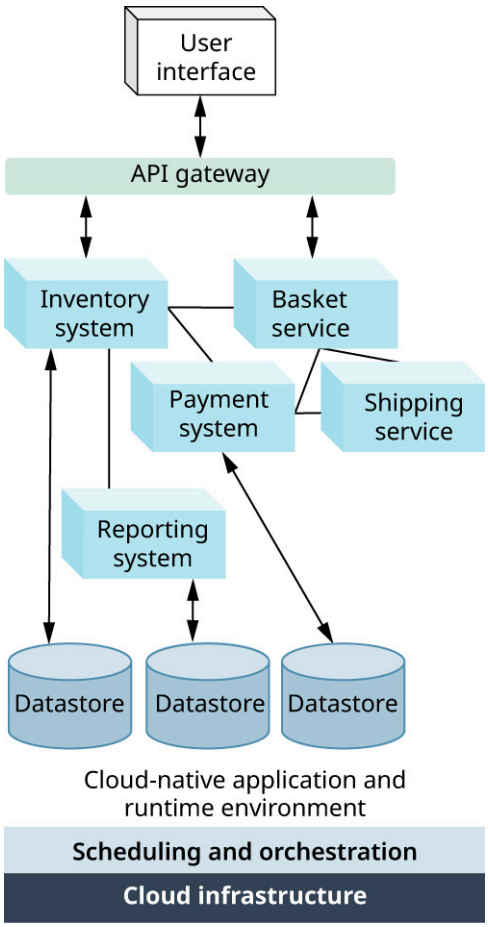


Figure 12.11 A cloud-native application architecture includes the cloud-native application and the cloud computing platform it leverages. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The cloud infrastructure, shown in Figure 12.11, is provided by a cloud computing platform. The cloud computing platform also provides a scheduling and orchestration service to manage the containers that the cloud-native application microservices are deployed in. Microservices used in a cloud-native application typically run in different locations within a cloud platform, which enables the application to scale out horizontally. Therefore, cloud-native applications must be designed with redundancy in mind. This allows the application to withstand equipment failures.

Finally, the cloud-native application and runtime environment is where the cloud-native application operates. A cloud-native architecture leverages a commoditized microservices architecture. In this architecture, business capabilities are implemented as cloud-managed microservices, which makes the cloud-native application more reliable and scalable.

Cloud-native applications should be designed to include standardized logging and support events that can be associated with a standard logging catalog. Scaling and managing multiple microservices require core services such as load balancing, service discovery, and routing, which are all managed by the scheduling and orchestration layer. In summary, designing cloud-native applications to leverage the services of a cloud computing framework makes it possible to add and support new business capabilities more quickly.

Sample Web-Native Architecture

Figure 12.12 illustrates a sample web-based, cloud-native "web-native" architecture. This cloud-native application consists of several microservices that are deployed and managed on the cloud. Each microservice is self-contained as well as programming language and framework independent. Microservices are containerized and managed by a container orchestrator. Each microservice contains its datastore that best suits its data needs. Some of these datastores are relational databases whereas others are NoSQL databases.

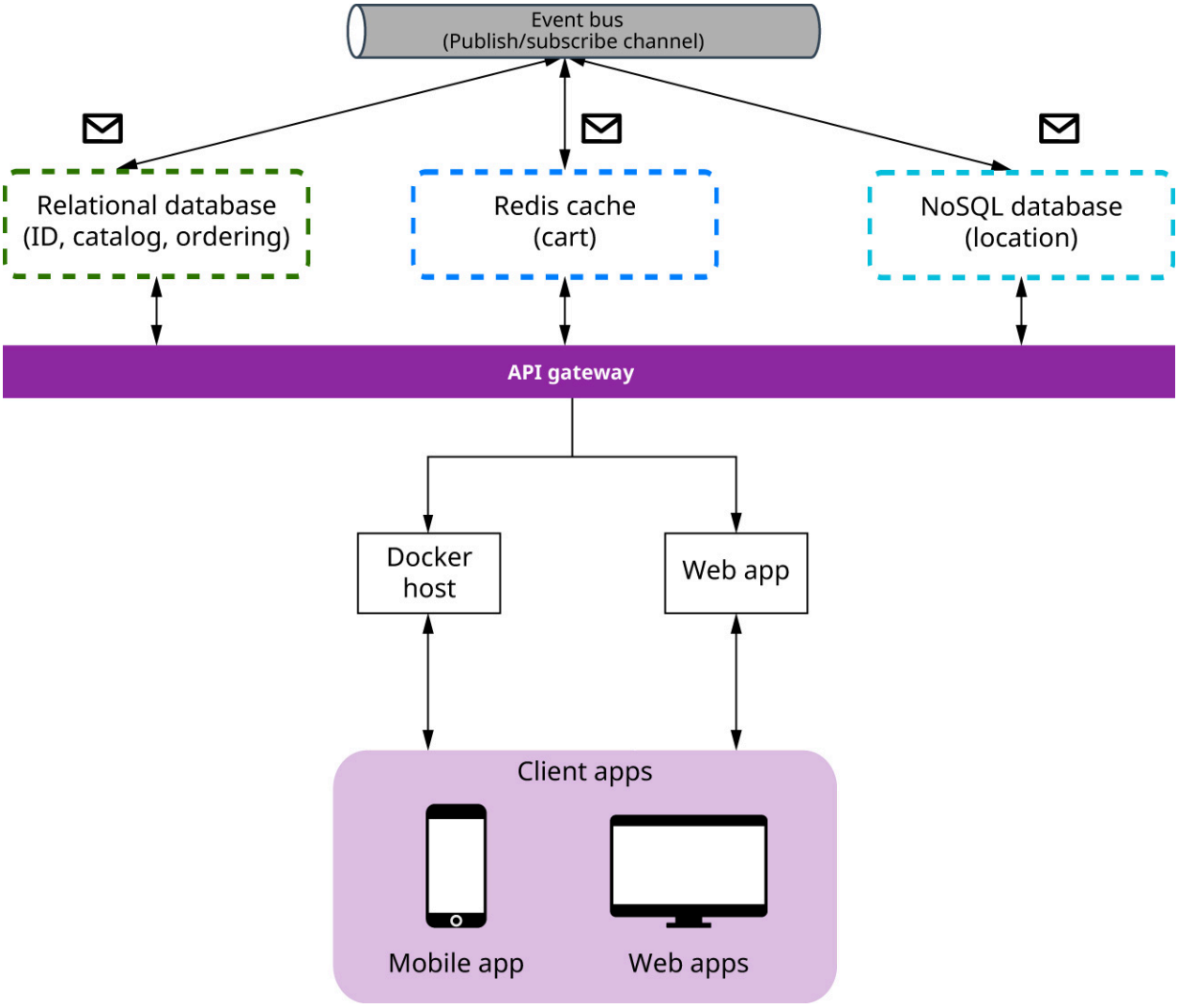


Figure 12.12 A web-native application leverages features of modern cloud platforms.[8] (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

8 Diagram is modernized by writer and based off https://devblogs.microsoft.com/premier-developer/wp-content/uploads/sites/31/2020/03/cloud-native-design.png

One of the microservices stores its state in a distributed Redis cache. An **API gateway** is used to route traffic between client apps and microservices. Most important, this cloud-native application takes full advantage of the scalability and resiliency features of modern cloud platforms.

INDUSTRY SPOTLIGHT

Cloud-Native Application in Industry

Cloud-native applications are broadly used in the industry today. In particular, Netflix, Uber, and WeChat expose cloud-native systems that consist of many independent services. Can you elaborate on the specific characteristics of these systems and explain why cloud-native is more suitable for the types of solutions provided by these companies? As an example, Netflix published the article "Completing the Netflix Cloud Migration"[9] discussing the benefits of this migration. A case study[10] was also published discussing Netflix's adoption of cloud-native computing environments for their services.

Features of Cloud-Native Applications

As mentioned previously, cloud-native applications, in contrast to cloud-based applications, leverage inherent characteristics of the cloud such as resource pooling, on-demand self-services, automation, scalability, and rapid elasticity. Below is a list of the key capabilities of cloud-native applications that make this possible.

Microservices-Based Applications

As noted earlier, microservices are based on a software architectural pattern where every functionality of a corresponding monolithic application is placed into its own microservice because it implements a separate business capability. These microservices are deployed and run in containers that communicate over application APIs, event streaming, and message brokers. Scheduling and orchestration tools manage these containers as they help coordinate and schedule the containers used to implement application services running on computing resources. Microservices development teams can rapidly add new capabilities whenever a business needs change.

Microservices can also be scaled, when a single capability faces too much load, independently of other microservices. This reduces the time and cost associated with having to scale another instance of the entire application. Typically, scaling is achieved by containerizing microservices and managing the containers using a container management framework (e.g., K8s) provided by a cloud platform. Another alternative is to leverage Function as a Service (e.g., AWS Lambdas or Google/Microsoft functions) as part of microservices and rely on cloud platform(s) to manage scalability.

Microservices are also distributed and loosely coupled, making them easier to iterate through. Because microservices are independent and just communicate over APIs, teams can commit code for microservices through a DevOps pipeline. Once tested, the microservices can be automatically deployed, making it easier for teams to iterate as fast as they need to bring value to end users. Additionally, these iterations are minimal, and if there is any instance where a microservice fails, the rest of the application is still functional, making updates less risky.

Best-Suited Languages and Frameworks for Microservices

The programming language and framework are chosen based on what is best suited for the functionality the microservice provides. Cloud-native applications are polyglots, meaning microservices are language and framework independent, making it possible to choose different technology stacks and frameworks for different microservices of a single cloud-native application.

9 https://about.netflix.com/en/news/completing-the-netflix-cloud-migration
10 https://www.cncf.io/case-studies/netflix/

Container-Based Framework

A container is a standardized unit of software that logically isolates an application, enabling it to run independently of physical resources. The packaging of a standardized unit of software that isolates an application, enabling it to run independently of physical resources is **containerization**. Containers provide consistent, complete, and portable environments for running applications anywhere, including all the system files and dependencies needed to run applications. Containerization facilitates the creation of cloud-native applications and maximizes the benefits of containers. Containers keep microservices from interfering with one another. They keep applications from consuming all the host's shared resources. They also enable multiple instances of the same microservice.

Containers boost DevOps efficiency and effectiveness through streamlined builds, testing, and deployments. They also provide consistent production environments that are isolated from other applications and processes. Because containers can run consistently anywhere, they facilitate hybrid and multicloud computing environments. Containers are lightweight because they are isolated from the operating system layer of a computer system. This makes them efficient and light on resources. Containers are portable because containers include all required dependencies and configurations, thus they, and the code contained within them, can be moved between different environments.

Figure 12.13 shows how containers in production can run on any computing resource that has a containerization platform. Containers are scalable due to their small size, making it easy to rapidly start up, scale, or shut down containers depending on use. Containers can be cost effective because they require fewer resources and are easier to scale.

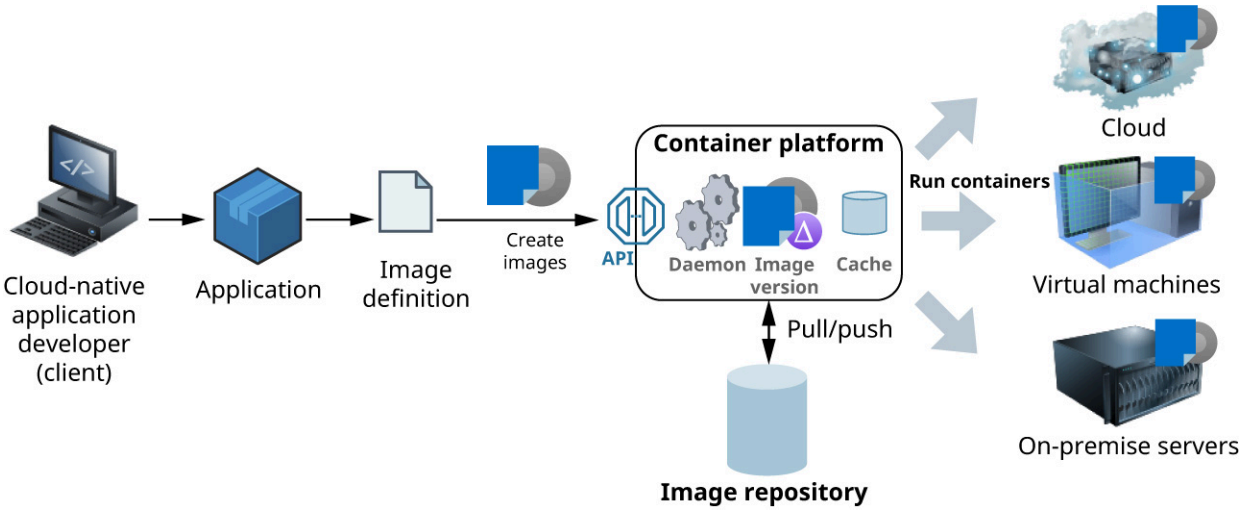


Figure 12.13 An image is built to deploy an application into containers that run on different computing resources. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Containers are instances of a container image. A **container image** represents a binary state that is built by "layering" an application and its dependencies on top of the required "layer" of binaries and libraries of the operating system, which establishes a parent-child relationship between image layers. Figure 12.14 shows the process for building an image. For example, at the root of the parent-child tree is a layer that provides a formatted file system. On top of this layer is a base image that might contain Ubuntu, Debian, CentOS, or some other operating system. This base image can then be a parent of some other image, such as an image that adds Python. Finally, another image on top of the Python image may contain the application that was implemented in Python, such as a Django web application. **Django** is an open-source web application framework, which is a software framework that facilitates the development, maintenance, and scaling of web applications. In effect, these layers of images are arranged in an image hierarchy where each layer creates a snapshot.

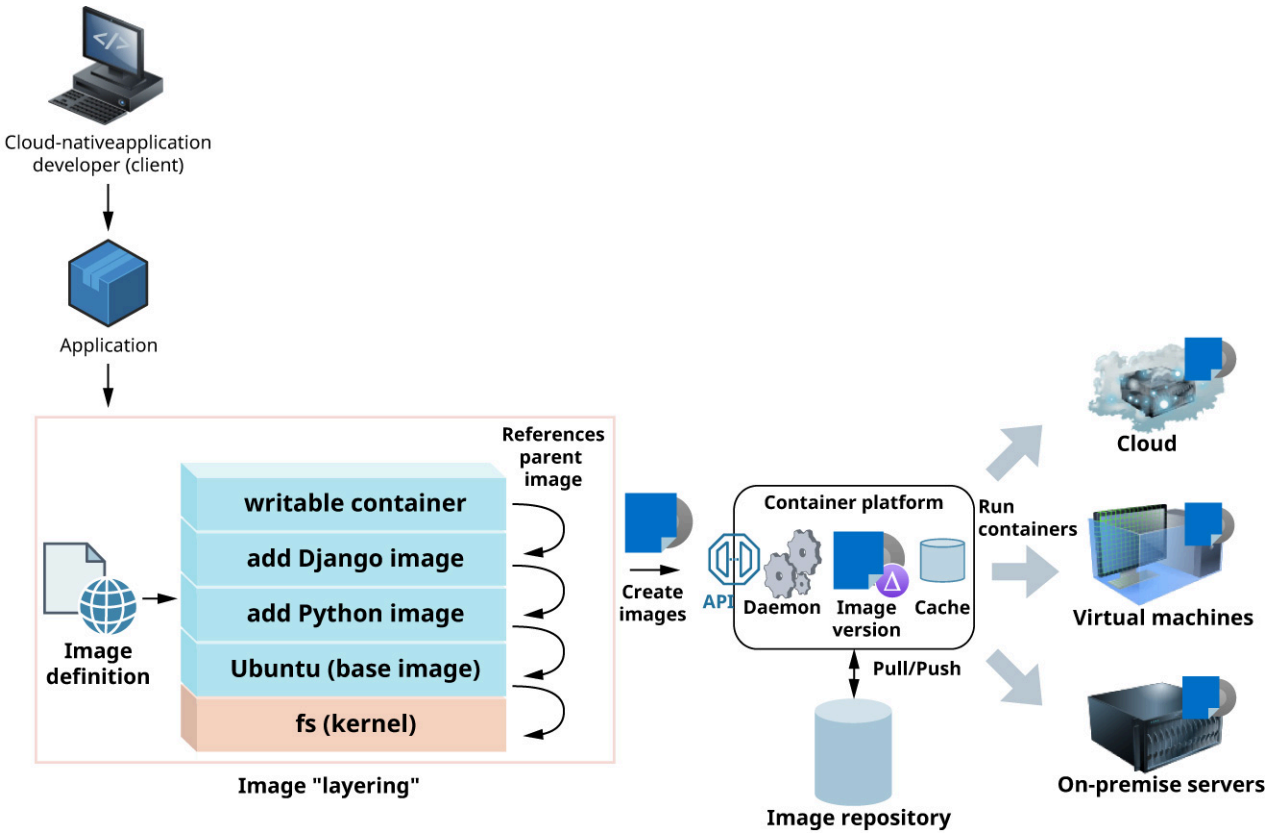


Figure 12.14 Images are built as layers of an application, its dependencies, and other required layers. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

An advantage of this hierarchical structure is that it makes it possible to share images at different levels of the hierarchy. If you want to run the Django web application, you pull the entire hierarchy of the built image. However, other layers of the hierarchy can be shared with other things (e.g., the Python layer), reducing the need to shift entire application stacks. The built container image is an independent and self-contained artifact that can be deployed in any environment that has a container runtime environment (e.g., Docker) installed. The Open Container Initiative is a community trying to govern and address uniformity and standardization of the container runtimes and images. This is an important initiative for containers to be able to "build once and run anywhere."

API-Based Framework

Microservices that rely on each other communicate using well-defined APIs. These APIs help configure the infrastructure layers to facilitate resource sharing across them. APIs connect microservices and containers while providing simplified maintenance and security. They enable microservices to communicate between otherwise loosely coupled services without sharing the same technology stacks, libraries, or frameworks. Cloud-native microservices use lightweight APIs that are based on protocols such as representational state transfer to expose their functionality. The architectural style for providing standards between resources so they can communicate with each other over the Web is called **representational state transfer (REST)**.

Dynamically Orchestrated Framework

As already mentioned, the scheduling and orchestration layer of a cloud computing framework manages the containers microservices are deployed in. As applications grow to span multiple containers deployed across multiple servers, operating them becomes more complicated. Container orchestration tools (e.g., Kubernetes) can help coordinate and schedule many containers. They can also help scale container instances to provide more computational power. For example, the orchestration tool, depicted as "Master" in Figure 12.15, deploys the microservices in containers (single instances, initially) of an application on a computing resource.

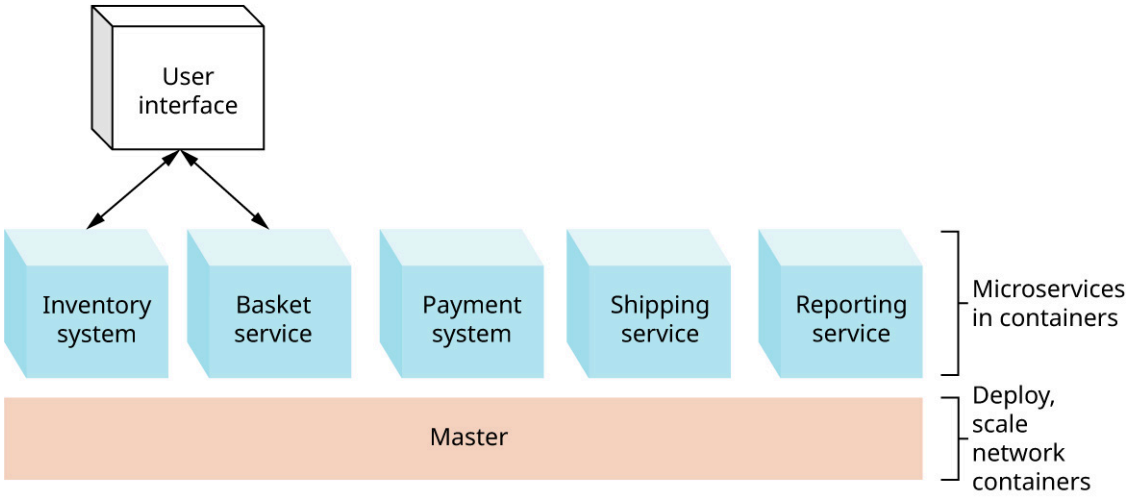


Figure 12.15 Container orchestration is used to manage microservices in a cloud-native application. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

As the microservices' consumption of the computing resources increases, additional resources are added, and the orchestration tool will start scheduling and scaling the microservices. An **orchestration platform** helps schedule the microservices and containers and optimizes the use of the computing resource. In contrast, a monolith application is not designed to leverage an orchestration platform and requires the use of load balancers. Other services, such as service discovery, also need to be performed manually. For example, if microservices need to communicate with one another, they cannot find the IP addresses of their respective containers and check if they are running. This is handled automatically by the orchestration platform (i.e., Microsoft AKS, Amazon AWS EKS, Google GKE). An orchestration platform typically runs workloads by placing corresponding containers into pods that run on nodes (i.e., a virtual or physical machine). Multiple nodes are typically grouped into clusters to ensure scalability. Pods are the smallest deployable units of computing; they group multiple containers, provide shared storage and network resources, and specify how to run the containers. The orchestration platform provides support when a pod that contains a running microservice container goes down. In that case, another pod is brought up rapidly and brings it within the purview of that service automatically. Figure 12.16 shows how microservice containers are scaled and managed in the case of failure.

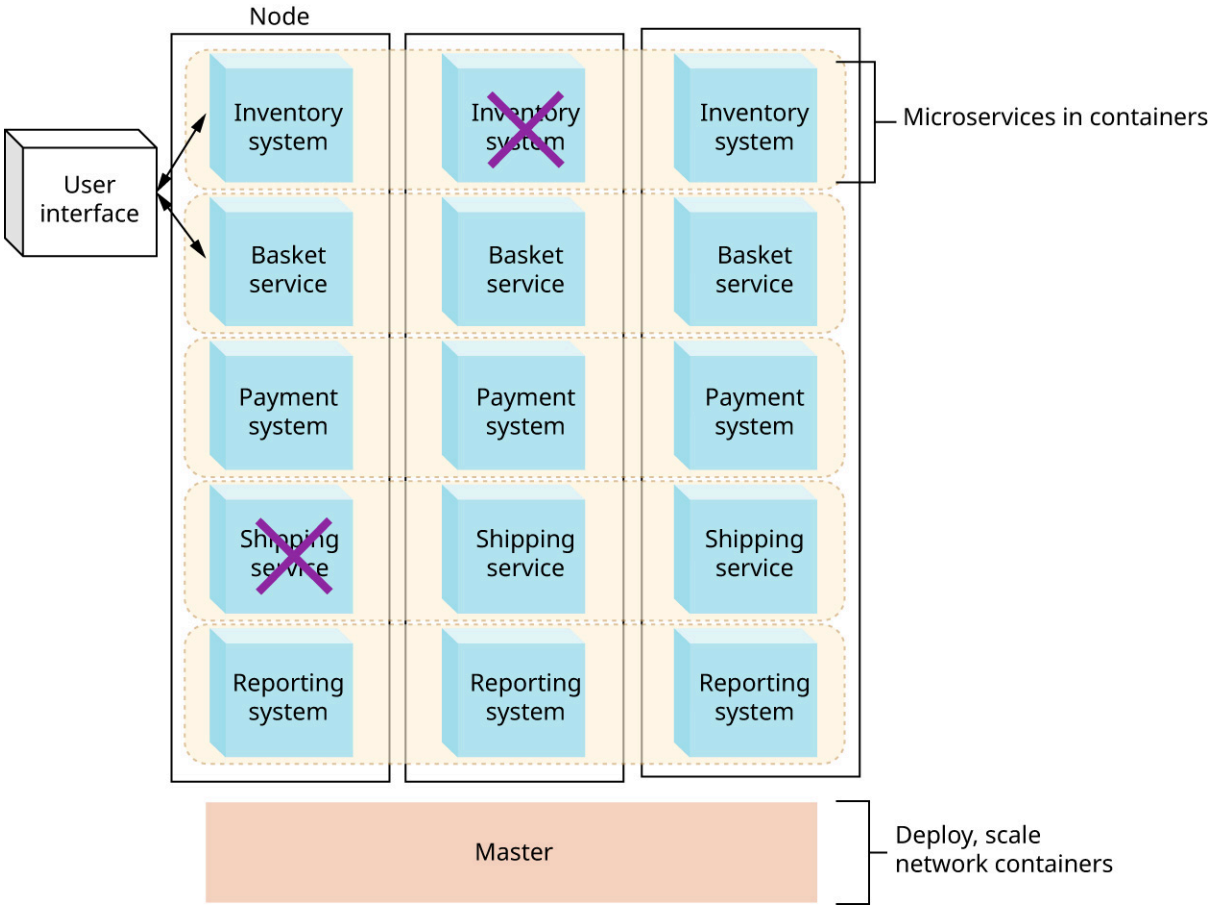


Figure 12.16 Container orchestration is used to auto-scale and perform health-checking of microservices in a scaled cloud-native application. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Agile DevOps and Automation using CI/CD

DevOps is a methodology that combines Agile software development and IT operations to deliver software faster and of higher quality. Because DevOps and cloud native focus on different aspects of the application development process, they complement each other in facilitating agility. DevOps focuses on the principles and practices used in development and operations. A goal of DevOps is to ensure that all team members are cross-functional, so they have a single mindset on improving customer experiences, responding faster to business needs, and ensuring that innovation is balanced with security and operational needs. Cloud native, on the other hand, focuses on the application environment and how to scale and deliver software more efficiently.

The use of a DevOps pipeline to migrate an application toward a microservices architecture achieves the goals of automation, discipline, and repeatability while reducing some of the challenges already identified. As a cloud-native application works its way through the DevOps pipeline, it also moves toward continuous delivery.

Figure 12.17 shows **continuous integration and continuous deployment (CI/CD)**, which is a set of DevOps operating principles that enable development teams to focus on rapid and frequent integrations of new code into the application in development as well as fast and frequent delivery or deployment of new iterations of the application. It allows teams to deliver code changes more frequently and reliably. A DevOps team establishes a CI/CD pipeline, which is a series of steps needed for integration and delivery or deployment that includes the build process through which the application is compiled, built, packaged, and tested. The **continuous integration** part automates the application compile, build, package, and testing process, enabling it to run independent of physical resources, which allows for a more consistent integration process. This improves team communication and leads to better software quality. The **continuous delivery** part goes a step further and automates the deployment stage as well. During this phase, the application is deployed to selected

infrastructure environments. Packages built during CI are deployed into multiple environments (e.g., development, staging). The application build undergoes integration and performance tests. Finally, the application is deployed into production and made available to end users.

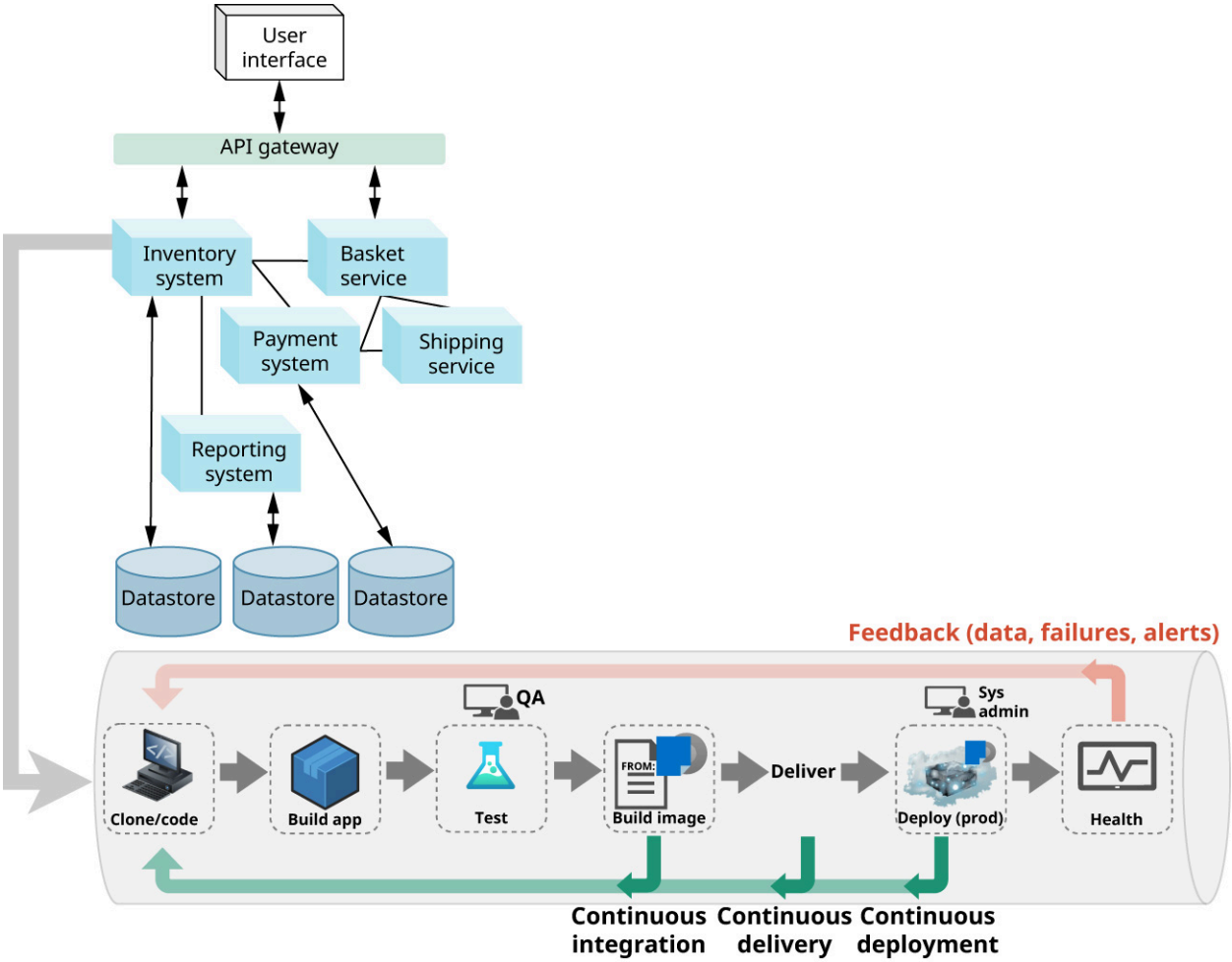


Figure 12.17 A DevOps pipeline is used for a cloud-native app as its microservices code updates move through it. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Elastic–Dynamic Scale-Up/Down

Cloud-native applications take advantage of the elasticity of the cloud by scaling microservices. Microservices that require additional resources can be scaled on demand to service an increased capacity. The microservice can also be scaled down when usage decreases. Resources that were allocated during the scale-up can be deallocated as they are no longer needed. A cloud-native application can thus adjust to the increased or decreased resources and scale as needed.

Benefits of Cloud-Native Applications

Cloud-native applications are designed and built to take advantage of the speed and efficiency of the cloud. Cloud-native applications are highly scalable, easy to update, and take advantage of cloud platforms, processes, and services to easily extend their capabilities. Some of the benefits of using cloud-native applications include the following:

- Cost-effectiveness: Orchestration tools can help scale a cloud-native application by automating the allocation of resources to the microservices as needed without having to duplicate the entire application. This eliminates the overprovisioning of hardware and the need for load balancing. Containers can also be used to reduce the complexity of managing the many microservices that make up a cloud-native

application as well as maximize the number of microservices that run on a host, saving time, resources, and money.

- Scalability: Each microservice is logically isolated and can scale independently without downtime. Microservices can scale up or down by allocating more or fewer resources to in-demand services in response to a change in user traffic. If one microservice is changed to scale, the others are not affected. Independent scalability also makes it easier to deploy or update any part of the cloud-native application without affecting the entire application.
- Portability: Cloud-native applications are vendor neutral because the containers microservices run in can be deployed anywhere, thereby avoiding vendor lock-in.
- Reliability and resiliency: If a failure occurs in one microservice, there is no effect on adjacent microservices because cloud-native applications use containers. Resiliency is provided at the core of the architecture. As with any software system, failure can also occur in distributed systems and hardware. Transient failures can also occur in networks. The ability of a system to recover from failures and continue to function is called **resiliency**. The goal of resiliency is to return the application to a fully functioning state following a failure minimizing downtime and data loss. An application is resilient if it (1) has **high availability**, which is the ability for an application to continue running in a healthy state without significant downtime; and (2) supports **disaster recovery**, which is the ability of the application to recover from rare but major incidents: nontransient, wide-scale failures, such as service disruption that affects an entire region. Applications can be made resilient by increasing redundancy with multinode clusters, multiregion deployments, and data replication. Other strategies for implementing resiliency include retries to handle transient network failures, adding more nodes to a cluster and load balance across them, throttling high-volume users, and applying circuit breakers to prevent an application from repeatedly trying an operation that is likely to fail.
  Testing for resiliency requires testing how the end-to-end workload performs under failure conditions that only occur intermittently—for example, injecting failures by crashing processes, including expired certificates, making dependent services unavailable, and so on. Resiliency tools and frameworks like Chaos Monkey can be used for such chaos testing. For example, Netflix uses Chaos Monkey for resiliency testing to simulate failures and address them.
- Ease of management: Cloud-native application updates and added features are automated as they move through a DevOps pipeline using CI/CD. This makes it easier for developers to track the microservices as they are being updated. Development teams can focus on managing specific microservices without worrying about how it will interact with other microservices. This architecture allows teams to be chosen to manage specific microservices based on the skill sets of their members.
- Visibility: Because a microservices architecture isolates services, it makes it easier for teams to learn how the microservices function together and have a better understanding of the cloud-native application as a whole.

Best Practices for Cloud-Native Application Development

Best practices for designing cloud-native applications are based on the DevOps principle of operational excellence to ensure the timely delivery of quality software. A cloud-native architecture has no unique rules, and businesses will approach development differently based on the business problem they are solving and the software they are using. Adopting the DevOps principles to develop cloud-native applications, businesses gain three core advantages, such as higher-quality software released more rapidly, faster responsiveness to customer needs, and improved working environment for development teams.

All cloud-native application designs should consider how the application will be built, how performance is measured, and how teams foster continuous improvement of the application's performance, compliance through the application life cycle at a faster pace, and higher quality. Here are the five essential best practices for cloud-native application design:

1. Automation: A development team should automate as much of the cloud-native application

development life cycle as possible. Automation helps reduce human errors and increase team productivity. It also allows for the consistent provisioning of cloud application environments across multiple cloud vendors. With automation, infrastructure as code (IaC) is used as a DevOps practice that uses versioning and a declarative language to automate the provisioning of infrastructure resources such as compute services, networks, and storage.

2. Monitoring: Teams should monitor the development environment, as well as how the application is being used. Monitoring ensures the cloud-native application performs without issues. Teams can also bolster the CI/CD pipeline with continuous monitoring of the application, logs, and supporting infrastructure. Continuous monitoring can also be used to identify productivity issues that may slow down the CI/CD pipeline.
3. Documentation: Many teams build cloud-native applications with limited to no visibility into what other teams are doing. Teams with specific skills are likely to manage certain aspects of the cloud-native application because microservices are built with different programming languages and frameworks that team members specialize in. It is important to document the specifics of the microservices they manage, track changes, and monitor team contributions to the cloud-native application.
4. Incremental releases: Changes made to the cloud-native application or the underlying architecture should be incremental and reversible. With IaC, developers can track changes in a source repository. Updates should be released as often as possible. Incremental releases reduce the possibility of errors and incompatibility issues.
5. Design for failure: Processes should be designed for the possibility of failures in a cloud environment. Implementing test frameworks to simulate failures can mitigate risks. They can also be used to learn from failures and to improve the overall functionality of the cloud-native application.

GLOBAL ISSUES IN TECHNOLOGY

Local and International Implications of Cloud Applications

While cloud-native applications are used all over the world, specific internationalization (I18N) and localization (L10N) requirements must be observed to facilitate the creation of applications that people can use. Because microservices are part of an application's back end, they typically return keywords that can be replaced via a I18N/L10N system within the application front end. What is your opinion regarding whether this approach addresses all I18N/L10N requirements for cloud-native applications?

Tools for Cloud-Native Application Development

Several tools are used for the cloud-native application development process. Together, they create a development stack. The following tools are typically found in a cloud-native development stack.

Docker

Docker is an open-source platform that creates, deploys, and manages containers using a common operating system. It isolates resources allowing multiple containers to use the same OS without contention. Docker has become the standard for container technology. An advantage containers offer is portability. Docker containers can be deployed anywhere, on any physical virtual machine or on the cloud. Using Docker helps reduce the size of development and provides smaller footprints of operating systems in containers. Typically measured in megabytes, Docker containers use far fewer resources than virtual machines and start up almost immediately. Docker containers are lightweight, which makes them easily scalable. Figure 12.18 illustrates an overview of the Docker process.

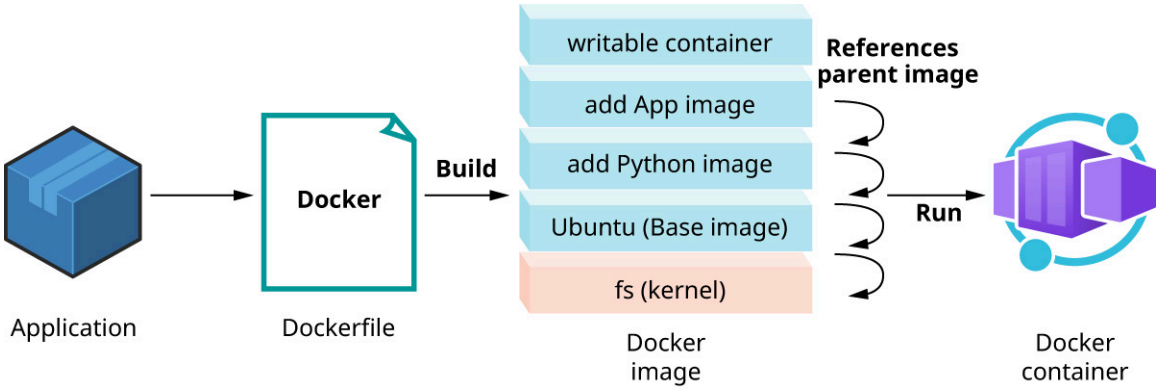


Figure 12.18 A Docker image is built from a Dockerfile to deploy an application that runs in a Docker container. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

To deploy applications using Docker, the typical first step is to create a Dockerfile. The Dockerfile is a text document that contains instructions for building a Docker image. A Docker image may be based on another image with some additional customization. Docker images are a collection of immutable layers where each instruction in a Dockerfile creates a layer in the image. Once built, Docker images become immutable templates for creating Docker containers, which are running instances of those images containing everything needed for the application to run.

Kubernetes

Kubernetes is an open-source orchestration platform used for automating deployment, scaling, and management of container-based workloads. An example of a Kubernetes platform is shown in Figure 12.19.

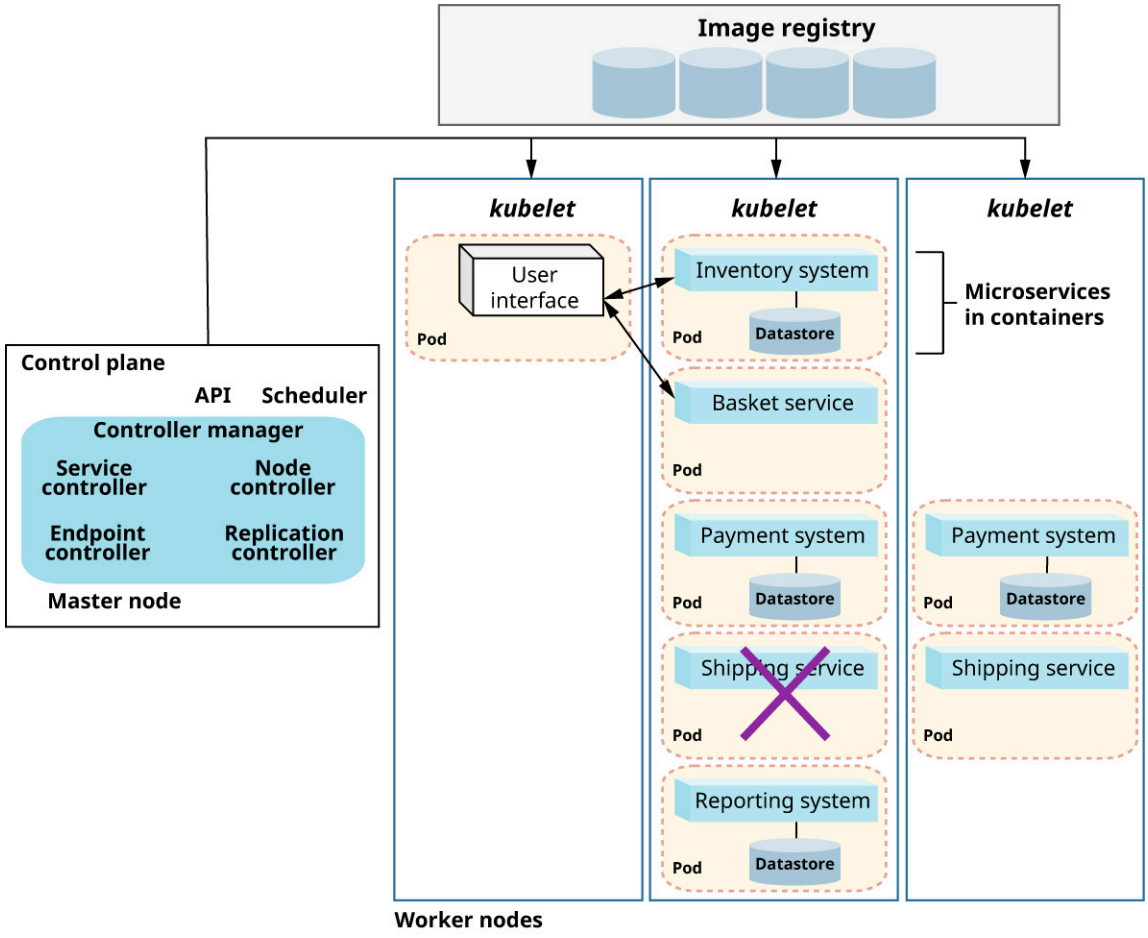


Figure 12.19 A cloud-native application is deployed in a Kubernetes environment. (attribution: Copyright Rice University, OpenStax,

under CC BY NC-SA 4.0 license)

The **workload** refers to an application being run on Kubernetes. The Kubernetes master node includes the control plane, which has several important components, including a service controller, node controller, endpoint controller, replication controller, and scheduler. These components are responsible for features such as load balancing by distributing network requests across containers efficiently; self-healing containers by regularly checking them for good health and where failed containers are replaced or restarted automatically; auto-scaling containers by adding or removing them in accordance with demand; automated deployment to handle setting up and deploying containers to the cloud; storage management to handle the storage needs of all the containers; and handling networking between containers.

The Kubernetes master node comprises several components, including the Kubernetes API server, which plays a crucial role in managing container-based workloads. Additionally, the Kubernetes cluster consists of worker nodes, each containing a kubelet. Working in tandem with the master node, the **kubelet** handles tasks like scheduling and ensuring the smooth operation of applications deployed within the worker nodes. Kubernetes has evolved into the preferred platform for deploying cloud-native applications. In this ecosystem, the microservices of a cloud-native application interact over the network, deployed and scaled within a Kubernetes cluster.

To deploy a cloud-native application in a Kubernetes environment, microservice containers, pulled from a registry, are deployed in pods. A **pod** is a small logical unit that runs a container within a worker node in the Kubernetes cluster. A cluster is a set of one or more nodes. Pods are a group of one or more containers with shared storage network resources and a specification for how the containers are run. Containers should only be scheduled together in a single pod if they are tightly coupled and need to share resources. Pods are ephemeral, meaning they are nonpermanent resources. Kubernetes manages the pods rather than the containers directly. The API server on the Kubernetes master node communicates with the worker node to deploy the container in a pod and start it up. Replicas of pods can be increased to scale the containerized microservice within it. If a pod fails, a new one is created. Kubernetes manages these deployments for the length of the deployment until the pod is deleted. In other words, it ensures that all pods and replicas are running.

Service discovery is another important service provided by the Kubernetes platform. As pods are added and replicated, an internal IP address is provided for the pod, making the container accessible. Pods are ephemeral so IP addresses can change as new pods are created. Service registry and service discovery capabilities address this issue by creating Kubernetes services. A Kubernetes service is an abstraction, which defines a logical set of pods and a policy by which to access them in a reliable way.

THINK IT THROUGH

Are Docker and Kubernetes Too Broad?

Given the fact that many tools are available for containerization and container orchestration, what justifies the broad acceptance of Docker and Kubernetes to handle these respective functions?

Terraform

Terraform is an open-source IaC tool used to build, update, and version cloud and on-premises resources. Some use cases for Terraform include managing Kubernetes clusters and provisioning resources in the cloud, among many others.

GitLab and GitHub CI/CD

GitLab is a cloud-based DevOps platform used to monitor, test, and deploy application code. GitLab includes a

cloud-based Git repository as well as several DevOps features such as CI/CD, security, and application development tools. The CI/CD tool can be used to automate testing, deployment, and monitoring applications. GitLab can also be used for security analysis, static analysis, and unit testing. GitHub is a cloud-based developer platform used for software development, collaboration, and security. GitHub also includes a cloud-based Git repository and several DevOps features similar to GitLab.

Red Hat OpenShift

OpenShift is Red Hat's cloud application platform. It includes several containerization software products, including the OpenShift Container Platform that runs on the Red Hat Linux operating system and Kubernetes. OpenShift is a Platform as a Service (PaaS) that includes the Kubernetes platform, Docker container images, as well as features that are exclusive to the OpenShift platform. Such features include CI/CD pipeline definitions, container automation tools, Kubernetes command-line interface, and security features. OpenShift provides a robust, multilanguage development environment, plus all the necessary software components to support applications. OpenShift is open-source and free-to-download application that enables developers to quickly deploy web, mobile, and IoT applications.

Tanzu

VMWare's cloud application software **Tanzu**, formerly known as Cloud Foundry, is a modular, application platform that provides a rich set of developer tools and a pre-paved path to production to build and deploy applications quickly and securely on any compliant public cloud or on-premises Kubernetes cluster. Tanzu provides a supply chain of operational and security outcomes for the development environment via configurations that are designed with operational and security principles so that applications can be pushed into production quickly, run, and scaled safely and securely. Such principles include running code through a testing environment and ensuring the code runs properly, applying security scanning so that the application that goes into production is audited and compliant with security policies, and deploying the application to production environments that may include several cluster environments that can run and scale in production.

Node

Node is an open-source, cross-platform, server-side JavaScript runtime environment that can be used to develop server-side tools and applications in JavaScript. Some examples of real-time applications include chats, news feeds, and other microservices. For example, Node can be used to create virtual servers and define the routes that connect microservices to external APIs such as operating system APIs, including HTTP and file system libraries, compared to browser-specific JavaScript APIs.

LINK TO LEARNING

This Cloud Native Trail Map (https://openstax.org/r/76clnativemap) illustrates the use of various open-source tools to develop cloud-native applications.

The Future of Cloud-Native Applications

Cloud-native applications have seen increased use in recent years and are predicted to be the future of software development. The Cloud Native Computing Foundation[11] estimated there were at least 6.8 million cloud-native developers in 2021 compared to 6.5 million in 2020. They also estimated there were 5.6 million developers using Kubernetes in 2021, up 67% from a year ago.[12]

Cloud-native applications solve some of cloud computing's inherent problems. The cloud-native approach is the new standard for enterprise architecture. It is a way of designing, building, and running applications in the

11 https://www.cncf.io/
12 https://www.cncf.io/blog/2021/12/20/new-slashdata-report-5-6-million-developers-use-kubernetes-an-increase-of-67-over-one-year/

cloud. It has been observed that companies that adopt a cloud-native approach can achieve higher levels of innovation, agility, and scalability. It also offers many benefits such as:

- reducing IT overhead and management costs by providing a streamlined software delivery process and on-demand consumption of resources
- reducing time to market and reducing the risk of deployments by enabling developers to rapidly build, test, and deploy new and updated services
- ability to react faster in the market due to constant availability of resources
- reduction in complexity
- less coupling between the services in an application

Nevertheless, migrating to the cloud to improve operational efficiencies has a range of challenges. We also pointed out earlier some of the challenges faced in microservices architectures. Some of the challenges that were learned earlier include to use microservices requires changing the allocation of responsibility between components, and determining the components' service boundaries could be difficult, debugging, logging, and monitoring microservices also become challenging, to name a few.

TECHNOLOGY IN EVERYDAY LIFE

Everyday Life and Microservices

There are myriads of web apps that people use every day that are based on microservices. For example, weather apps typically leverage web services that invoke microservices. More complex applications include e-commerce websites, as illustrated earlier. Can you provide additional illustrative scenarios that demonstrate how the use of microservices help people in everyday life? Your scenarios should not be limited to describing how microservices are used, but rather should describe situations where these architectures are applied in real-life contexts.

12.2 Cloud-Based and Cloud-Native Applications Deployment Technologies

Learning Objectives

By the end of this section, you will be able to:

- Understand cloud deployment technology
- Relate to cloud deployment technology options
- Explain cloud deployment technology use cases and implementation
- Select cloud deployment technologies
- Relate to the future of cloud deployment technologies

Various cloud technologies may be used to deploy cloud-based and cloud-native applications. These technologies differ in their implementation and use. While these technologies span storage, network, and compute services, the emphasis will be on compute options. There are many deployment options available, and there is no right or wrong choice. Only a subset of these options is applicable to the deployment of cloud-native applications. However, all options will be covered because enterprise solutions are typically hybrid solutions that may combine on-premises, cloud-based, and cloud-native components. Many organizations will need to implement more than one option. Accordingly, selecting the best option to support workloads while controlling complexity can be a daunting task.

Introduction to Cloud Deployment Technology

The delivery of computing services, such as databases, software, analytics, and storage over the Internet is called **cloud computing**. The cloud computing model is composed of three service models—Infrastructure as

a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS)—and four deployment models—private cloud, community cloud, public cloud, and hybrid cloud. Figure 12.20 describes the relationship between cloud deployment models, service models, and cloud deployment technologies.

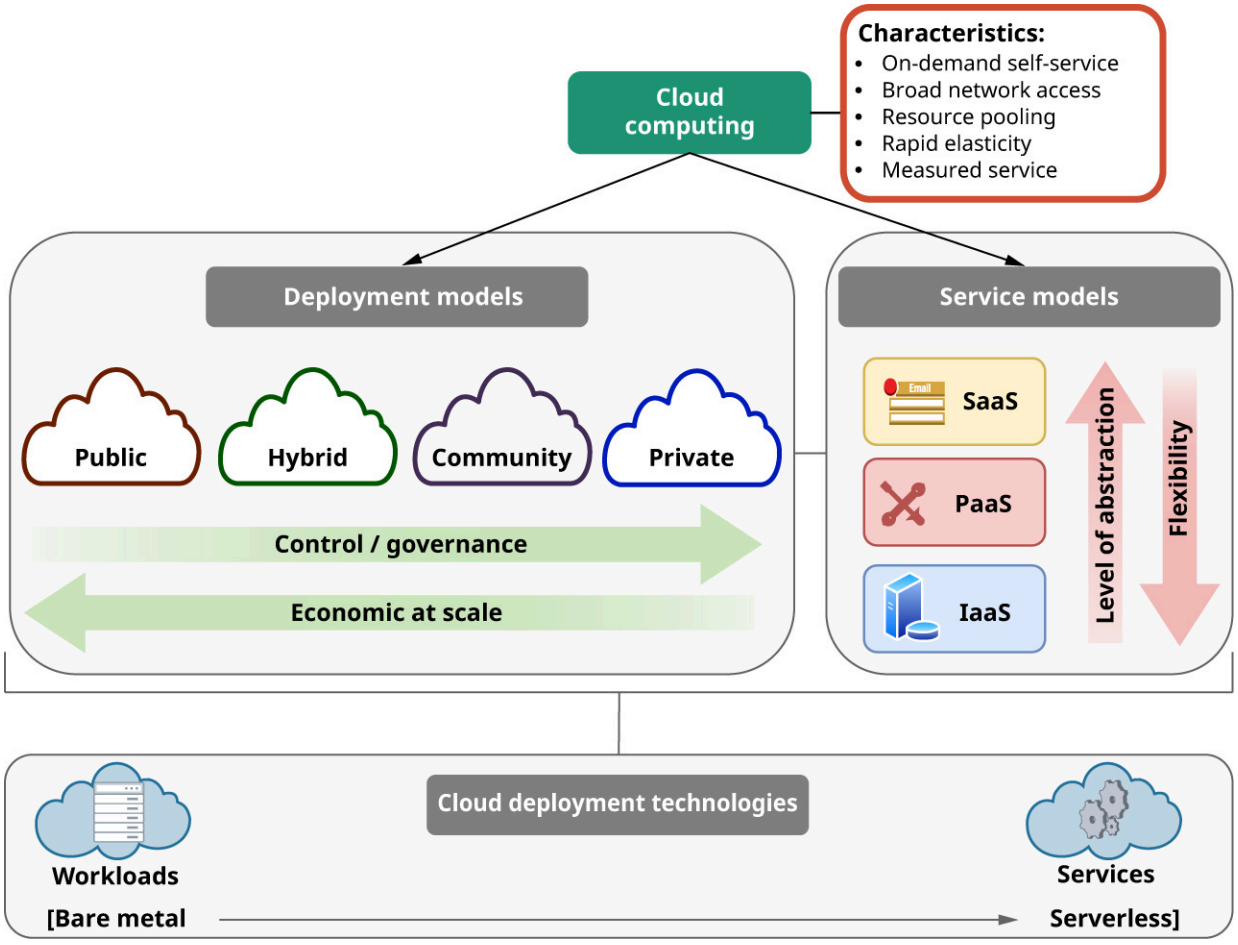


Figure 12.20 Cloud deployment models, cloud service models, and cloud deployment technologies are all considered for workload and service deployments. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

When examining workload and service deployments, there are three key architectural considerations. First, the deployment models in the cloud are evaluated, offering choices for where service models are deployed, such as public, hybrid, community, or private cloud environments. Second, the service model options are assessed to determine the level and type of services, including Infrastructure as a Service (IaaS) and Platform as a Service (PaaS). Each of these cloud service models will be discussed in more detail. Finally, workloads and services are actualized in the cloud using various deployment technologies, ranging from bare metal (i.e., high-performance physical servers dedicated to a single customer where the burden of provisioning, maintaining, or scaling applications on these servers falls on the customer) to serverless computing (i.e., where customers deploy applications without the burden of provisioning, maintaining, or scaling these applications as they are provided by the cloud service provider). These technologies serve as the foundation for executing cloud-based and cloud-native applications, allowing organizations to harness the benefits of cloud computing. Understanding these options empowers companies to select the most suitable approach, ensuring performance, deployment speed, scalability, portability, and security align with business requirements while managing costs effectively. Enterprises must consider different factors when deploying various workloads and services in a cloud environment. For instance, **high-performance computing (HPC)** demands significant compute and memory resources, whereas legacy applications may have lower resource requirements.

Cloud Service Models

The three cloud service models—IaaS, PaaS, and SaaS—define a combination of information technology resources (e.g., compute servers, storage, memory, middleware) offered by a cloud provider and are not mutually exclusive. These service models change the way information technology (IT) resources are consumed. Traditionally, IT resources were purchased, managed, and maintained in a company's on-premises data center. The cloud computing service models, in contrast, provide a more economical solution where these IT resources are accessed and scaled on-demand from a cloud service provider at a predictable cost. These service models are categorized as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). The following sections elaborate on these three cloud service models.

Infrastructure as a Service (IaaS)

Infrastructure as a Service (IaaS) is on-demand access to a cloud-hosted computing infrastructure that includes servers, storage, and network resources. These resources can be provisioned, configured, and used by the customer like on-premises resources are. The difference is the cloud service provider hosts, manages, and maintains these resources in its own data centers, as shown in Figure 12.21.

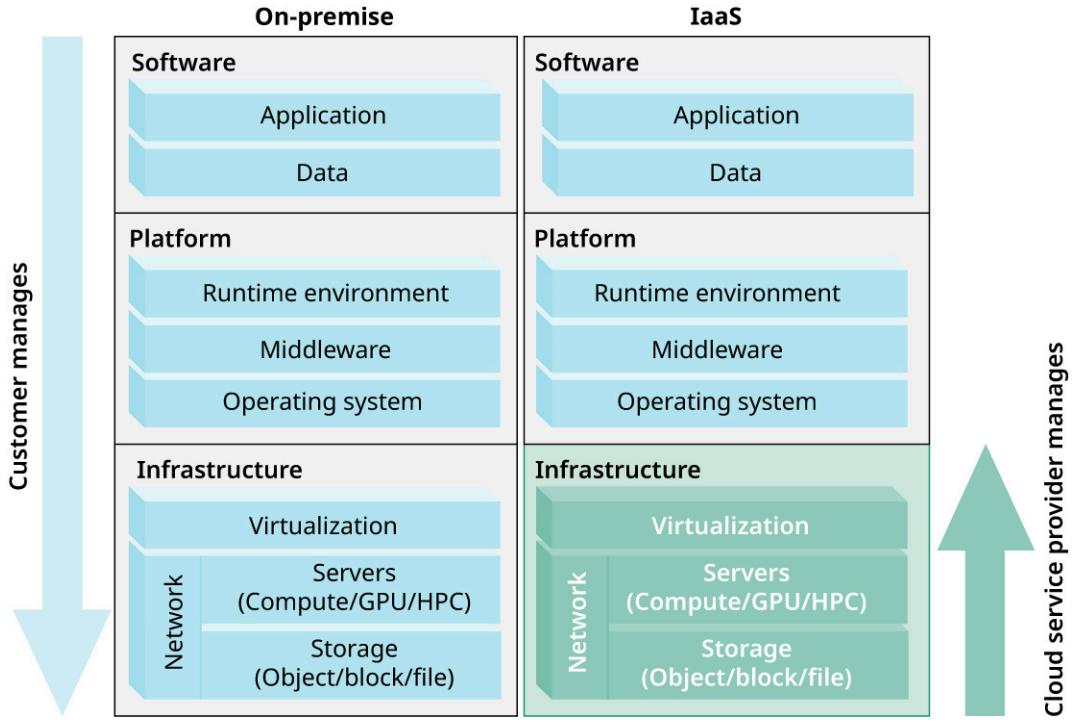


Figure 12.21 IaaS resources are managed by a cloud provider compared to on-premises resources that are managed by the customer. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

IaaS customers access these resources via the Internet and pay for these services on a subscription or pay-as-you-go basis. Infrastructure consists of three main components: compute for processing tasks, storage, and network. The compute component could, for example, include a general-purpose processing capability or more specific processing capabilities such as a high-speed graphics processor (GPU) or high-performance computing (HPC). The storage component could support different storage needs such as object storage, block storage, or file storage. Finally, the network component allows for the compute and storage components to communicate.

IaaS resources are multitenant and are made available to multiple different customers simultaneously. The cost for the services varies depending on the technologies chosen. IaaS customers make use of these services on-demand where the demanded resources are provisioned and delivered to the customer in a matter of minutes or hours as opposed to days, weeks, or months as is with traditional IT on-premises resources. IaaS

also provides the flexibility for customers to scale up or down resources depending on their needs.

Platform as a Service (PaaS)

As discussed previously, IaaS is a set of compute, storage, and networking resources that have been virtualized by a cloud service provider and configured by the customer to suit their needs. PaaS takes advantage of all the virtualized resources from IaaS and adds to this. **Platform as a Service (PaaS)** is on-demand access to a cloud-hosted platform for developing, running, and managing applications by prepackaging middleware, language runtimes, and tools as containers. The cloud service provider hosts, manages, and maintains all the software included in the platform. This includes the servers, storage, and networking services. It also includes the operating system, middleware, and runtime environment. Figure 12.22 illustrates how PaaS customers can deploy their application and data.

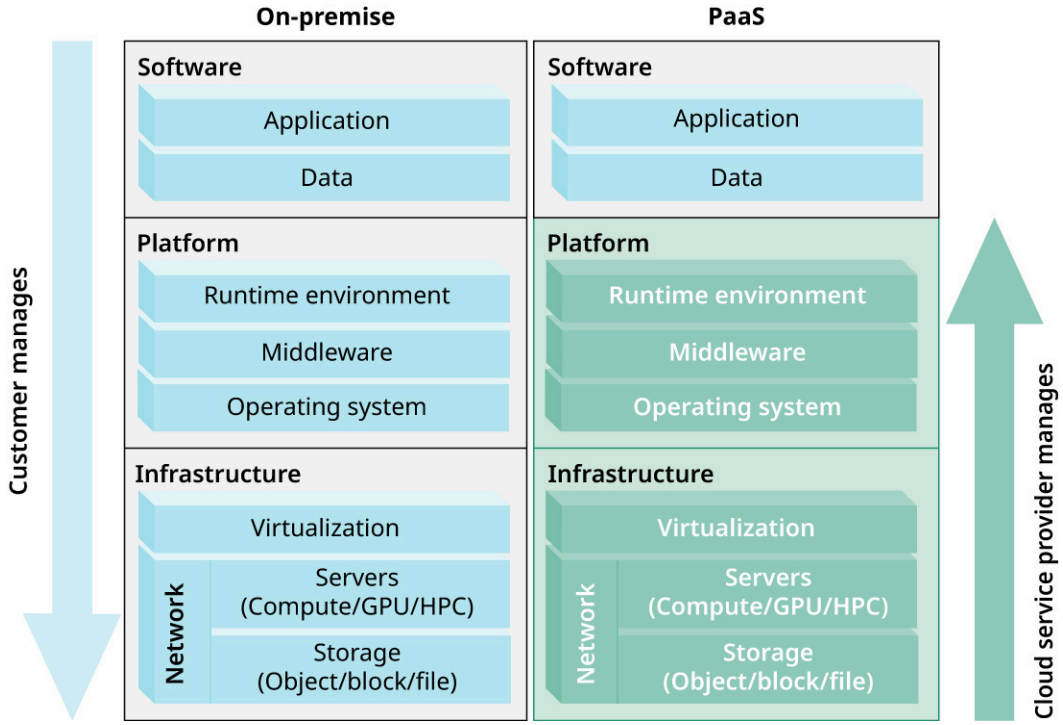


Figure 12.22 PaaS resources are managed by a cloud provider compared to on-premises resources that are managed by the customer. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

PaaS generally makes it easy to get an application up and running quickly. PaaS is cost effective in that customers don't have to bear the cost to manage the runtime environment for their applications. PaaS cloud providers generally provide tools such as DevOps and collaboration tools, as well as API marketplaces as services that could easily be integrated with applications. PaaS also provides the flexibility for customers to scale resources up or down depending on their needs.

Software as a Service (SaaS)

Software as a Service (SaaS) is a cloud-hosted, ready-to-use software or application that end users access with a client (e.g., web browser, desktop client, mobile app) via a subscription model. SaaS takes advantage of all the resources from PaaS, but also includes the application and data, as shown in Figure 12.23.

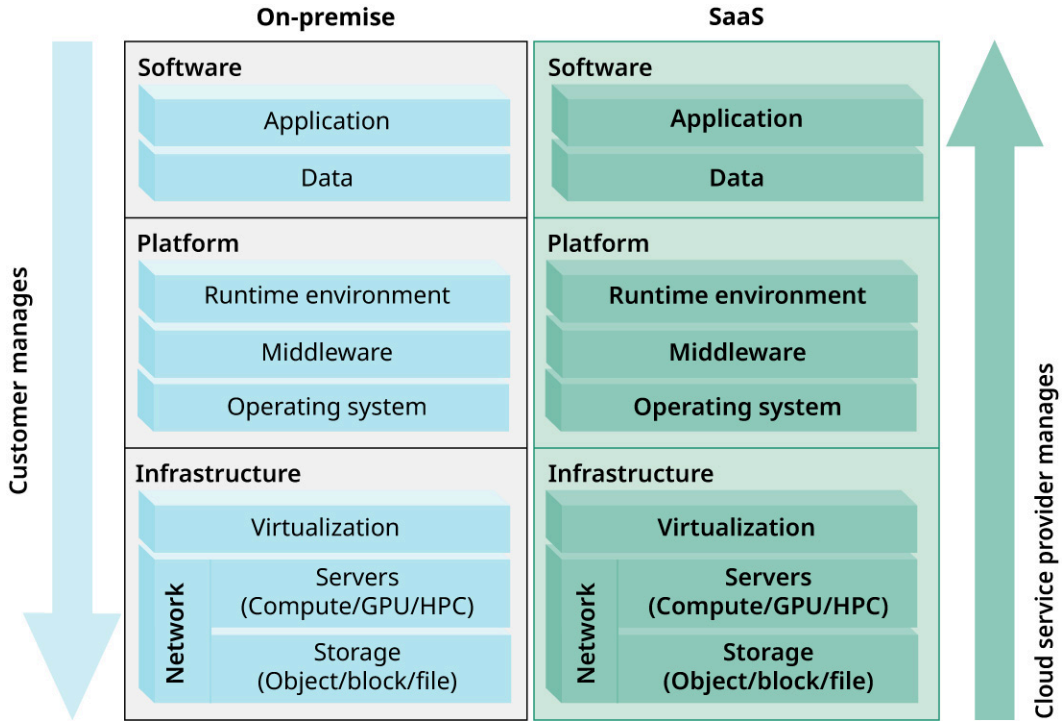


Figure 12.23 SaaS resources are managed by a cloud provider, like PaaS resources, but they are subscription based and include applications and data. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Unlike IaaS and PaaS, SaaS has the highest level of abstraction where the cloud service provider hosts, manages, and maintains all layers of the stack, including the application as well as all the infrastructure required to provide the application service. This allows for the SaaS application to always be upgraded to the latest version. SaaS applications are scalable at any level of the stack. Additional infrastructure resources such as compute power and platform resources, such as databases, can be added as needed.

SaaS resources are also multitenant where multiple users are all accessing the same pool of resources. Tenants would have access to the same hosted environment; however, they would have their own dedicated space to securely store their data. SaaS applications are cost effective compared to the other cloud service models because the cloud service provider maintains and manages the entire stack. SaaS applications are also cost effective compared to traditional applications hosted on-premises where companies would have to bear the cost of the hardware in addition to continued cost for IT support.

One important benefit of cloud computing is automation. Depending on the service model chosen, organizations can reduce the layers of management of hardware, infrastructure, and applications. This reduces costs and effort of investment into these resources and allows organizations to focus more on innovation. For an organization to choose between the cloud service models and on-premises solutions, it must balance the level of complexity and effort of investment in each solution. These solutions present a spectrum from complex and high effort of investment to simple and low effort of investment, as shown in Figure 12.24.

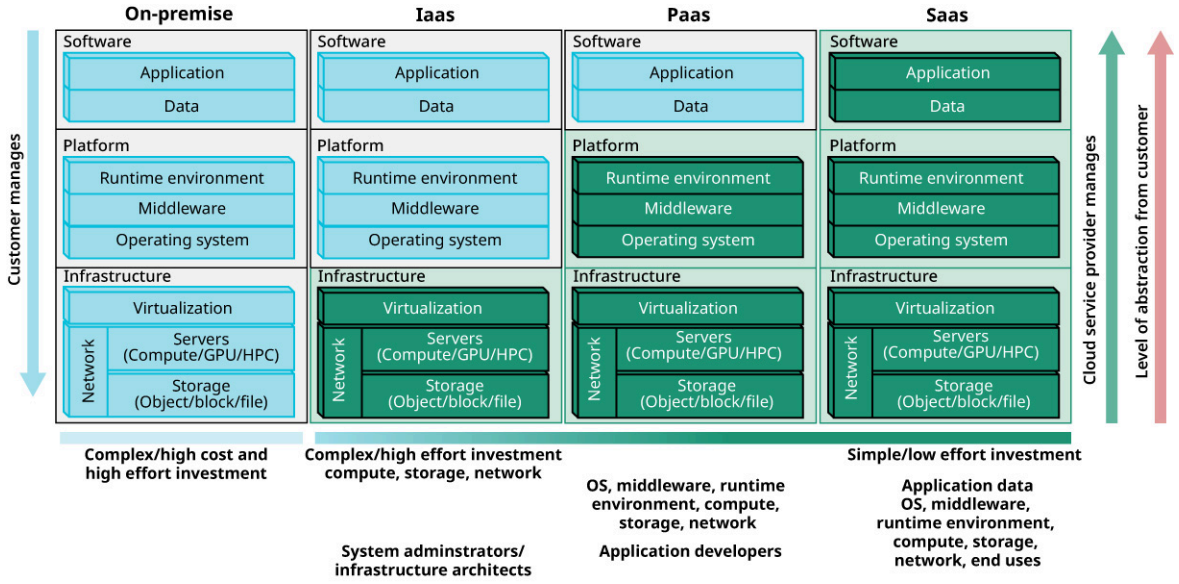


Figure 12.24 Complexity and level of investment effort decrease as level of abstraction from customer increases depending on the solution chosen. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

On-premises and IaaS solutions offer the greatest level of flexibility but are the most complex and require more effort of investment. These solutions also increase costs as they require the hiring of systems personnel as these solutions require a higher level of knowledge and management of infrastructure resources. Moving from left to right, PaaS solutions are less complex with less effort of investment and are generally managed by application developers. Finally, SaaS is the least flexible, but also the least complex, and requires the least effort of investment. SaaS solutions are applications or software that don't require any installation or manual upgrading, and so the end user can be anyone. Many organizations use a combination of these solutions with the flexibility to change the service models used as their needs change.

Cloud Deployment Models

Cloud deployment models dictate how cloud services are implemented, hosted, and accessed by end users. They revolve around the principle of virtualizing server computing power into segmented, software-driven applications offering processing, storage, and networking capabilities. The following sections elaborate on various cloud deployment models, including private, community, public, and hybrid clouds.

Public Cloud Model

A **public cloud** is a cloud deployment model where resources are remotely accessible by anyone offered through subscriptions or on-demand pricing plans. A public cloud deployment model is shown in Figure 12.25. This approach allows for the flexibility of provisioning resources on-demand and pay only for what is consumed. Public cloud resources are owned and managed by the cloud service provider. This type of cloud hosting allows cloud service providers to provide various services to a variety of customers. A cloud service provider's resources are provisioned for any type of customer from individual users to major industry groups.

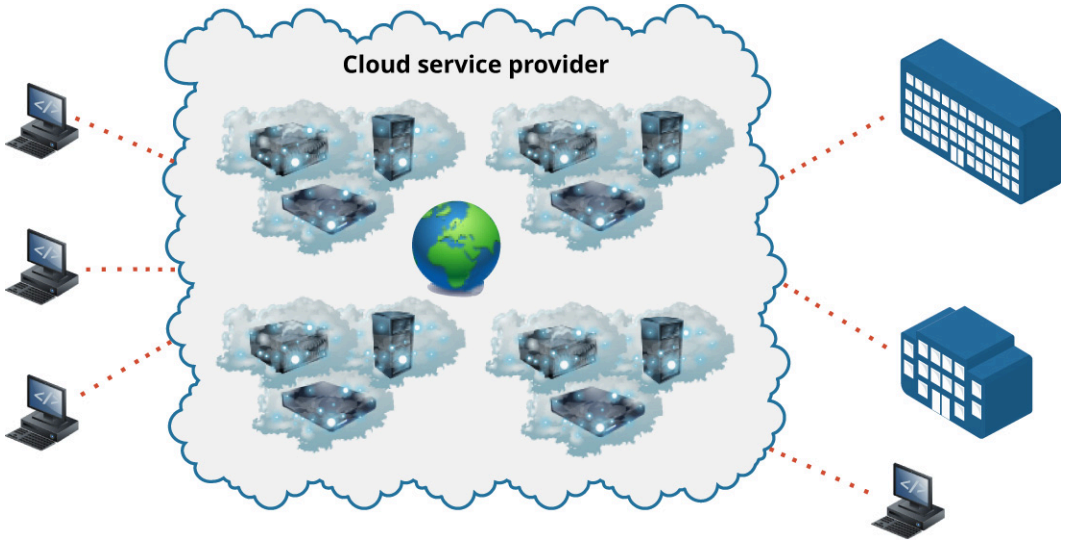


Figure 12.25 A public cloud deployment model consists of resources accessible to all users. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Private Cloud Model

A **private cloud** is a cloud deployment model where resources are dedicated to one customer such as a single organization for internal use where cloud resources are accessed and managed by an organization. A private cloud deployment model is shown in Figure 12.26. It is not open to the public. Private clouds are isolated and contain the proper security to restrict access to them. They are protected with robust firewalls and a supervised secure environment. Private clouds could run on-premises but can also run on vendor-owned data centers remotely. Private cloud resources are managed either internally by the organization or by a third party. Figure 12.26 shows two private clouds. A cloud service provider's secured resources are provisioned and dedicated to an organization.

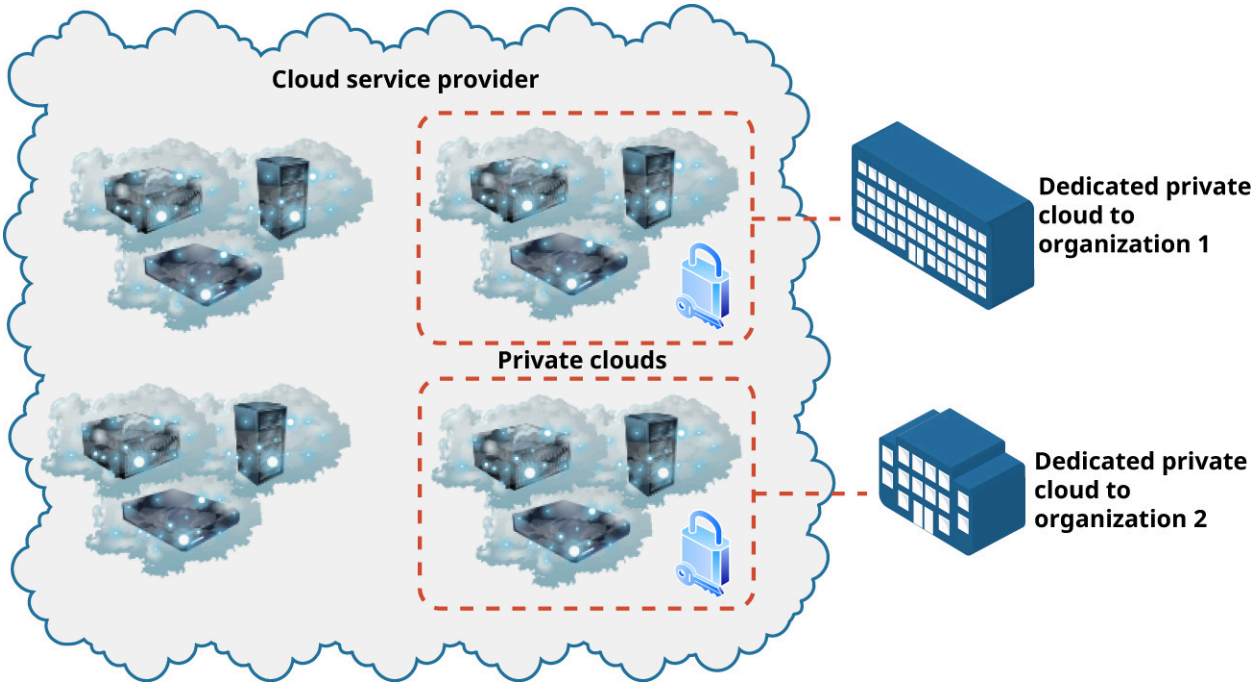


Figure 12.26 A private cloud deployment model consists of dedicated resources with restricted access. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Community Cloud Model

A **community cloud** is a cloud deployment model where resources are only accessible by a selected group of organizations. The resources of a community cloud are mutually shared between the organizations that belong to a particular community or industry. Examples of community clouds are clouds for banks or governments. The community or industry members generally share similar privacy, performance, and security concerns. A community cloud consists of an infrastructure that integrates the services of different clouds to meet the specific needs of the community or industry. Community cloud resources are managed either by members of the community or industry or by a third party. Figure 12.27 outlines a community cloud deployment model and shows two community clouds. A cloud service provider's resources are provisioned and dedicated to a community of organizations or government entities.

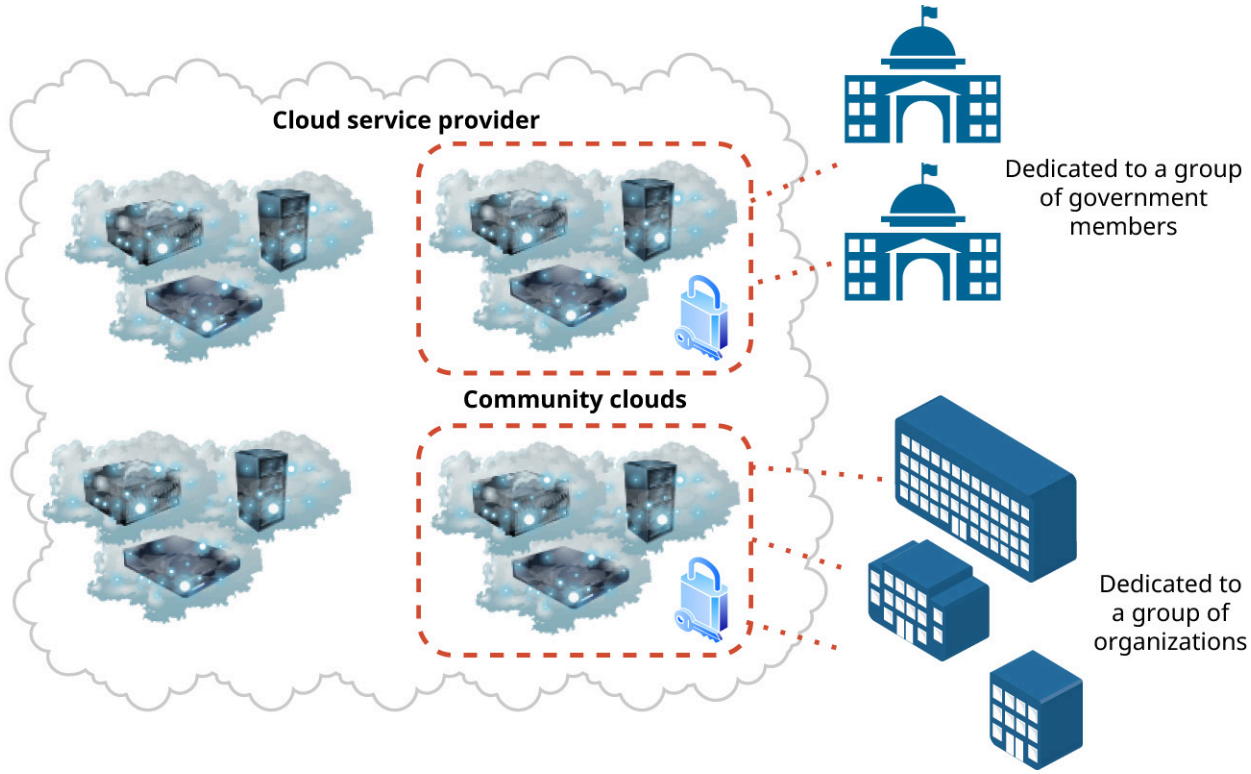


Figure 12.27 A community cloud deployment model consists of dedicated resources limiting access to a community. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Hybrid Cloud Model

A hybrid cloud is a distributed computing architecture with two or more environments consisting of private and public clouds and on-premises infrastructure. A hybrid cloud deployment model is shown in Figure 12.28. This model provides greater flexibility compared to other cloud deployment models because it provides orchestration and management across all environments that lets customers run workloads wherever they need them increasing workload portability across all cloud environments.

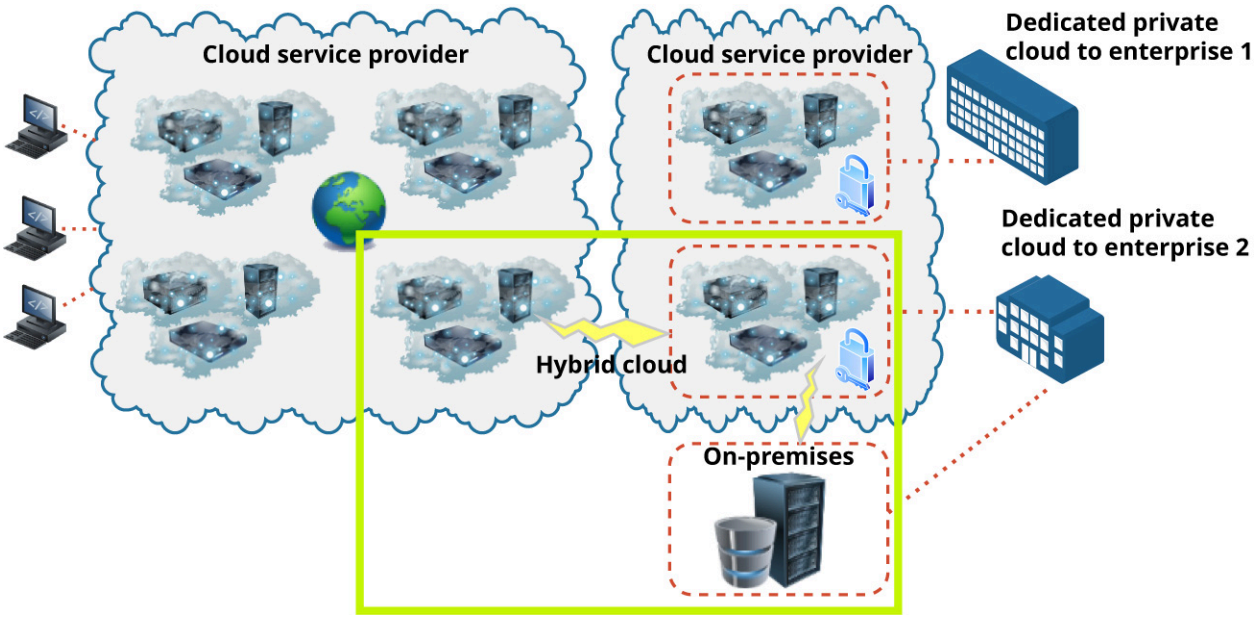


Figure 12.28 A hybrid cloud deployment model combines two or more cloud deployment models. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Cloud Deployment Technology Options

Cloud deployment technologies offered by cloud service providers include bare metal servers, virtual machines (VMs), and containers. The following sections elaborate on each of these deployment technologies. Another deployment technology that is available, however not widely used, are unikernels. Unikernels are also discussed in detail in the following sections. IaaS and PaaS are two service models through which these deployment technologies are provided. There are additional service models through which deployment technologies are provided: Containers as a Service (CaaS) (i.e., a version of IaaS where applications are deployed, managed, and scaled using a container-based virtualization); and serverless computing, also known as Function as a Service (FaaS). Figure 12.29 shows how PaaS and FaaS may leverage CaaS or IaaS for deployment. CaaS is positioned between IaaS and PaaS in the cloud computing stack. CaaS leverages the virtualization of compute, storage, and network resources from the underlying IaaS infrastructure. PaaS allows users to focus on application dependencies and runtime environments while having less control of the operating system and limited portability. CaaS, on the other hand, returns this control as it provides increased portability by facilitating operating system virtualization and customization in containerized deployments.

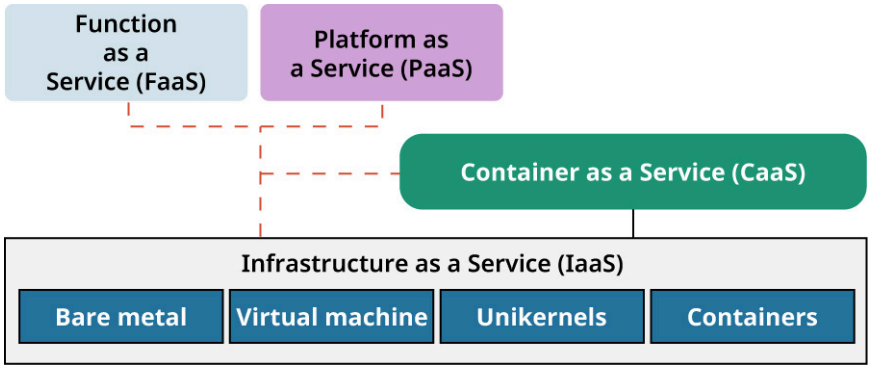


Figure 12.29 Cloud deployment technology options are available when IaaS is leveraged by other service models, including FaaS, PaaS, and CaaS. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

IaaS Deployment Options

Bare metal servers, VMs, unikernels, and containers are made available in an IaaS cloud service model. These deployment options offer a level of infrastructure control to address specific performance requirements,

compliance considerations, and legacy dependencies. Bare metal servers offer the most control and flexibility but with more overhead to maintain them. Moving from bare metal toward containers, the deployment options require less overhead because there is an increased level of automation that makes these options quicker and easier to provision or get up and running. They are also more portable. These four deployment options, in order of more control versus more portability, are illustrated in Figure 12.30.

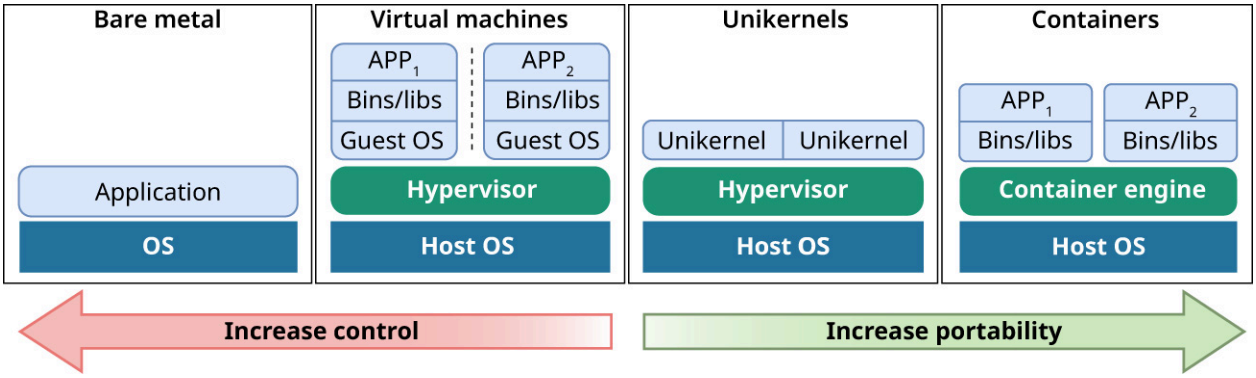


Figure 12.30 IaaS deployment options offer either more control or provide more portability. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Bare Metal Server

A **bare metal server** is a high-performance cloud server that is composed of a single-tenant, nonvirtualized physical server. Customers have complete control over the server's physical components of hardware, compute, and storage resources to optimize them to accommodate specific workloads. These servers may run any amount of work for the customer, or may have multiple simultaneous users, but they are dedicated entirely to the customer.

Virtual Machines

A **virtual machine (VM)** is a virtualization of a computer system. The process of creating software-based "virtual" versions of something such as compute, storage, networking, or applications is called virtualization. Virtualization is feasible because of a hypervisor, the software that runs on top of a physical server or a compute host that virtualizes it.

Hypervisors virtualize the resources, such as CPUs, RAM, network, and, in some cases, storage, of the physical server. The hypervisor divides these resources while allocating what is needed for each operating system of a virtual server instance (VSI). The hypervisor, also referred to as a virtual machine monitor (VMM), is a software layer that lies between the physical hardware and the VSIs that run on top of it. The hypervisor allows for the scheduling of multiple VSIs with these divided resources. Thus, multiple virtual servers can run on a single physical server. Hypervisors also provide a level of security between VSIs so that data on one VSI is not accessible from another, maintaining isolation between them, as indicated by the dotted lines between the VSIs in Figure 12.31.

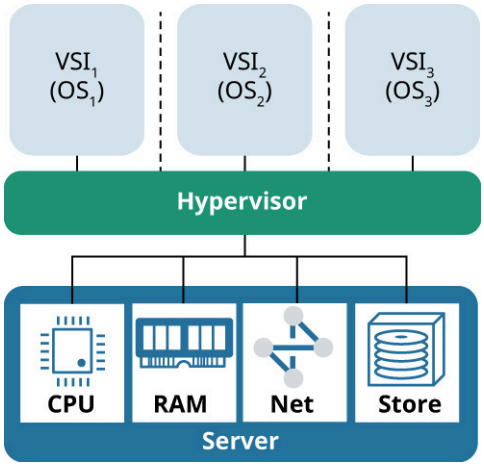


Figure 12.31 Virtualization of compute resources via hypervisors is used to schedule VSIs. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Virtualization via hypervisors makes multitenancy of VSIs possible. This drives the cost of compute down. Physical servers individually can be quite costly. Virtualization is a cost-effective way to run multiple virtual servers on a single physical computer, thereby maximizing the utilization of resources and reducing cost.

As shown in Figure 12.32, VMs run like a physical server that runs on a hypervisor. They can run their own different operating systems and are completely independent of one another. Running multiple VMs from one physical server also drastically reduces a customer's physical infrastructure footprint. VMs increase agility and speed because they can be provisioned quickly and easier, with automation, compared to provisioning an entire new physical server and environment. This also lowers downtime in case a VM unexpectedly fails because they are portable as they can be moved from one hypervisor to another on a completely different physical server quickly.

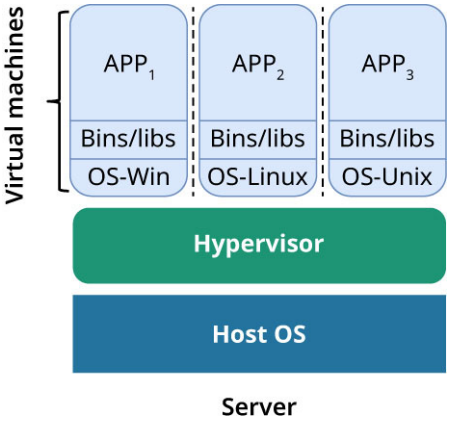


Figure 12.32 Multiple VMs are created and run on a hypervisor. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Unikernels

Originally introduced as MirageOS, a **unikernel** is a single-purpose machine image that is compile-time specialized into a dedicated stand-alone kernel with only the libraries needed to run the application. Once deployed to a cloud platform, unikernels are protected against any modifications. Unikernels are built by compiling the application source code directly into a customized operating system that includes only the functionality required by the application logic, as shown in Figure 12.33. They are specialized machine images that run directly on a server's hypervisor or on bare metal. Unikernels are considered more secure, compared to other cloud deployment technology options, due to their smaller attack surface.

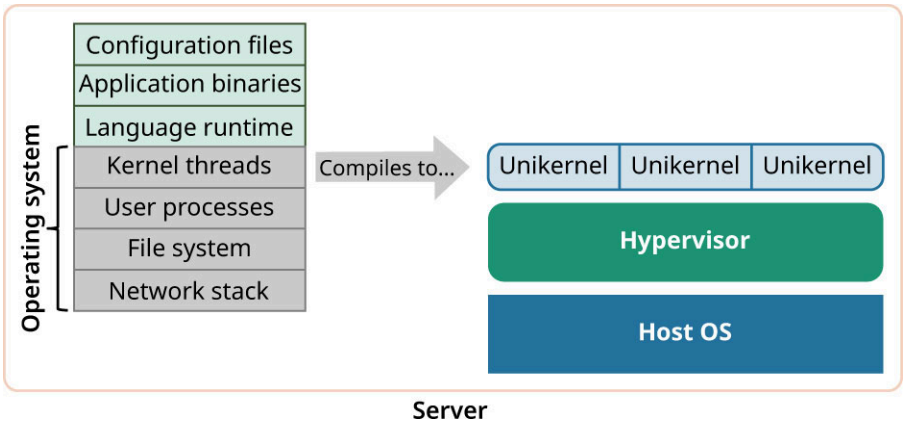


Figure 12.33 Multiple unikernels are built as specialized images and run on a hypervisor. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Unikernels provide all the advantages of VMs and containers while reducing the server footprint needed to deploy applications, because unikernels offer a significant reduction in image sizes, have a reduced memory footprint, and greatly reduce the need for disk space. The unikernels architecture increases agility and speed. They have minimal overhead, and because they have faster load times with lower latencies, they can start up very quickly, making them faster than containers and VMs. Unikernels are also very portable. Their small size makes it easier and more cost effective to move them from one hypervisor to another.

The adoption of unikernels remains low due to a combination of lack of awareness and low availability of orchestrators. They are becoming more prevalent in single-purpose devices such as network function virtualization (NFV) appliances, however data center adoption remains low.

Containers

Containers are a form of “operating system virtualization” that is more efficient than hardware virtualization because they utilize a full system hardware virtualization. In contrast, containers leverage and run on a single instance of an operating system where the virtualization is managed by the host operating system, as shown in Figure 12.34.

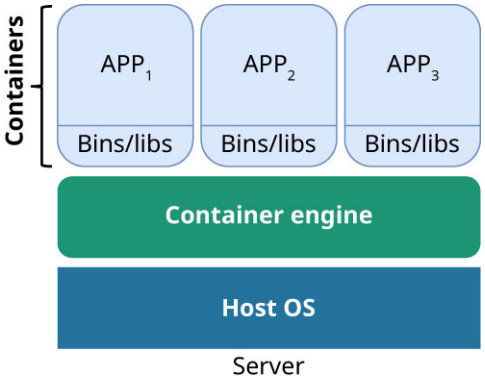


Figure 12.34 Multiple containers are built and run on a container engine. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

While VMs run as isolated servers on a hypervisor, containers share the same operating system and kernel of the host server. It appears as if each container runs on its own operating system and contains all the required binaries, libraries, and application code that is needed to run the application deployed in it. This is possible because containers run as isolated processes making it possible to run many containers on a single bare metal server or VM without any interference between them. Containers access shared resources of the operating system, but also remain as isolated processes because the shared kernel manages process isolation. Namespaces and control groups (cgroups) also help provide the illusion of process isolation. Namespaces allow for the customization and the appearance that each container instance has its own operating system.

Control groups, on the other hand, monitor and manage shared resources controlled by the containers, which helps significantly reduce the number of compute instances needed to run applications.

Container deployment options include building your own container service, utilizing off-the-shelf Containers as a Service platforms, or choosing managed container services, such as managed Kubernetes, provided by large cloud service providers (CSPs). A **Container as a Service (CaaS)** is a type of IaaS specifically geared toward efficiently deploying, running, scaling, and managing a single application using a container-based virtualization. Container deployment using the build-your-own-service option requires the customer to manage the container technology, container scheduling and orchestration, as well as cluster management. Customers deploy the containers on top of IaaS, either bare metal or VM. The CSP, however, manages the underlying infrastructure.

Container deployment using an off-the-shelf CaaS platform requires the customer to containerize the middleware, libraries, and applications or microservices. Customers deploy the containers on-premises or in a public (or off-premises private) cloud environment. This option has a higher overhead for the customer compared to PaaS because the customer is responsible for creating and updating the container images for each of these components. Container deployment using the managed container services option requires the customer to manage the workload. The CSP manages the container environment as well as the underlying infrastructure.

PaaS Deployment Options

The cloud deployment technologies that are provided through IaaS deployment options are also provided through PaaS deployment options. In addition to the infrastructure provided, PaaS deployment options also provide the operating system, middleware, runtime, and other infrastructural components that also need to be managed with IaaS. PaaS, thus, offers a fully managed solution for developers to quicky deploy and launch their applications, significantly reducing infrastructure and middleware overhead for developers. This allows developers to focus on developing and deploying their applications while the burden of managing the infrastructure, back-end services, and any other system administrative services falls on the cloud service provider.

FaaS Deployment Options

Function as a Service (FaaS) and serverless computing are mostly synonymous. In the cloud-native development model of **serverless computing**, developers can build and run applications but are not responsible for provisioning, maintaining, and scaling the server infrastructure as this is outsourced to the CSP. The customer is thus focused exclusively on the business logic of the application. As shown in Figure 12.35, navigating up the *y*-axis toward serverless computing customers focuses more on the business logic and writing the application code abstracting from the underlying infrastructure. Navigating the *x*-axis toward serverless computing decreases stack implementation and customers have less control of how the application is deployed.

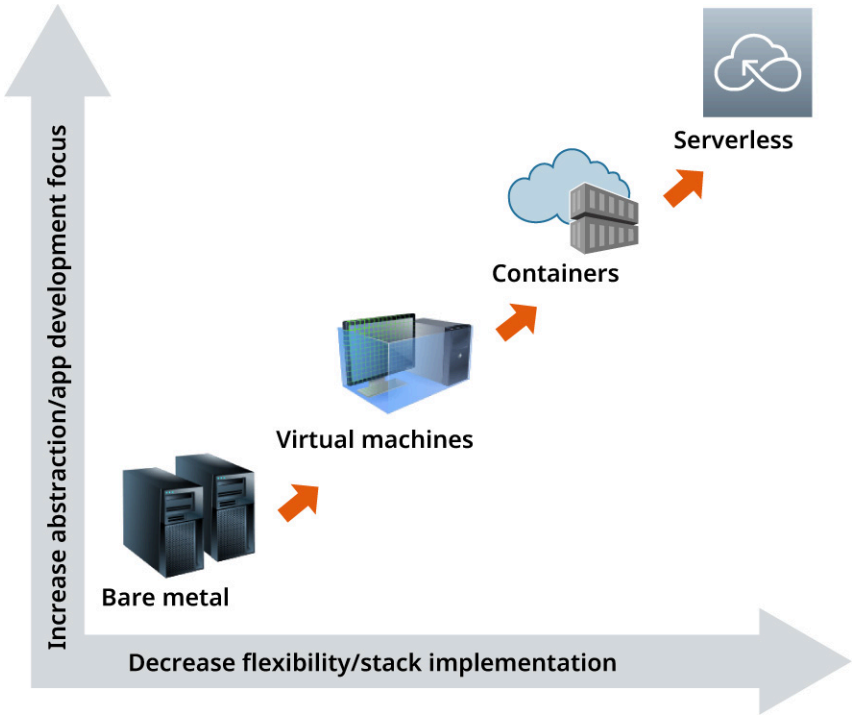


Figure 12.35 Serverless computing provides the highest level of abstraction from the infrastructure with a focus on application development. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Serverless computing offerings typically fall into two groups, Backend as a Service (BaaS) and Function as a Service (FaaS). **Backend as a Service (BaaS)** is any third-party service that can be integrated with an application where the BaaS code does not need to be managed, and there are no servers. BaaS gives developers access to a variety of third-party services and apps. For instance, a cloud provider may offer authentication services, extra encryption, cloud-accessible databases, and high-fidelity usage data. With BaaS, serverless functions are usually called through APIs.

More commonly, when developers refer to serverless, they are talking about a FaaS service model. **Function as a Service (FaaS)** is a compute platform for serverless where developers can run functions that are single units of deployment of the code, which is run by events. They are typically invoked via an API gateway. An API gateway translates requests to a single endpoint and then routes it to a FaaS function. FaaS/serverless functions have the following basic characteristics:

- An **event-driven architecture**, provided by the cloud service provider, is an architecture where functions are invoked by an event. The function may initiate other events, which could invoke other functions.
- Functions that are a single unit of deployment of the code and configuration run in a managed environment.
- Functions are executed in the cloud and run in ephemeral stateless containers.

LINK TO LEARNING

Read this article about serverless architectures (https://openstax.org/r/76serverless) for more information.

With FaaS/serverless computing, the cloud service provider manages the infrastructure in its entirety. The FaaS/serverless underlying infrastructure is automatically scaled as demand increases. FaaS/serverless solutions have a faster time to market. Customers are not responsible for the management of any of the underlying infrastructure, making it easier to build and deploy FaaS solutions faster and bring them to market. FaaS/serverless is a polyglot environment. FaaS functions can be written in any language if it is supported by

the cloud service providers. FaaS/serverless solutions are inherently highly available. FaaS/serverless solutions can be deployed across multiple availability zones in different geographical regions. Cloud service providers manage fault tolerance and deployment across the available regions making the FaaS/serverless solution highly available.

Serverless offerings are usually metered on-demand through an event-driven execution model. Customers pay for execution only. When a serverless function is idle, there is no cost. Customers must, however, be aware that providers place limits on the resources available to functions, processing time, and concurrency of event processing. These constraints must be assessed to ensure the user experience is not degraded.

LINK TO LEARNING

Learn more about use cases for using a serverless architecture (https://openstax.org/r/76usecasesserv) in three specific application areas.

PaaS and FaaS Comparison

PaaS provides a platform that allows customers to provision and scale the infrastructure needed as well as develop, run, and manage applications without the complexity of managing the underlying infrastructure required to run the application. FaaS goes a step further in that it provides an environment where developers can focus on running and managing code without having to provision resources or scaling the underlying infrastructure. In summary, both FaaS and PaaS provide an infrastructure managed by the cloud service provider. Both provide scalability options. Both deliver high availability from remote infrastructure managed automatically by the cloud service provider at scale. FaaS/serverless and PaaS, on the surface, appear to be very similar in their implementation and use from a development point of view. In reality, these two cloud deployment services are quite different. Scalability, pricing, start-up time, tooling, and the ability to deploy differ between the two platforms.

Serverless applications offer true auto-scaling capabilities to customers without any additional configurations. PaaS provides scalability options that are more advantageous compared to deployments on bare metal servers. However, developers still must consider how to construct and scale a PaaS platform. They must forecast resource capacity and configure PaaS-hosted applications to scale up and down based on demand. Serverless cost structure is based on usage, and there are no fixed monthly charges for services. With FaaS, customers pay per event-based invocation of the function only. With PaaS, pricing models differ depending on the PaaS provider, and are generally associated with the use of compute, storage, and network resources costs. This means, for PaaS, costs accrue during idle time. In some cases, serverless can be a more cost-effective option for software deployments.

FaaS functions execute in response to events, offering rapid start-up times crucial for event-based processing. Conversely, PaaS applications lack event-triggered start-up capabilities and are tolerant of slower start-up times due to their long-lived nature. FaaS abstracts the application layer, allowing developers to focus solely on function code while cloud service providers manage other aspects. This abstraction, however, reduces flexibility, as developers must adhere to provider-specific frameworks and language limitations. PaaS, on the other hand, offers developers greater control over the development environment, including the choice of programming languages and frameworks.

There are at least three application areas where it makes sense to go serverless:

1. Unpredictably fluctuating workloads or frequent low usage ideal for the pricing model of FaaS/serverless solutions.
2. Processing large, distributed data sources because of the feasibility of auto-scaling of FaaS/serverless environments where computing performance can be dramatically boosted.

3. Building highly modular solutions potentially incorporating external services that can also exploit the pricing model of FaaS/serverless solutions.

More generally, serverless architecture is ideal for asynchronous, stateless applications that can be started instantaneously. Likewise, serverless is a good fit for use cases that see infrequent, unpredictable surges in demand, that are high-volume, and with parallel workloads. An example of an infrequent in-demand task would be a batch processing of incoming image files. This task can be high-volume if a large batch of images to be processed arrives all at once. Serverless is also a good fit for use cases that involve incoming data stream processing for implementing applications such as chat bots, scheduled tasks, or business logic. Some other common serverless use cases are back-end APIs and web apps, business process automation, serverless websites, and integration across multiple systems.

CSPs offer several serverless services. Understanding the minute differences between these services can be overwhelming, making it challenging to select the set of services to develop an optimal serverless architecture. It is expected to invest up-front time to get familiar with these services to understand the runtime expectations and resource consumption associated with a prospective serverless solution. It is also challenging to maintain a serverless architecture in a continuous deployment procedure.

There are many cloud deployment technology options available to customers, which will continue to evolve. The vast possibilities available provide support for a wide host of services. The sections that follow provide insight into common use cases that are associated with the use of these deployment technology options.

INDUSTRY SPOTLIGHT

CaaS, PaaS, or FaaS?

Deployment technologies are broadly used in the industry today to deploy cloud-native applications. Containers/CaaS, PaaS, and FaaS/serverless deployment technologies are used by Netflix, Uber, and WeChat to deploy cloud-native systems that consist of many independent services. Knowing these systems, can you elaborate on the suitability of Containers/CaaS as compared to PaaS or FaaS/serverless for these particular systems?

Cloud Deployment Technology Use Cases and Implementation

There are various types of situations that prompt the use of cloud deployment technologies. The corresponding use cases and their implementation are covered in the following sections.

Cloud Deployment Technology Use Cases

The most common types of deployments are for enterprise data centers, which will be the focus of these use cases. Typically, multiple cloud deployment technologies are used together to provide the services and deployment requirements to satisfy customer needs. As customers seek to deploy large-scale enterprise applications, they must assess the required infrastructure, middleware, runtime dependencies, and constraints to choose the most appropriate cloud deployment technologies. For cloud-native applications, cloud deployment technologies such as CaaS, PaaS, and FaaS are the most popular.

Bare Metal Deployment Technology

With bare metal servers, there is no virtualization and no hypervisor overhead, thus their resources (CPU, memory, and I/O) are dedicated to the application deployed on it. These resources, along with the hardware, are highly customizable and can be tuned up for maximum performance, making bare metal servers a viable option to support resource-intensive workloads such HPC. This also helps with latency-sensitive applications (e.g., streaming services) and large databases that consume significant resources. A combination of cloud deployment technologies can be used to satisfy the various requirements of a solution architecture. Bare

metal servers can be used for the resource-intensive components, whereas VMs, containers, and other cloud deployment technologies deliver application services. Bare metal servers are also an option for legacy applications that may not be virtualized or not well suited for a virtualized environment.

Bare metal servers are also a good fit for security-oriented applications because additional security measures can be easily integrated. Bare metal servers are the best fit for solutions that must comply with regulations or contract agreements that require single-tenant hosting.

VMs Deployment Technology

Legacy applications that can be virtualized but have known and manageable behavioral issues are well suited for cloud deployment using VMs. This also applies to legacy applications that are not well documented but there is basic knowledge of how they run. VMs are a viable option for monolithic and tiered applications that are not mature enough to be migrated to a microservices architecture. Such applications could be migrated to the cloud as cloud-based monoliths.

VMs can also be used to provide enhanced container security, although with additional overhead. The use of a VM can provide an additional layer of isolation when deploying containers eliminating the use of a shared kernel model.

Containers/CaaS Deployment Technology

Containers enable scaling, replicating, and choreographing services in a microservices architecture. Containers are also lightweight, portable, and platform independent. The ability to start containers in a fraction of the time of VMs is critical. With these advantages, cloud customers have developed a plethora of microservices ranging from user interface (UI) front ends to back-end microservices. A microservices architecture is a good use case for using containers.

Legacy applications that depend on out-of-support operating systems or other out-of-support dependencies are candidates for containers. Application dependencies can be containerized to enable execution. However, there are limitations to this approach, and it can be very challenging to get such applications up and running in a container. The application dependencies must be assessed to ensure containerization is a viable option. Legacy applications that can be modernized and migrated to the cloud are also good candidates for containers.

Platform flexibility is the primary use case for CaaS. Containers can consistently run anywhere without the limitations of middleware and platform tools offered by PaaS platforms. CaaS provides the customers with a container-based environment they can control (security, scalability, and availability) without significant effort.

Unikernels Deployment Technology

The need to optimize VMs is the most common use case for unikernels at this time. Unikernels have smaller footprints, in part, because they are built without unnecessary binaries and libraries. Because they have less overhead, unikernels can start up very quickly, faster than VMs and containers. With the smaller footprint and speed comes improve security when it comes to unikernels. Because unikernels are built to include only the libraries needed to run the application, this reduces the cybersecurity attack surface, thereby making them more secure compared to the other cloud deployment technologies.

Mutable server infrastructures are where servers are continually updated and modified in place even after they are deployed. Immutable infrastructures are where servers are never modified after they are deployed. The main difference between mutable and immutable infrastructures is the policy used that determines whether components of the environment are designed to be changed after deployment. Customers may need or expect an immutable infrastructure to preserve the integrity of an environment. Although VMs are not permitted in an immutable environment, unikernels, by design, protect against such modifications because any updates or changes to any components are built from an image and provisioned to replace the old ones. Unikernels, thus, are a good choice for the need of an immutable infrastructure.

PaaS Deployment Technology

PaaS is used to deploy modern, cloud-native application architectures. PaaS provides a complete cloud platform that includes hardware, infrastructure, middleware, and runtime environment. It is most often delivered using a container-based infrastructure, and therefore supports the same use cases as containers.

The most prevalent use case for PaaS is speed to market. PaaS reduces or eliminates the burden of deploying and managing infrastructure, enabling cloud customers to focus instead on customer needs and development of the applications. PaaS can be more affordable, especially for software development houses, which often do not have the budget to hire personnel to build and manage infrastructure, as it offers access to a wider range of services up and down the application stack. PaaS also provides a cost-effective way for customers to scale up and down resources as needed. PaaS also provides a development and CI/CD environment to quickly develop, integrate, build, and deploy applications.

FaaS Deployment Technology

The use case for FaaS/serverless is limited to microservices-based applications, more specifically applications that are short-lived and event-based. Examples of services include data/stream and image processing, reservation handling, and IoT data processing. Other use cases include high-volume and parallel workloads and data aggregation tasks.

FaaS/serverless is used to reduce the customer's responsibility to manage middleware dependencies, runtime considerations, and the underlying infrastructure. The FaaS/serverless platform takes care of this, leaving the customer with the responsibility to write the function code. The result is increased velocity in delivering highly available, technology capability, and satisfying business demand.

Cloud Deployment Technology Implementation

There are several tools cloud customers must consider when implementing services using the cloud deployment technologies discussed earlier. Cloud customers who adopt automation will increase deployment speeds while lowering costs. With the growth of cloud environments, customers must manage increasingly complex environments. There are several tools that can be used to automate tasks, such as server provisioning and configuration management. Automating these tasks improves the efficiency of DevOps teams and speeds up the deployment of services because they reduce the volume of manual tasks required to manage cloud resources and infrastructure. Tasks that must be automated in the case of each cloud deployment technology include the IaaS tasks of orchestration and provisioning, PaaS tasks include source code management and CI/CD pipelines, and FaaS/serverless tasks include orchestrating the deployment of functions and their dependencies.

Automation tools help customers reduce operational costs and minimize errors. Automation tools can be used to configure and install containers, VMs, or other systems; provision and deprovision resources for auto-scaling; and allocate resources to optimize workload performance. There are several tools that are available to assist with automation of development, delivery, and management of applications, workloads, and infrastructure. Tools such as Ansible, Pulumi, or Terraform can help with automating orchestration. Tools such as Ansible, Chef, Puppet, or Salt can help with automating configuration management. Tools such as BitBucket, GitHub, or GitLab can help with automating source control management. Tools such as Broccoli, CircleCI, Codeship, or Maven can help with automating builds. Tools such as Bamboo, CircleCI, Codeship, and Jenkins can help with automating continuous integration. For automating continuous deployment, additional options include Go, Julu, or Octopus Deploy.

GLOBAL ISSUES IN TECHNOLOGY

DevOps Across the Globe

While cloud-native applications are deployed all over the world, they involve the use of DevOps methods and tools that are mostly developed in the United States or Europe. For example, DevOps refers to terms like CI/CD, which relate to terms (i.e., continuous integration/continuous deployment) and tools (e.g., Jenkins, GitLab, Docker) that are respectively used and implemented by companies in the United States. Therefore, it is not certain how well those methods and tools work in other cultural contexts. What is your opinion regarding these concerns?

IaaS Cloud Deployment Implementation

When adopting IaaS, IaC can be used to automate infrastructure management provisioning by using code instead of manually. This includes managing servers, managing operating system installations and updates, kernel modifications, and the management of storage and other infrastructure components needed for the development and deployment of applications.

Orchestration tools (e.g., Kubernetes) can be used on any of the cloud deployment technologies discussed to automate services such as allocating resources and scaling applications as needed. Configuration management tools (e.g., Ansible) can also be used on any of the cloud deployment technologies to automate the process of configuring and managing changes to these resources. Both orchestration and configuration management tools provide support for integrating with IaaS services provided by the major cloud service providers (e.g., Amazon Web Services, Google Cloud Platform, IBM Cloud, and Microsoft Azure).

LINK TO LEARNING

Learn more about infrastructure as a code (https://openstax.org/r/76infracode) as well as design principles for IaC and design patterns for infrastructure deployment stacks.

PaaS Cloud Deployment Implementation

As mentioned, PaaS also provides services for managing the operating system, middleware, and runtime environments for applications. As such, available PaaS tools are used for automating tasks, such as streamlining code integration between systems as well as managing application build and deployment processes.

A tool discussed earlier, Tanzu, provides a major improvement over traditional PaaS deployment options by enabling deployment on any cloud that provides a container orchestration mechanism. It is equivalent to an application server for the cloud that can deploy cloud-based and cloud-native application workloads on any cloud.

FaaS Deployment Technology Implementation

When adopting FaaS, applications are deployed on server infrastructure in response to events without customers having to manage the underlying infrastructure. Although the managing of the infrastructure is not the customer's concern, there are setup requirements customers must deal with, including setting up the serverless runtime environment, deploying serverless functions, cloud service dependencies (e.g., storage), and the required amount of memory needed. There are tools (e.g., Ansible) that are available to help deploy these cloud services.

Cloud management platforms (CMPs) are used to simplify the deployment and management of cloud services

across various cloud service providers. Customers manage several large-scale applications in very complex environments using services from multiple cloud service providers. CMPs can be helpful in simplifying the deployment and management of all these services. Understanding the common use cases as well as the available cloud deployment technologies available help customers to select the appropriate solutions to support their needs.

Selection of Cloud Deployment Technologies

When selecting the most appropriate cloud deployment technologies, customers should both strategically and tactically consider these choices. Strategically, customers should consider their business needs and select the cloud deployment technologies that are relevant to meet these needs. Tactically, customers should consider the application requirements and relevant cloud services needed to support those applications. In summary, the steps to select the appropriate cloud deployment technologies include:

- Assess how well the cloud deployment technology options meet the business needs.
- Understand infrastructure and workload/service requirements.
- Select cloud deployment technologies that satisfy the application requirements and provide the services needed.

Fit to Business Needs

With so many cloud deployment technology options, customers should regularly assess these options with the following considerations in mind:

- Cost: Understand the different cost models associated with the various cloud deployment technologies.
- Architectural fit: Understand the application deployment needs and how the different cloud deployment technologies support those needs.
- Performance: Understand the application performance requirements (e.g., workloads performance requirements, bandwidth requirements, latency requirements) and how the different cloud deployment technologies satisfy these requirements.
- Compliance: Identify any specific regulatory or contractual requirements that impose limitations that can impact the choices of cloud deployment technologies.
- Elasticity requirements: Identify the level of elasticity needed to satisfy the possible need to grow or shrink infrastructure resources dynamically as needed.
- Control requirements: Determine the level of control required to manage the various cloud deployment technologies and at what layers of the infrastructure.
- Cloud service provider lock-in: For customers who manage diverse, complex environments across multiple cloud service providers, determine the level of portability needed to enable the flexibility to change providers to optimize services utilized.

Although, generally, there is no formal process for selecting cloud deployment technologies, customers can benefit from a more formalized process as it allows customers to implement the appropriate orchestration and management tools and establish processes needed to deliver effective and efficient services.

Understanding Workload/Service Requirements

When selecting cloud deployment technologies, requirements (e.g., latency limitations, industry-dependent data protection controls) related to the types of workloads or services should also be considered.

Selecting Application/Service Cloud Deployment Technology

As the strategic requirements and workload/service constraints discussed previously are defined, customers can then evaluate the cloud deployment technologies that are most appropriate to meet their business needs. The cloud deployment technologies selected must adhere to the service levels required to satisfy business needs. These options may change over time as the range of available cloud deployment technology options continues to evolve. Prudent customers should proactively periodically evaluate these cloud deployment

technologies to continuously meet business needs.

Applications Composed of Multiple Cloud Services

A **cloud mashup** is a technique for seamlessly combining multiple cloud applications from several sources into a single integrated solution. Cloud mashups can be realized in many ways covering different scopes depending on the purpose of the mashup. For example, PaaS services on public clouds can be used to create innovative cloud-based and cloud-native applications as cloud mashups, which enable corporations to undergo digital transformations to provide competitive solutions while differentiating themselves. An example of a cloud mashup that provides a driving route recommendation service is the mashup of AWS EC2, Facebook's authentication and authorization services, and Google's MapReduce services.

The selection of cloud deployment technologies is an important factor in the overall quality of cloud mashups measured by quality of service and quality of experience assurances as these metrics specify the desired performance requirements of the cloud mashup. Regarding the previously mentioned driving route example, the recommended driving route and the speed at which the route is calculated are important factors to determine the quality of this service. In addition, the discovery of the best-choice web services and the streamlining of services into a mashup are also impacted by the selection of cloud deployment technologies. The increasing demand for innovative cloud services have led to cloud service providers competing to provide mashup service discovery, automated composition techniques, and reusable cloud component's APIs as services that support the ability to quickly assemble mashups.

TECHNOLOGY IN EVERYDAY LIFE

Real Life and Technology

How does the use of cloud deployment technologies help people in everyday life? As was mentioned, cloud deployment technologies can be used to create cloud mashups. One example discussed is a driving route recommendation service (the mashup of AWS EC2, Facebook's authentication and authorization services, and Google's MapReduce services). Many people depend on a driving route recommendation service in GPS systems today. It was also mentioned that the selection of cloud deployment technologies directly impacts the overall quality of the cloud mashup as the accuracy of the recommended routes and the speed at which routes are calculated are important features of such a service. Can you think of other ways cloud deployment technologies help people in everyday life? Provide a couple of illustrative scenarios to explain your opinion.

Key to Success Summary

There are lots of options when it comes to choosing the most appropriate cloud deployment technologies, and these options will continue to grow. Customers should understand the options that are available and how these cloud deployment technologies can best fit the requirements of deploying their applications to the cloud. To help select the most appropriate options, customers should consider: (1) understand how the currently available cloud deployment technologies and services work and fit in the delivery of cloud services, (2) identify the tools available to automate and support the delivery of these cloud services, and (3) assess the technical factors (e.g., application architecture, data implications, runtime dependencies) of these cloud deployment technologies and select the technologies that provide the "best fit" to their business needs. These steps should be repeated periodically to reassess the available cloud deployment technologies as these technologies evolve and more are added. Customers should keep in mind that the benefits of researching the most appropriate selection of cloud deployment technologies to fit business needs up front can significantly outweigh the costs of taking an ad hoc approach of combining technologies that may end up providing a suboptimal user experience.

THINK IT THROUGH

Which Solution to Select?

Given the fact that many deployment technologies are available today, is there a process that facilitates the selection of these technologies to deploy cloud-native applications?

12.3 Example PaaS and FaaS Deployments of Cloud-Native Applications

Learning Objectives

By the end of this section, you will be able to:

- Understand how to deploy a cloud-native application on a PaaS platform
- Understand how to deploy a cloud-native application using VMWare Tanzu
- Understand how to deploy FaaS functions on a serverless platform

This module focuses on building sample applications that illustrate the steps taken to deploy sample applications using various cloud deployment technologies. The first section focuses on how to build a sample cloud-native application on a PaaS platform. The sample application provided illustrates the use of microservices, Docker containers, and Kubernetes orchestration. The second section focuses on how to set up a suite of products that are used to manage Kubernetes clusters and monitor applications that are deployed in Kubernetes clusters. Finally, the third section focuses on how to deploy FaaS functions that are parts of a distributed application on a serverless platform. The example provided illustrates the use of various metrics and performance dashboards used to monitor a distributed application. When working through these examples, keep in mind that they are based off tutorials that are made available by the cloud service providers. As the technologies used in these tutorials evolve, the tutorials may change. As a result, there may be differences in the configuration options in the cloud service provider consoles or some of the steps may have changed. Regardless, the underlying goals of these examples should remain achievable. These examples also require subscriptions to AWS and Microsoft Azure. All cloud services providers provide free-trial credits. These examples were completed without exceeding the free-trial credit and using as many free-tier services as possible.

PaaS Deployment of a Sample Cloud-Native Application

The example in this section illustrates PaaS deployment of a cloud-native application on Microsoft Azure. A subsidiary of Microsoft, **Azure** is a cloud computing platform that offers a wide range of services that allow customers to build, deploy, and manage applications and services in the cloud. The sample cloud-native application[13] includes two microservices. Both communicate with a single datastore. Each microservice is containerized and deployed in a Kubernetes environment illustrating Kubernetes orchestration. The PaaS deployment of this sample cloud-native application in Azure is illustrated in Figure 12.36.

13 Sample based off tutorials: https://learn.microsoft.com/en-us/training/modules/cloud-native-build-basic-service/ and https://learn.microsoft.com/en-us/training/modules/cloud-native-apps-orchestrate-containers/

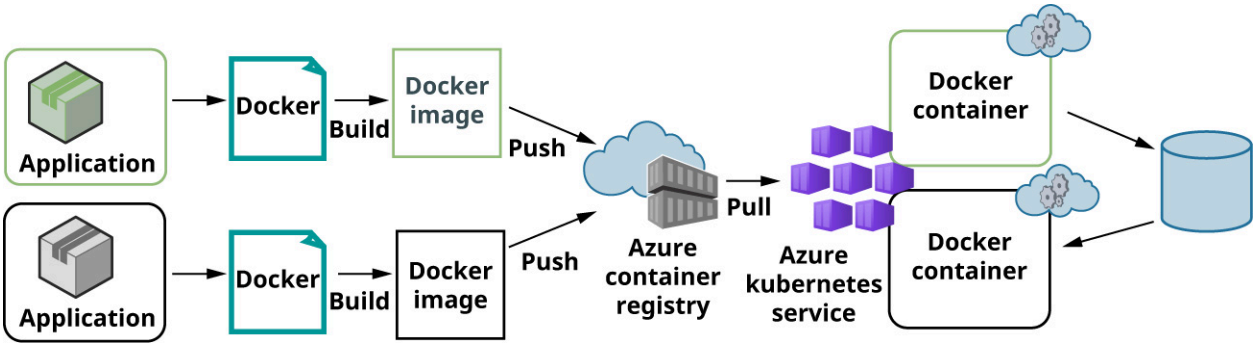


Figure 12.36 A cloud-native application is deployed in Azure. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

One microservice implements a web service in JavaScript using Node. Express.js is used to implement the REST API for the web service. **Express.js** is a back-end Node web application framework used to implement RESTful APIs. This microservice pushes data updates to a datastore via the REST API. The other microservice implements a web service using Next.js. **Next.js** is an open-source React framework used to create full-stack web applications. React is a library used to create components to render to certain environments including web and mobile applications. This microservice reads data from the same datastore.

Docker images are created for each microservice and pushed to an image registry. Azure's Container Registry Service (ACR) is used for this purpose. Each microservice is self-contained and encapsulated into Docker containers that are pulled from ACR and deployed into worker nodes in a Kubernetes cluster. Scaling the microservices is managed by Kubernetes. The Azure Kubernetes Service (AKS) is used for this purpose. Both microservices communicate with a single datastore. The datastore used is a PostgreSQL database hosted in Azure.

Prerequisites:

- Open a web browser and log into the Azure Portal. The **Azure Portal** is a web-based console that allows customers to manage their cloud services and Azure subscriptions.
- An Azure resource group. An Azure **resource group** is a container that holds related resources used in a cloud solution. In this example, the resource group *rg-nativeapps-eastus* is used.

Set Up a Postgres Database in Azure

The first step is to create a datastore that both microservices will communicate with. *Azure Database for PostgreSQL* is the resource used for this purpose. The following steps create a relational database management service (RDBMS). Once the RDBMS is created, a PostgreSQL database is created along with tables to store the data. Finally, data is inserted into the database.

Create the Resource

1. In the Azure Portal, search for *Azure Database for PostgreSQL*. Select Azure Database for PostgreSQL listed under the Marketplace section.
2. Select Azure Database for PostgreSQL Flexible server for the Resource type and click Create.
3. On the Basics tab, configure the resource attributes. Table 12.1 shows the list of settings that should be used. Any settings not included in the table should be set to the default values provided in the wizard.

Basics Tab Settings	
Subscription	Select the default subscription.
Resource group	For this example, *rg-nativeapps-eastus* is used.
Server name	Enter a unique name for the resource. For this example, *na-dbserver-flex* is used.
Data source	Select None.
Location	For this example, select the region that is used for the resource group.
Version	Select 11.
Compute + storage	Click on the Configure server link. On the Configure blade, select Basic, set the vCore value to 1 and Storage to 2 GiB, and then click Save.
Admin username	Enter a username. For this example, *Student* is used.
Password	Enter a password. For this example, *Pa55w0rd1234* is used.

Table 12.1 Configuration for the PostgreSQL Database

4. To create the resources as configured in the Table 12.1, click Review + create and then click Create. The provisioning of the database server may take several minutes. A status message appears when the deployment is complete. Click on Go to resource.

Configure Connection Security

Security policies need to be added to allow resources, including all microservices, to connect to the datastore securely. This step configures the connection security settings so that the microservices can securely connect to it. This is done on the Connection Security page for the datastore resource.

1. From the menu on the left under Settings, click Networking.
   - Enable the database server to allow connectivity from the cloud-native application deployed and running in Azure. To do this, click Yes for Allow access to Azure services. Immediately below this configuration, click + Add current client IP address (Figure 12.37).

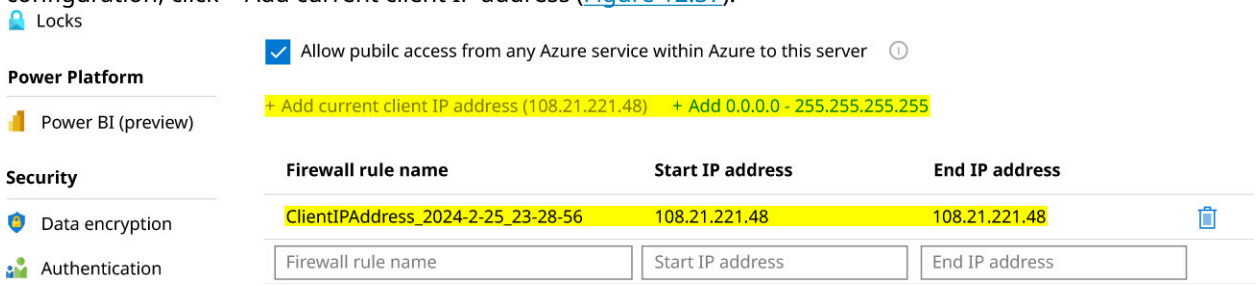


Figure 12.37 It is important to configure the connection security settings so that the microservices can securely connect to it. (Used with permission from Microsoft)

   - For this example, disable the SSL settings. If this step was missed before provisioning the database,

do the following: click Server parameters. In the search field, search for "require_secure_transport." Click Off (Figure 12.38).

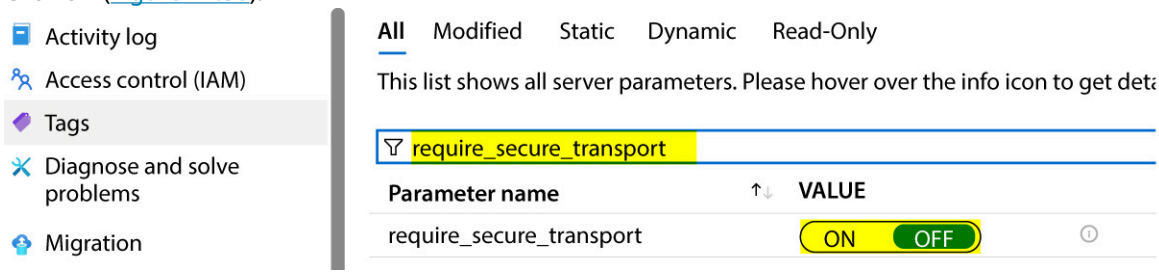


Figure 12.38 Be sure to disable the SSL settings. (Used with permission from Microsoft)

- Click Save.

2. From the menu on the left, click Overview. Make a note of the Server name and Admin username values. These values are used to connect to the database from the cloud-native application deployed and running in Azure.

Create the Database, Tables, and Initial Data

The datastore is now configured with a valid hostname and user account. The next step is to create a database and tables for the data to be stored. Once the database and tables are created, initial data is inserted.

1. Open the Azure Cloud Shell. To do this, click on the Cloud Shell icon to the right of the search bar in the Azure Portal. In the bottom frame of the browser page, the Cloud Shell will load. Click Bash, if prompted when the Cloud Shell loads. Click Create storage if prompted to complete loading the Cloud Shell.
2. In Cloud Shell, connect to the database with the following psql command. Insert the Server name and Admin username values obtained earlier for <*server_name*> and <*username*>, respectively as shown below. A postgres command prompt appears.
   ```
   psql –host=<server_name> --port=5432 –username=<user_name> --dbname=postgres
   ```
3. Create the database, create a table, and insert data that will be used in this example.
   - Run the SQL statement below to create a new PostgreSQL database. The database name used in this example is *cnainventory*.
     ```
     CREATE DATABASE cnainventory;
     ```
   - Run the command below to switch to the newly created database. This step is necessary so that the tables are created in the correct database.
     ```
     \c cnainventory
     ```
   - Run the SQL statement below to create a new table. The table created for this example is *inventory*. It contains four fields: id, which is the primary key, name, quantity, and date.
     ```
     CREATE TABLE inventory(
         id serial PRIMARY KEY,
         name VARCHAR(50),
         quantity INTEGER,
         date DATE NOT NULL DEFAULT NOW()::date
     );
     ```
   - Confirm the *inventory* table was created using the following command.
     ```
     \dt
     ```
   - Insert data into the *inventory* table with the SQL statements below.
     ```
     INSERT INTO inventory (id, name, quantity) VALUES (1,'yogurt', 200);
     INSERT INTO inventory (id, name, quantity) VALUES (2,'milk', 100);
     ```
   - Confirm the data was successfully inserted with the following command:

```
SELECT *FROM inventory;
```

- The output lists the data records.
- Type \q to disconnect from the database.

Create and Deploy a Cloud-Native Application

Now that the datastore for the cloud-native application has been successfully created and configured, the next step is to create each of the two microservices of the cloud-native application. As previously mentioned, the cloud-native application consists of two microservices. One of the microservices is implemented using Node/Express.js. This microservice serves as a back-end service. The second microservice is implemented using Next.js and serves as a front-end web service. Although these microservices do not directly communicate with each other, both communicate with the datastore.

Create the Back-End Service

The first microservice created is the back-end service. This service exposes a set of functions that can receive requests via a REST API that inserts inventory data into the datastore.

1. Open the Azure Cloud Shell. To do this, click on the Cloud Shell icon to the right of the search bar in the Azure Portal. In the bottom frame of the browser page, the Cloud Shell will load. Click Bash, if prompted when the Cloud Shell loads. Click Create storage if prompted to complete loading the Cloud Shell.
2. Create a directory for the application and navigate into it with the following command.
   ```
   mkdir -p can-node-express && cd can-node-express
   ```
3. Use the command to initialize a Node project. A package.json file, among other files, is generated for the Node project. The package.json file is updated to include dependencies for the Node/Express.js back-end service.
   ```
   npm init -y
   ```
4. Express.js is used to build the REST API for the back-end service. Install Express.js with the following command. Confirm the package.json file is updated listing express as a dependency.
   ```
   npm install express
   ```
5. Create a new file named *index.js* with the command code index.js and add the code shown below. To save the file, type CTRL+S. Close the file by typing CTRL+Q. The code creates an Express application server that listens on port 8080. It accepts client requests sent in JSON format.
   ```
   const express = require('express')
   const port = process.env.PORT || 8080
   const app = express()
   app.use(express.json())
   app.listen(port, () => console.log('Sample app is listening on port ${port}!'))
   ```

Connect the Cloud-Native Application to the Database

Now that the back-end service has been successfully created, the next step is to add code to the Express.js application that allows it to connect to the datastore. The **object-relational mapping (ORM)** technique converts a data object in the Express.js code to a table in the PostgreSQL relational database. Sequelize is used for this purpose.

1. In Azure Cloud Shell, run the command below to install the Sequelize package.
   ```
   npm i sequelize pg pg-hstore
   ```
2. Edit the index.js file to add code that allows the Express.js application to connect to the *cnainventory* database. Insert the code below. Substitute the Server name value for <*server_name*> (appears twice). This code provides the connection hostname and user account to the datastore so that the back-end service can connect to it.
   ```
   const Sequelize = require('sequelize')
   const sequelize = new
   ```

```
Sequelize('postgres://Student%40<server_name>:Pa55w0rd1234@<server_name>.postgres.database.azu
cnainventory)
sequelize
.authenticate()
.then(() => {
  console.log('Connection has been established successfully.');
})
.catch(err => {
  console.error('Unable to connect to the database:', err);
});
```

3. To use Sequelize in the Express.js application, add the following code to the file the index.js. This is the code that does the mapping between data objects in the Express.js code to data records in the database table. The variable Inventory is declared to define the mapping between the Express.js code and the *inventory* table. Notice how this definition contains the exact same fields that were declared in the *inventory* table in the *cnainventory* PostgreSQL database when it was created.

```
const Inventory = sequelize.define('inventory', {
  id: { type: Sequelize.INTEGER, allowNull: false, primaryKey: true },
  name: { type: Sequelize.STRING, allowNull: false },
  quantity: { type: Sequelize.INTEGER },
  date: { type: Sequelize.DATEONLY, defaultValue: Sequelize.NOW }
}, {
  freezeTableName: true,
  timestamps: false
});
```

Create the Express.js REST API Endpoints

Now that the Express.js application is configured to access the PostgreSQL database, the next step is to create the REST API to accept client requests. These REST routes call functions that perform read and write operations on the PostgreSQL database. Two Express.js routes are added in the code. The first route performs a read from the database in response to receiving a GET HTTP request. The second route performs a write to the database in response to receiving a POST HTTP request.

1. Edit the index.js file and add the code shown. The code adds a route that accepts HTTP GET requests to fetch an inventory record. The ID for the record is included in the request, and the ID, name, quantity, and date fields for the inventory record are returned.

```
app.get('/inventory/:id', async(req, res)=> {
  const id=req.params.id
  try {
const inventory = await Inventory.findAll({
  attributes: ['id', 'name', 'quantity', 'date'],
  where: {
    id: id
  } })
  res.json({ inventory })
} catch(error) {
  console.error(error)
}})
```

2. Add the second route by adding the following code to the index.js file. The code adds a route that accepts HTTP POST requests to create a new inventory record. Values for the record are included in the HTTP request body with exception for the date, which is calculated from the current date.

```
app.post('/inventory', async (req, res) => {
try {
  const newItem = new Inventory(req.body)
  await newItem.save()
  res.json({ inventory: newItem })
} catch(error) {
  Console.error(error)
}})
```

Create the Front-End Component

The second microservice created is the front-end web service. This web service provides a web-based user interface to fetch inventory data.

1. In the Azure Cloud Shell, use this command to create a Next.js application.
   `npx create-next-app`
2. Answer the prompts. It is important to select No for the App Router prompt. Note that the project name is cna-next. This is the root directory for the Next.js application.
3. Navigate into the cna-next directory.
4. Recall in the back end, Express.js application **Sequelize** is used as the ORM to convert data objects in the Express.js code to data records in the *inventory* table in the database. **Prisma** is a Node ORM used to map data objects to tables in a relational database.
   - Install the prisma and prisma-client packages with the following commands.
     `npm install prisma (npm install prisma -save-dev)`
     `npm install @prisma/client`
   - Configure the Next.js application to use Prisma by running the command. This creates the prisma/ subdirectory and generates the schema.prisma configuration file inside it. This command also generates a dotenv (.env) file in the root directory of the project.
     `npx prisma init`
   - In the prisma/ directory, edit the generated schema.prisma file and add the content shown in Figure 12.39. This adds the data model for the *inventory* table.

```
// This is your Prisma schema file,
// learn more about it in the docs: https://pris.ly/d/prisma-schema

generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model inventory {
  id        Int @id
  name      String
  quantity  Int
  date      DateTime
}
```

Figure 12.39 This content adds the data model for the *inventory* table. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

   - Notice how the schema.prisma is configured to read the data source database URL from a dotenv (.env) file. In the cna-next/ directory, edit the generated dotenv (.env) file and change the database connection string as shown in the code snippet below. Replace USER_NAME with Admin name,

PASSWORD with Password, and SERVER_NAME with Server name for the *cnainventory* PostgreSQL database.

```
DATABASE_URL="postgresql://USER_NAME%40SERVER_NAME:PASSWORD@SERVER_NAME.postgres.database.az
cnainventory"
```

- Make a copy of the .env file and name it .env.local as this is the file that will be copied into the Docker container and used by Prisma.
- To use Prisma in the Next.js application, the Prisma Client must be configured. The Prisma Client serves as a query builder tailored to the application data. Query builders are part of the ORM that generate the SQL queries used to perform the database operations for the application. To do this, run the command:

```
npx prisma generate
```

- Add the Prisma Client code to the Next.js application. To do this, create the lib/ subdirectory and navigate into it.
- Inside the lib/ directory, create the file prisma.tsx and add the following code.

```
import { PrismaClient } from '@prisma/client'

const globalForPrisma = global as unknown as {
prisma: PrismaClient | undefined
}

export const prisma =
globalForPrisma prisma ??
new PrismaClient({
log: ['query'],
})

if(process.env.NODE_ENV !=='production') globalForPrisma.prisma = prisma
```

5. Now that the Next.js application is configured to map to the *inventory* table, the next step is to implement the web service code. The web service code consists of React components that render in a browser. The React component InventoryProps is an array of data records once fetched from the database. The React component Inventory implements how the data is displayed as a web page in the browser. The React component Layout adds additional navigation to the web page.
   - In the cna-next/ directory:
     - Create and navigate to a directory named components/ and add the two code files that follow into it (confirm the components/ directory is at the same level as the pages/ directory that was generated):
     - Create the file Inventory.tsx and add the code that follows.

```
import React from "react";

export type InventoryProps = {
  id: string;
  name: string;
  quantity: string;
  date: string;
};

constInventory: React.FC<{ inventoryrec: InventoryProps }>=({
inventoryrec})=> {return(
  <div
```

```
  className="flex bg-white shadow-lg rounded-lg mx-2 md:mx-auto mb-5 max-
w-2xl"
  >
  <divclassName="flex items-start px-4 py-3">
    <div className="">
      <div className="inline items-center justify-between">
        <p className="text-gray-700 text-sm">
        <strong>ID: {inventoryrec.id}</strong> Name: {inventoryrec.name}
(quantity: {inventoryrec.quantity})
        </p>
        <small className="text-red-700 text-sm">
        Date: {inventoryrec.date.toString().substring(0,10)}
        </small>
      </div>
    </div>
  </div>
</div>
</div>
);
};

export default Inventory;
```

- Create the file Layout.tsx and add the following code.

```
import React,{ ReactNode } from "react";
import Head from "next/head";

type Props = {
  children: ReactNode;
};

const Layout: React.FC<Props> = (props) => (
  <div>
    <div className="w-full text-center bg-red-800 flex flex-wrap items-
center">
      <div className="text-3xl w-1/2 text-white mx-2 md:mx-auto py-5">
        Inventory Data
      </div>
  </div>
  <div className="layout">{props.children}</div>
  <style jsx global>{`
    html {
      box-sizing: border-box;
    }
    *,
    *:before,
    *:after {
      box-sizing: inherit;
    }
    body {
      margin: 0;
```

```
          padding: 0;
          font-size: 16px;
          font-family: -apple-system,BlinkMacSystemFont,"Segoe UI",Roboto,
            Helvetica,Arial,sans-serif,"Apple Color Emoji","Segoe UI Emoji",
            "Segoe UI Symbol";
          background: rgba(0,0,0,0.05);
        }
        input,
        textarea {
          font-size: 16px;
        }
        button {
          cursor: pointer;
        }
      `}</style>
      <style jsx>{`
        .layout {
          padding: 0 2rem;
        }
      `}</style>
      </div>
    );

    export default Layout;
```

- Edit the index.tsx file and replace the default code with the following code.

```
declare global {
  namespace NodeJS {
    interface Global {
      prisma: any;
    }
  }
}

import { prisma } from '../lib/prisma';
import Inventory,{ InventoryProps } from "../components/Inventory";
import Layout from "../components/Layout"

export const getServerSideProps = async () => {
  const inventoryrecs = await prisma.inventory.findMany({
  })
  return { props: { inventoryrecs: JSON.parse(JSON.stringify(inventoryrecs)) }
}
}

type Props = {
  inventoryrecs: InventoryProps[]
}

// index.tsx
```

```
const InventoryFeed: React.FC<Props> = (props) => {
  return (
    <Layout>
      <div className="page">
        <br/>
        <main>
          {props.inventoryrecs.map((inventoryrec) => (
            <div key={inventoryrec.id} className="post">
              <Inventory inventoryrec={inventoryrec} />
            </div>
          ))}
        </main>
      </div>
      <style jsx>{`
        .post:hover {
          box-shadow: 1px 1px 3px #aaa;
        }
        .post + .post {
          margin-top: 2rem;
        }
      `}</style>
    </Layout>
  )
}

export default InventoryFeed
```

Build and Store Microservices Images in an Azure Container Registry

Now that the two microservices for the cloud-native application have been successfully created, the next step is to create an Azure Container Registry (ACR) to store Docker images for these microservices. Each microservice of the cloud-native application is containerized. Their images are pulled from the ACR and deployed in a Kubernetes environment hosted in the cloud.

Create the Azure Container Registry

1. In the Azure Portal, on the home page, click on Create a resource. Click Container Registry.
2. On the Basics tab, configure the resource attributes. Table 12.2 shows the list of settings to configure on the Basics tab. Any settings not included should be set to the default values provided in the wizard.

Basics Tab Settings	
Subscription	Select the default subscription.
Resource group	For this example, *rg-nativeapps-eastus* is used.
Registry name	Enter a unique name for the resource. For this example, *ncaregistryflex* is used.
Location	For this example, select the region that is used for the resource group.
SKU	Select Standard.

Table 12.2 Configuration for the Azure Container Registry

3. Click Review + create. Make a note of the Registry name and Resource Group as these will be needed in a later step. Click Create. When the provisioning is completed, a status message appears.
4. Click on Go to resource. Make a note of the registry name that was provided in the create wizard. In this example, the registry name is *ncaregistryflex*.
5. Generate access keys for the container registry, which will be needed later. For this example, the container registry used is *ncaregistryflex*. In the Azure Portal, navigate to the container registry resource. From the menu on the left, under Settings, click Access keys. Enable Admin user.

Build the Docker Images

Now that the container registry has been successfully created, the next step is to build Docker images for each microservice. These images are then pushed to the container registry.

1. Setting specific environment variables makes it easier to run the commands that follow. In the Azure Cloud Shell, run the following commands to set the required environment variables. Note the resource group name in this example is *rg-nativeapps-eastus*. The registry name in this example is *ncaregistryflex*.
```
RESOURCEGROUP={resource-group-name}
REGISTRYNAME={registry_name}
```

Containerize the Back-End Service

To containerize the back-end service, a Dockerfile must be created with a list of instructions to build the Docker image.

1. Navigate to the can-node-express/ directory. Create a Dockerfile and add the instructions below. The Dockerfile starts with a base image for Node indicated by the FROM instruction. A working directory is created and the package.json file is copied into it. The dependencies listed in the package.json file are used to install the dependent packages. Next, the source code for the Express.js application is copied. Port 8080 is exposed for the Express.js application listens on. Finally, the command to start the Express.js application server is added as the last instruction.
```
FROM node:14-alpine
# Create app directory
WORKDIR /src
# Copy package.json and package-lock.json
COPY package*.json /src/
# Install npm dependencies
ENV NODE_ENV=production
RUN npm ci --only=production
# Bundle app source
COPY ./src
EXPOSE 8080
CMD [ "node", "index.js" ]
```
2. Build the Docker image and push it to the ACR registry. The Docker image must be built locally and then pushed to the ACR registry later. This is assumed a Docker engine is installed on the local computer. To build the image, run the command below. Notice the command has a space followed by a “.” at the end. This references the current directory and must be part of the command. Wait until the Docker image build is complete.
```
docker build -t expressimage .
```
3. Test run the back-end application by running the Docker container. Run the following command.
```
docker run -d --name expressimage -p 8080:8080 expressimage:latest
```
4. Open a browser and enter the URL: http://127.0.0.1:8080/inventory/1.



(attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

5. Tag the image so that it can be pushed to the ACR registry with the command below. *Note*: run "docker images" to confirm the correct image name is used for the docker tagging. Substitute <registry_name> with the correct name of the ACR registry.
   `docker tag expressimage:latest <registry_name>.azurecr.io/expressimage:v1`

Containerize the Front-End Service

To containerize the front-end service, a Dockerfile must also be created with a list of instructions to build the Docker image.

1. Navigate to the can-next/ directory. Create the Dockerfile and add the instructions below. The Dockerfile starts with a base image for Node indicated by the FROM instruction. A working directory is created and the package.json file is copied into it. The dependencies listed in the package.json file are used to install the dependent packages. Next, the source code for the Next.js application is copied. The Prisma client is generated. Port 3000 is exposed because the Next.js application listens on port 3000. Finally, the command to start the Next.js application server is added as the last instruction.

```
FROM node:lts-buster-slim AS base
RUN apt"-" get update && apt"-" get install libssl"-" dev ca"-" certificates "-
" y
WORKDIR /app
COPY package.json package"-" lock.json ./
FROM base as build
RUN export NODE_ENV"=" production
RUN yarn
COPY . .
RUN npx prisma generate
RUN yarn build
FROM base as prod"-" build
RUN yarn install "--" production
COPY prisma prisma
RUN npx prisma generate
RUN cp "-" R node_modules prod_node_modules
FROM base as prod
COPY "--" from"=" prod"-" build /app/prod_node_modules /app/node_modules
COPY "--" from"=" build /app/.next /app/.next
COPY "--" from"=" build /app/public /app/public
COPY "--" from"=" build /app/prisma /app/prisma
EXPOSE 3000
CMD ["yarn","start"
```

2. Create the docker-compose.yml file and add the content below. This step is required so that the .env.local file is properly copied into the Docker container.

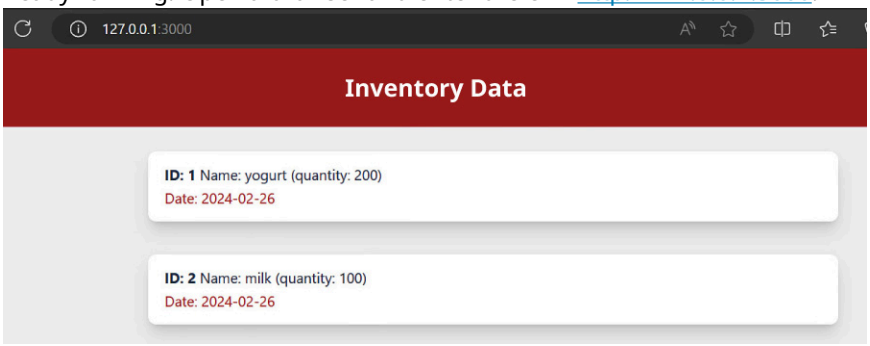
```
services:
  web:
    ports:
      - "3000:3000"
    build:
      dockerfile: Dockerfile
      context: ./
    volumes:
      - .env.local:/app/.env.local
```

3. Build the Docker image and push it to the ACR registry with this command. The Docker image must be built locally and then pushed to the ACR registry later. To build the image, run the command below. The command will build the image and then run the container. Wait until the image builds and the container starts.

```
docker compose up -d
```

To stop the container, run the command `docker compose down`.

4. Test run the front-end application. Because docker compose was used for the front end, the Docker container is already running. Open a browser and enter the URL http://127.0.0.1:3000.



(rendered using Docker; attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

5. Tag the image so that it can be pushed to the ACR registry with the command below. *Note*: run "docker images" to confirm the correct image name is used for the docker tagging. Substitute <registry_name> with the correct name of the ACR registry.

```
docker tag cna-next_web:latest <registry_name>.azurecr.io/cna-next_web:v1
```

6. Push Images to ACR Registry. First, confirm the images for both the front-end and back-end applications are tagged properly. Run the command: "docker images." The following four images should be listed (the originally built images, and then the images tagged for the ACR registry).

```
$ docker images
REPOSITORY                                  TAG       IMAGE ID       CREATED        SIZE
cna-next_web                                latest    8e95d5095f07   2 hours ago    759MB
ncaregistryflex.azurecr.io/cna-next_web     v1        8e95d5095f07   2 hours ago    759MB
expressimage                                latest    bdd5943f5bcb   3 hours ago    165MB
ncaregistryflex.azurecr.io/expressimage     v1        bdd5943f5bcb   3 hours ago    165MB
```

(attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

7. Log in to the ACR registry with the command below.

```
az acr login --name $REGISTRYNAME
```

Note: running the above command assumes being logged into the Azure Portal. This can be done with the following Azure CLI command.

```
az login --scope https://management.core.windows.net//.default
```

8. Push both images to the ACR registry. Run the two commands below. Substitute <registry_name> with

the correct name of the ACR registry.

```
docker push <registry_name>.azurecr.io/expressimage:v1
docker push <registry_name>.azurecr.io/cna-next_web:v1
```

9. In the Azure Console, navigate to the container registry and click on the registry name, *ncaregistryflex*. Under Services, click Repositories. Confirm that both image repositories are created. For this example, the repositories are *expressimage* and *cna-next-web*.

Create an Azure Kubernetes Service Instance

Now that the Docker images for the two microservices have been created and pushed to their respective repositories, the next step is to create a Kubernetes cluster.

1. In the Azure Portal search bar, search for Kubernetes services. Click on Kubernetes services. Select Create a Kubernetes cluster.
2. On the Azure Portal Home page, click on Create a resource. From the menu on the left, click Containers. Click on Azure Kubernetes Service (AKS).
3. On the Basics tab, configure the resource attributes. Table 12.3 shows the list of settings to configure on the Basics tab. Any settings not included should be set to the default values provided in the wizard.

Basics	Tab Settings
Subscription	Select the default subscription.
Resource group	For this example, *rg-nativeapps-eastus* is used.
Kubernetes cluster name	Enter a unique name for the resource. For this example, *nca-aks* is used.
Scale method	Select Manual.
Location	For this example, select the region that is used for the resource group.
Node count	Set to 2.

Table 12.3 Configuration for the Azure Kubernetes Service

4. On the Integrations tab, select the container registry that was created previously.
5. Click Review + create. Then, click Create. Wait until the Kubernetes cluster is provisioned.

Deploy Microservices to the Kubernetes Cluster

Now that the Kubernetes cluster is provisioned successfully, the next step is to deploy the containerized microservices into it. The images for these microservices are pulled from the registry and deployed into pods in the Kubernetes cluster.

Set Up the Environment

1. Setting specific environment variables makes it easier to run the commands that follow. Environment variables need to be set up for the resource group, Kubernetes cluster, and the container registry. Run the following commands to set these variables. Note that for this example, the resource group used is *rg-nativeapps-eastus*, the Kubernetes cluster name used is *nca-aks*, and the registry name is *ncaregistry*.

```
RESOURCEGROUP={resource_group}
CLUSTERNAME={cluster_name}
REGISTRYNAME={registry_name}
```

2. The Kubernetes cluster must be able to connect to the ACR registry to pull the Docker images from it. Connect the Kubernetes cluster to the ACR registry using this command.
   `az aks get-credentials --resource-group $RESOURCEGROUP --name $CLUSTERNAME`
3. **Kubectl** is the Kubernetes command-line tool that is used to manage Kubernetes clusters. This is the tool that interacts with the Kubernetes cluster on Azure. The first step in interacting with the Kubernetes cluster is to connect to it. Run the command, which lists the nodes of the Kubernetes cluster. The output lists the nodes for the cluster.
   `kubectl get nodes`
4. The next step is to obtain the hostname to the ACR registry. This is added to the deployment manifest files for the microservices so that the Docker images can be pulled from the registry and deployed into the Kubernetes cluster. Run the command to query the ACR server.
   `az acr list --resource-group $RESOURCEGROUP --query "[].{acrLoginServer:loginServer}" --output table`

Create and Apply the Deployment Manifests

Deployment manifest files are used to deploy the Docker images for the microservices into the Kubernetes cluster. They provide declarative updates for the Kubernetes Pods and ReplicaSets for the microservices. Initially, only one instance of each microservice is made available. Each microservice can scale up or down as needed.

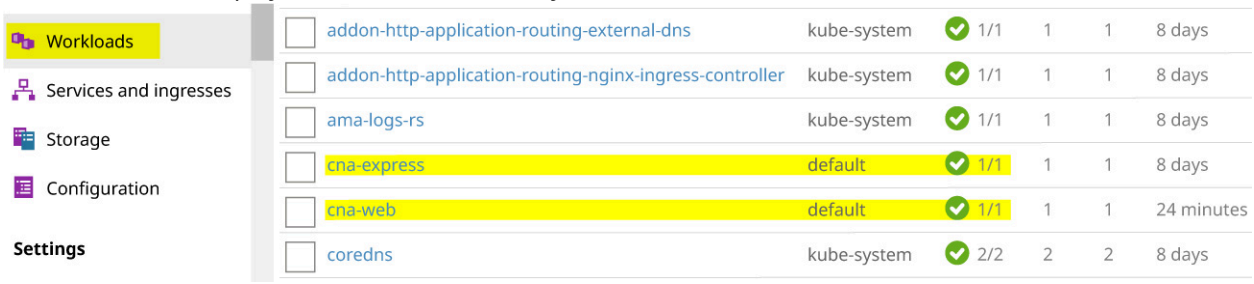
1. Create a deployment manifest file for the Docker image, *expressimage*, for the back-end service. Create the express-deployment.yaml file and enter the following content. The deployment manifest deploys the back-end service with the label *cna-express*. In this deployment manifest file, the Docker image is pulled from the registry and gets deployed in a pod in Kubernetes.

```
  # deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cna-express
spec:
  selector: # Define the wrapping strategy
    matchLabels: # Match all pods with the defined labels
      app: cna-express # Labels follow the `name: value` template
  template: # This is the template of the pod inside the deployment
    metadata:
      labels:
        app: cna-express
    spec:
      containers:
        - image: ncaregistry.azurecr.io/expressimage
          name: expressimage
          ports:
            - containerPort: 80
```

2. Apply the deployment manifest to the Kubernetes cluster with the following command. A message indicates that the deployment object was successfully created.
   `kubectl apply -f ./express-deployment.yaml`
3. Create a deployment manifest file for the Docker image, *webimage*, for the front-end service. Create the web-deployment.yaml file and enter the following content. The deployment manifest deploys the front-end service with the label *cna-web*. In this deployment manifest file, the Docker image is pulled from the registry and gets deployed in a pod in Kubernetes.

```
# deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cna-web
spec:
  selector: # Define the wrapping strategy
    matchLabels: # Match all pods with the defined labels
      app: cna-web # Labels follow the `name: value` template
  template: # This is the template of the pod inside the deployment
    metadata:
      labels:
        app: cna-web
    spec:
      containers:
        - image: ncaregistry.azurecr.io/webimage
          name: webimage
          ports:
            - containerPort: 80
```

4. Apply the deployment manifest to the Kubernetes cluster with this command. A message indicates that the deployment object was successfully created.
   `kubectl apply -f ./web-deployment.yaml`
5. Confirm the deployments for both the back-end and front-end services were successful with the following commands. Both microservices should display a status of "Running" in the Kubernetes cluster.
   `kubectl get deploy cna-express`
   `kubectl get pods`
6. In the Azure Console, navigate to the Kubernetes resources page. Click on Workloads. The microservices are deployed with a status of Ready.



(Used with permission from Microsoft)

Create and Apply the Service Manifests

Both microservices are now deployed in a Kubernetes cluster. Kubernetes Services are created for each microservice so that they can receive client requests. Kubernetes provides the Pods where the microservices are deployed with IP addresses and a single fully qualified domain name (FQDN) for a set of Pods. In addition, Services expose TCP ports to the containers where the microservices are running.

1. Create a service manifest file for the Docker image, *expressimage*, for the back-end service. Create the express-service.yaml file and enter the following content. The service manifest creates the Kubernetes Service for the back-end service with the label *cna-express*.

```
#service.yaml
apiVersion: v1
kind: Service
metadata:
```

```
  name: cna-express
spec:
  type: ClusterIP
  selector:
    app: cna-express
  ports:
    - port: 8080 # SERVICE exposed port
      name: http # SERVICE port name
      protocol: TCP # The protocol the SERVICE will listen to
      targetPort: 8080
```

2. Apply the service manifest to the Kubernetes cluster with this command. A message indicates that the service object was successfully created.
   ```
   kubectl apply -f ./express-service.yaml
   ```
3. Create a service manifest file for the Docker image, *webimage*, for the front-end service. Create the web-service.yaml file and enter the following content. The service manifest creates the Kubernetes Service for the front-end web service with the label *cna-web*.
   ```
   #service.yaml
   apiVersion: v1
   kind: Service
   metadata:
     name: cna-web
   spec:
     type: ClusterIP
     selector:
       app: cna-web
     ports:
       - port: 3000 # SERVICE exposed port
         name: http # SERVICE port name
         protocol: TCP # The protocol the SERVICE will listen to
         targetPort: 3000
   ```
4. Apply the service manifest to the Kubernetes cluster with this command. A message indicates that the service object was successfully created.
   ```
   kubectl apply -f ./web-service.yaml
   ```
5. Confirm the service deployment was successful. The Services for each microservice should be listed. IP addresses (CLUSTER-IP) and ports should also be specified for each microservice.
   ```
   kubectl get service cna-express
   kubectl get service
   ```

Create and Apply the Ingress Controllers

Now that the services are created with assigned IP addresses and exposed ports, Ingress controllers are used to define how the deployed microservices are exposed to outside requests.

1. First, enable the Kubernetes cluster so that it can use HTTP Application Routing with the following command.
   ```
   az aks enable-addons --resource-group $RESOURCEGROUP \
   --name $CLUSTERNAME --addons http_application_routing
   ```
2. Next, configure and deploy the Ingress controller. As mentioned earlier, an FQDN is provided for a set of Pods of the Kubernetes cluster. Run this command to obtain this FQDN. The output is the FQDN that is used to expose the microservices to outside requests.
   ```
   az aks show --resource-group $RESOURCEGROUP --name $CLUSTERNAME -o tsv \
   ```

```
--query
addonProfiles.httpApplicationRouting.config.HTTPApplicationRoutingZoneName
```

3. Create the Ingress descriptor file, express-ingress.yaml, for the back-end service labeled as *cna-express* and add the following content. Note that the **host** includes the FQDN that was obtained in the previous step. It is prepended with *cna-express* making it unique for the back-end service.

```
#ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: cna-express
  annotations:
    kubernetes.io/ingress.class: addon-http-application-routing
spec:
  rules:
  - host: cna-express.8776bb8bc0324e10946c.eastus.aksapp.io
    http:
      paths:
      - path: / # Which path is this rule referring to
        pathType: Prefix
        backend: # How the ingress will handle the requests
          service:
            name: cna-express # Which service the request will be forwarded to
            port:
              name: http # Which port in that service
```

4. Apply the Ingress manifest to the Kubernetes cluster with this command. A message indicates that the Ingress object was successfully created.

```
kubectl apply -f ./express-ingress.yaml
```

5. Create the Ingress descriptor file, web-ingress.yaml, for the front-end web service labeled as *cna-web* and add the following content. Note that the **host** includes the FQDN that was obtained in the previous step. It is prepended with *cna-web* making it unique for the front-end web service.

```
#ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: cna-web
  annotations:
    kubernetes.io/ingress.class: addon-http-application-routing
spec:
  rules:
  - host: cna-web.8776bb8bc0324e10946c.eastus.aksapp.io
    http:
      paths:
      - path: / # Which path is this rule referring to
        pathType: Prefix
        backend: # How the ingress will handle the requests
          service:
            name: cna-express # Which service the request will be forwarded to
            port:
              name: http # Which port in that service
```

6. Apply the ingress manifest to the Kubernetes cluster with this command. A message indicates that the

Ingress object was successfully created.

```
kubectl apply -f ./web-ingress.yaml
```

7. Confirm the Ingresses were deployed successfully with the commands below. The Ingresses for both microservices will display host names (HOSTS) that are unique for each microservice.

```
kubectl get ingress cna-express
kubectl get ingress
```

8. The command below queries Azure for the FQDN that was created earlier. It serves as the ZoneName. The command also returns the ResourceGroup value that is used to access the microservices. Run the command below and make a note of the ZoneName and ResourceGroup values.

```
az network dns zone list --output table
```

9. Substitute the values for ResourceGroup and ZoneName obtained in the previous step for <resource-group> and <zone-name>, respectively, in the command that follows. Execute the edited command in the Cloud Shell, which results in the table shown. Two records are added for *cna-express* and two for *cna-web*, which show in the Name column.

```
az network dns record-set list -g <resource-group> -z <zone-name> --output table
```

TTL	Fqdn	Name	ProvisioningState	ResourceGroup
172800	8776bb8bc0324e10946c.eastus.aksapp.io.	@	Succeeded	mc_rg-nativeapps-eastus_nca-aks_eastus
3600	8776bb8bc0324e10946c.eastus.aksapp.io.	@	Succeeded	mc_rg-nativeapps-eastus_nca-aks_eastus
300	cna-express.8776bb8bc0324e10946c.eastus.aksapp.io.	cna-express	Succeeded	mc_rg-nativeapps-eastus_nca-aks_eastus
300	cna-express.8776bb8bc0324e10946c.eastus.aksapp.io.	cna-express	Succeeded	mc_rg-nativeapps-eastus_nca-aks_eastus
300	cna-web.8776bb8bc0324e10946c.eastus.aksapp.io.	cna-web	Succeeded	mc_rg-nativeapps-eastus_nca-aks_eastus
300	cna-web.8776bb8bc0324e10946c.eastus.aksapp.io.	cna-web	Succeeded	mc_rg-nativeapps-eastus_nca-aks_eastus

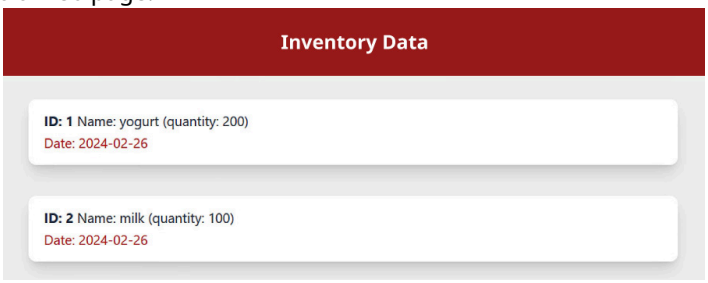
(attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

10. In a browser, access the back-end service using the URL that is generated (e.g., http://cna-express.8776bb8bc0324e10946c.eastus.aksapp.io/inventory/1), which includes the route that was implemented as part of the REST API for the microservice. The back-end service receives requests in JSON format to create and store inventory records into the datastore. It also retrieves inventory records from the datastore and renders them in JSON format.


```
{"inventory":[{"id":1,"name":"yogurt","quantity":200,"date":"2023-04-17"}]}
```


(attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

11. In a browser, access the front-end web service using the URL that is generated (e.g., http://cna-web.8776bb8bc0324e10946c.eastus.aksapp.io). The front-end web service renders inventory records from the datastore to a web page.



(rendered using Kubernetes; attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

PaaS Deployment of a Sample Cloud-Native Application Using VMWare Tanzu

This section demonstrates how to use VMWare Tanzu deployment technology to launch a dashboard on AWS monitoring application metrics exposed by the Kuard (Kubernetes Up and Running Demo) application. **Amazon Web Service (AWS)** is an Amazon cloud computing platform that offers a wide range of services that allow customers to build, deploy, and manage applications and services in the cloud, and will be used in this example.

Set Up an Environment for VMWare Tanzu

The first step is to create an AWS EC2 instance. This is a VM hosted in the cloud. Once the EC2 instance is provisioned, a set of tools need to be installed, including Docker, Kubernetes CLI tool, and other relevant package managers.

Create the EC2 Instance

1. **AWS Portal** is a web-based console that allows customers to manage their cloud services and AWS subscriptions. In the AWS Portal, search for EC2 and click Launch Instance. **Amazon Elastic Compute Cloud (EC2)** is a cloud based, on-demand, compute platform that can be auto-scaled to meet demand.
2. Under Application and OS Images (Amazon Machine Image), select Ubuntu, and leave the remaining configurations to the default settings for this section.
3. Under the Instance type section, select t2.micro for the Instance type.
4. Under the Key pair (login) section, click on Create new key pair.
5. Enter a value for Key pair name and keep the default settings. Click on Create key pair. The name used for this example is *cnatanzu-private-key*. Download the *cnatanzu-private-key.pem* file. This file is used to log in to the EC2 instance. The name provided, *cnatanzu-private-key*, is populated for the Key pair name in the Key pair (login) section.
6. Under the Networking settings section, select Create security group. Enable Allow SSH traffic from, Allow HTTPS traffic from the internet, Allow HTTP traffic from the internet. Set Anywhere to 0.0.0.0/0.
7. Under the Summary section, click Launch instance. Wait until the provisioning is complete. A Success message appears with the Instance ID. Click on the Instance ID link.
8. Open a terminal and navigate to the directory where the *cnatanzu-private-key.pem* file was downloaded. Under Instance summary, copy the value from Public IPv4 DNS. Use the command to make an SSH connection to the EC2 instance by inserting the copied Public IPv4 DNS.
   ```
   ssh -i nucamp-private-key.pem ubuntu@< EC2 public IPv4 DNS>
   ```

Install Homebrew (brew)

Homebrew is an open-source package management system used to install and manage packages on MacOS and Linux operating systems. Homebrew, also referred to as “brew,” is used to install Octant later.

1. Homebrew uses a compiler environment to build packages that may need to be built. The first step is to install a compiler environment with this command. The package build-essentials provides all required packages for the compiler environment. Run the command below to install the build-essentials package
   ```
   sudo apt install build-essential
   ```
2. Run the following command to download the brew installation script that is used to install brew.
   ```
   curl -fsSL -o install.sh
   ```
3. Run the following command to launch the installation script.
   ```
   /bin/bash install.sh
   ```
4. Once the installation script is complete, run the following two commands to add brew to the PATH environment variable. This makes the command brew recognizable in the EC2 instance.
   ```
   (echo; echo 'eval \"$(/home/linuxbrew/.linuxbrew/bin/brew shellenv)"') >> \
   /home/ubuntu/.profile eval \"$(/home/linuxbrew/.linuxbrew/bin/brew shellenv)"
   ```
5. Confirm brew was added to the PATH environment variable. Type the command brew to confirm the brew help menu is printed to the console. This indicates that brew was installed successfully.

Install and Set Up Docker

Tanzu requires the docker engine to be up and running. **Docker Engine** is an open-source containerization platform used to build Docker images and manage Docker containers. To install the Docker engine, **Advanced Package Tool (apt)** is used. Apt is a packaging tool used to install new packages and update existing packages.

1. Run the command to update the apt utility itself before installing Docker.
   ```
   sudo apt update
   ```
2. Run the command below to install the following packages as prerequisites for installing Docker:
   - apt-transport-https, which allows the package manager to transfer files over the HTTPS protocol
   - ca-certificates, which makes available common Certificate Authority certificates to aid in verifying the security of connections
   - curl, which is used for transferring data to or from a server
   - software-properties-common, which is a package to help manage the software installations

   ```
   sudo apt install apt-transport-https ca-certificates curl software-properties-common
   ```
3. Download a public key file from Docker with the command below. The public key file is then added to a list of trusted keys managed by apt.
   ```
   curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add
   ```
4. Run the add-apt-repository command below to add the external Docker repository to the apt sources list.
   ```
   sudo add-apt-repository \"deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable\""
   sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"
   ```
5. After making the changes to the apt package information in the previous steps, update the apt index with this command.
   ```
   sudo apt update
   ```
6. Run the command below to install Docker on the EC2 instance:
   ```
   sudo apt install docker-ce
   ```
7. Run the next three commands to configure Docker and install Docker Compose.
   - Add the current ubuntu user to the docker group.
     ```
     sudo usermod -aG docker ${USER}
     ```
   - Download Docker Compose. **Docker Compose** is a tool used for defining, via descriptor files, and managing applications that consist of multiple Docker containers.
     ```
     sudo curl -L \
     "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" \
     -o /usr/local/bin/docker-compose
     ```
   - Run this command to change the permissions of the downloaded docker-compose file to an executable so that it can be run.
     ```
     sudo chmod +x /usr/local/bin/docker-compose
     ```
8. Confirm that both docker and docker-compose were installed. Run these commands and confirm that the versions for docker and docker-compose are displayed to the console.
   ```
   docker -v
   docker-compose -v
   ```
9. Run the command below to start the Docker engine in the EC2 instance.
   ```
   sudo systemctl start docker
   ```
10. Run the command below to test if Docker was installed correctly.
    ```
    sudo docker run hello-world
    ```

Install and Set Up Kubectl

Tanzu requires kubectl. The Kubernetes command-line tool, kubectl, is used to interact with the Tanzu Kubernetes cluster.

1. Download the latest release of kubectl with the command below.
   ```
   curl -LO \"https://dl.k8s.io/release/$(curl -L -s \
   https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
   ```
2. Install kubectl with the command below.
   ```
   sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
   ```
3. Run the following commands to change the permissions to kubectl file to executable so that it can be run and copy it to the ~/.local/bin directory.
   ```
   chmod +x kubectl
   mkdir -p ~/.local/bin
   mv ./kubectl ~/.local/bin/kubectl
   ```
4. Append ~/.local/bin to the $PATH environment variable. Edit the ~/.bashrc file and add the following line to the end of it.
   ```
   export PATH=$PATH:~/.local/bin
   ```
5. Save and close the file. Apply the changes by running the following command:
   ```
   source ~/.bashrc
   ```
6. Confirm, by running the command kubectl, that it can be executed. The kubectl help menu should display in the console. This indicates that it was installed successfully.

Install Tanzu

1. Install jq as a support package for Tanzu with the following commands:
   ```
   sudo apt-get update
   sudo apt-get install jq
   ```
2. Install xdg-utils as a support package for Tanzu with the following command:
   ```
   sudo apt-get install --reinstall xdg-utils
   ```
3. Ensure the kubectl version is compatible with the Tanzu version.
   ```
   curl -H \"Accept: application/vnd.github.v3.raw" -L \
   https://api.github.com/repos/vmware-tanzu/community-edition/contents/hack/get-tce-release.sh\
   | bash -s v0.10.0 linux
   ```
4. Unpack the gzip file and install Tanzu using the provided shell script with the command below.
   ```
   tar xzvf tce-linux-amd64-v0.10.0.tar.gz
   ```
5. Navigate into the *tce-linux-amd64-v0.10.0/* directory and run the installation shell script with the command below.
   ```
   /install.sh
   ```
6. Confirm Tanzu was installed successfully. Run the command tanzu. The Tanzu help menu should be displayed in the console.

Install and Test the Kuard Demo Application

Now that the Docker engine, Docker Compose, Kubectl, and Tanzu are all installed, the next step is to install the Kuard demo application.

Create Required AWS Secret and Access Keys

AWS secret keys are used to manage AWS resources via the AWS CLI. **AWS CLI** is a command-line interface used to manage Amazon cloud services. Access keys are used in the Tanzu configuration to provision and configure the Tanzu Management Cluster. An RSA key pair is also used in the Tanzu configuration to communicate with the Tanzu Management Cluster.

1. In the AWS shell, install awscli. First, download the AWS CLI zip file with the curl command below.
   ```
   curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
   "awscliv2.zip"
   ```

- Install the unzip CLI utility that is used to unpack the downloaded awscliv2.zip file. Run the command below:
  ```
  sudo apt install unzip
  ```
- Unzip the awscliv2.zip file with the command below.
  ```
  unzip awscliv2.zip
  ```
- Install the unpackaged AWS CLI with the command below.
  ```
  sudo ./aws/install
  ```
- Run the command below to verify that the AWS CLI was installed successfully. The version should be displayed.
  ```
  aws - version
  ```

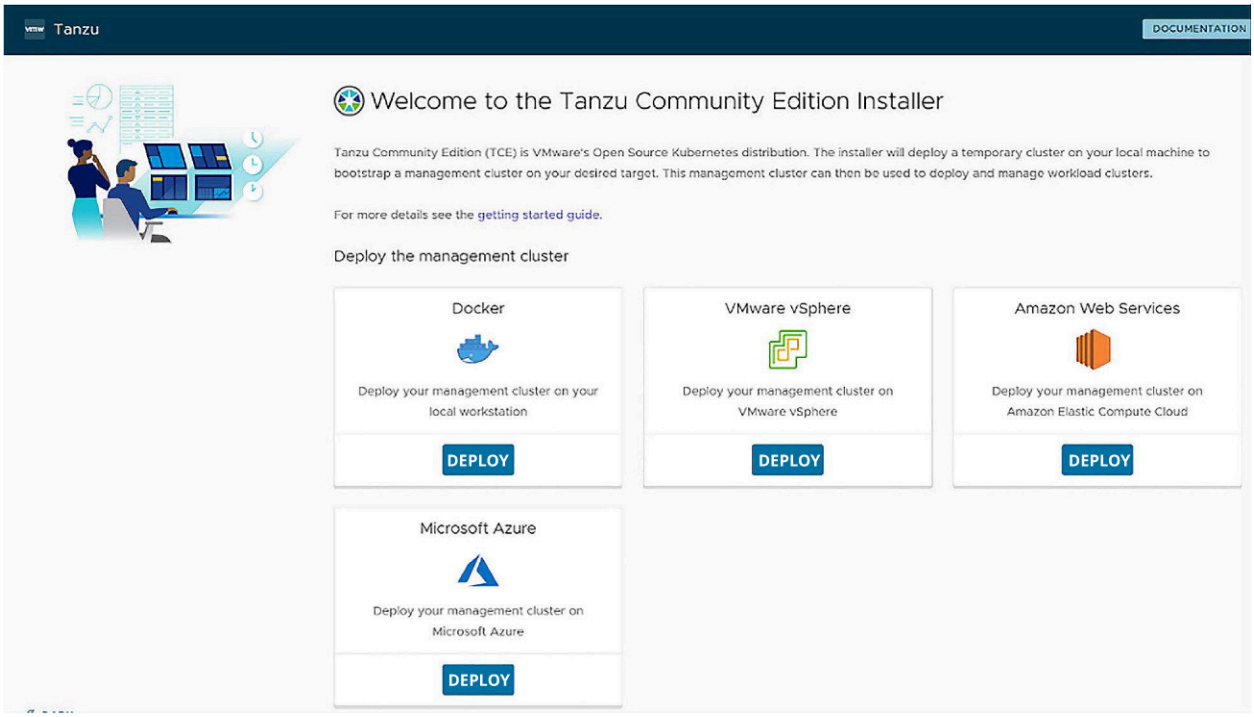
2. In the AWS Portal under Access management, click on Security credentials. Click on Create access key to generate the key. Make a note of the key pairs as they will be added to the Tanzu Management Cluster settings.
3. Finally, generate an RSA key pair. The name used in this example is *tanzu-key-pair*. Make a note of the name of the RSA key pair as it will be added to the Tanzu Management Cluster settings.
   ```
   aws ec2 create-key-pair \
     --key-name tanzu-key-pair \
     --key-type rsa \
     --key-format pem \
     --query "KeyMaterial" \
     --output text > tanzu-key-pair.pem
   ```

Create the Tanzu Management Cluster Configuration

Now that the environment to run the Tanzu Management Cluster is set up, the next step is to create a Tanzu Management Cluster. A management cluster configuration file is needed. One way to create a management cluster configuration file is to use the Tanzu installer, which is a web page that is used to generate the configuration file.[14]

1. Use the Tanzu installer to create the management cluster configuration file. Launch the Tanzu installer web page with the command below. A browser page automatically launches.
   ```
   tanzu management-cluster create
   ```

14 See VMware Tanzu Kubernetes Grid documentation at https://docs.vmware.com/en/VMware-Tanzu-Kubernetes-Grid/1.5/vmware-tanzu-kubernetes-grid-15/GUID-mgmt-clusters-config-aws.html.



(credit: Tanzu, under Mozilla Public License 2.0)

2. Select Deploy on Amazon Web Services. This generates a configuration file that deploys the cluster on the Amazon EC2 instance that was created earlier.
3. Under the IaaS Provider section, select the REGION that is also used as a location for the EC2 instance.
4. Under the Management Cluster Settings section, make sure that Bastion Host is unchecked (disabled). For EC2 KEY PAIR, enter the name of the RSA key pair that was created earlier. For this example, the name used in this example is *tanzu-key-pair*. Select *t3a.large* for the AZ1 WORKER NODE INSTANCE TYPE.
5. Under the Identity Management section, disable Enable Identity Management Settings.
6. Under the OS Image section, select *ubuntu-20-04-amd64* as the OS IMAGE.
7. Click Review Configuration. Table 12.4 shows the complete list of settings expected for configuring the Tanzu Management Cluster.

IaaS Provider Settings	
IaaS Provider	Validate the AWS provider credentials for Tanzu Community Edition
AWS CREDENTIAL PROFILE	Default
REGION	us-east-2
VPC for AWS settings	
VPC for AWS	Specify VPC settings for AWS
VPC CIDR	10.0.0.0/16

Table 12.4 Configurations for Tanzu

IaaS Provider Settings	
Management Cluster Settings	
Management Cluster Settings	Development cluster selected: 1 node control plane
DEV INSTANCE TYPE	t3a.large
MANAGEMENT CLUSTER NAME	tkg-cnamgmt-cluster-aws
EC2 KEY PAIR	tanzu-key-pair
ENABLE MACHINE HEALTH CHECKS	Yes
ENABLE BASTION HOST	No
ENABLE AUDIT LOGGING	No
AUTOMATE CREATION OF AWS CLOUDFORMATION STACK	Yes
AVAILABILITY ZONE 1	us-east-2c
WORKER NODE INSTANCE TYPE 1	t3a.large
PROD INSTANCE TYPE	
Metadata Settings	
Metadata	Specify metadata for the management cluster
LOCATION (OPTIONAL)	
DESCRIPTION (OPTIONAL)	
LABELS	
Kubernetes Network Settings	
Kubernetes Network	Cluster Pod CIDR: 100.96.0.0/11
CNI PROVIDER	Antrea
CLUSTER SERVICE CIDR	100.64.0.0/13
CLUSTER POD CIDR	100.96.0.0/11
ENABLE PROXY SETTINGS	No

Table 12.4 Configurations for Tanzu

IaaS Provider Settings	
Identity Management Settings	
Identity Management	Specify identity management
ENABLE IDENTITY MANAGEMENT SETTINGS	No
OS Image Settings	
OS Image	OS Image: ubuntu-20.04-amd64 (ami-06159f2d2711f3434)
OS IMAGE	ubuntu-20.04-amd64 (ami-06159f2d2711f3434)

Table 12.4 Configurations for Tanzu

8. Copy the generated CLI command to the clipboard (or click Deploy Management Cluster). Clicking on the Deploy Management Cluster button connects to the EC2 instance and provisions the Tanzu Management Cluster. Alternatively, the configuration file that was generated is copied into the /home/ubuntu/.config/tanzu/tkg/clusterconfigs/ directory. In this example, the generated filename for the configuration file is *kldlaarqyl.yaml*. Run the copied command in the AWS shell to provision the Tanzu Management Cluster.
9. After the copied command is executed, a config file, with "config_" prepended as the filename, is generated, and copied into the kube-tkg/tmp/ directory.
10. Run the command below to check the status of the Tanzu Management Cluster. Change <config_file> with the filename generated.
    `Kubectl get \ po.deploy.cluster.kubeadmcontrolplane,machine,machinedeployment\ -A --kubeconfig /home/ubuntu/.kube-tkg/tmp/<config_file>`

```
NAMESPACE                           NAME                                                                READY   STATUS    RESTARTS   AGE
capa-system                         pod/capa-controller-manager-7dfd4798f5-wctbd                        2/2     Running   0          8m51s
capi-kubeadm-bootstrap-system       pod/capi-kubeadm-bootstrap-controller-manager-6494884869-rxmph      2/2     Running   0          8m58s
capi-kubeadm-control-plane-system   pod/capi-kubeadm-control-plane-controller-manager-857d687b9d-dwk8z  2/2     Running   0          8m56s
capi-system                         pod/capi-controller-manager-778bd4dfb9-46jtg                        2/2     Running   0          8m59s
capi-webhook-system                 pod/capa-controller-manager-76f6f484c5-5vtdt                        2/2     Running   0          8m53s
capi-webhook-system                 pod/capi-controller-manager-9995bdc94-47khp                         2/2     Running   0          9m
capi-webhook-system                 pod/capi-kubeadm-bootstrap-controller-manager-68845b65f8-c6nm2      2/2     Running   0          8m58s
capi-webhook-system                 pod/capi-kubeadm-control-plane-controller-manager-9847c6747-vwgqf   2/2     Running   0          8m56s
cert-manager                        pod/cert-manager-77f6fb8fd5-2fxxk                                   1/1     Running   0          9m27s
cert-manager                        pod/cert-manager-cainjector-6bd4cff7bb-ghl7n                        1/1     Running   0          9m27s
cert-manager                        pod/cert-manager-webhook-fbfcb9d6c-4l2r2                            1/1     Running   0          9m27s
kube-system                         pod/coredns-657879bf57-6dknf                                        1/1     Running   0          9m46s
kube-system                         pod/coredns-657879bf57-6xvd4                                        1/1     Running   0          9m46s
kube-system                         pod/etcd-tkg-kind-ch7d1qi4lfgn5u7cfv30-control-plane                1/1     Running   0          9m53s
kube-system                         pod/kindnet-9ljsn                                                   1/1     Running   0          9m47s
kube-system                         pod/kube-apiserver-tkg-kind-ch7d1qi4lfgn5u7cfv30-control-plane      1/1     Running   0          9m53s
kube-system                         pod/kube-controller-manager-tkg-kind-ch7d1qi4lfgn5u7cfv30-control-plane  1/1     Running   0          9m53s
kube-system                         pod/kube-proxy-cfvhf                                                1/1     Running   0          9m47s
kube-system                         pod/kube-scheduler-tkg-kind-ch7d1qi4lfgn5u7cfv30-control-plane      1/1     Running   0          10m
local-path-storage                  pod/local-path-provisioner-8b46957d4-2xwd2                          1/1     Running   0          9m46s

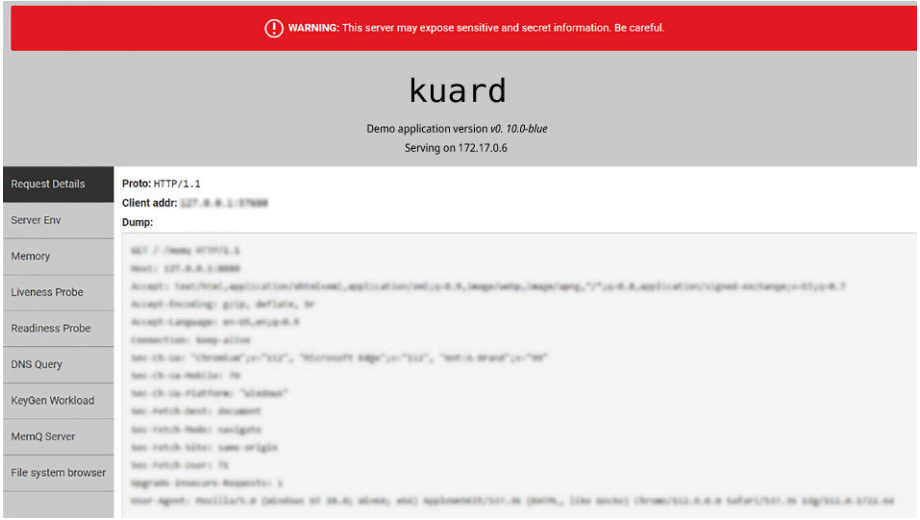
NAMESPACE                           NAME                                                          READY   UP-TO-DATE   AVAILABLE   AGE
capa-system                         deployment.apps/capa-controller-manager                       1/1     1            1           8m52s
capi-kubeadm-bootstrap-system       deployment.apps/capi-kubeadm-bootstrap-controller-manager     1/1     1            1           8m58s
capi-kubeadm-control-plane-system   deployment.apps/capi-kubeadm-control-plane-controller-manager 1/1     1            1           8m56s
capi-system                         deployment.apps/capi-controller-manager                       1/1     1            1           8m59s
capi-webhook-system                 deployment.apps/capa-controller-manager                       1/1     1            1           8m53s
capi-webhook-system                 deployment.apps/capi-controller-manager                       1/1     1            1           9m
capi-webhook-system                 deployment.apps/capi-kubeadm-bootstrap-controller-manager     1/1     1            1           8m58s
capi-webhook-system                 deployment.apps/capi-kubeadm-control-plane-controller-manager 1/1     1            1           8m57s
cert-manager                        deployment.apps/cert-manager                                  1/1     1            1           9m27s
cert-manager                        deployment.apps/cert-manager-cainjector                       1/1     1            1           9m27s
cert-manager                        deployment.apps/cert-manager-webhook                          1/1     1            1           9m27s
kube-system                         deployment.apps/coredns                                       2/2     2            2           10m
local-path-storage                  deployment.apps/local-path-provisioner                        1/1     1            1           9m56s
```

(attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Run the Kuard Demo Application

Kuard (Kubernetes Up and Running Demo) is a demo application that provides information about Kubernetes environments that are running. The Kuard demo application is deployed in the Tanzu Management Cluster as a containerized application.

1. Run the command below to set the default context to the Tanzu cluster that was just created.
   `kubectl config use-context tkg-cnamgmt-cluster-aws-admin@tkg-cnamgmt-cluster-aws`
2. Run the command below to pull the Kuard image and start a single instance of a Kuard Pod.
   `kubectl run --restart=Never –image=gcr.io/kuar-demo/kuard-amd64:blue kuard`
3. Configure Kuard to listen on port 8080 and forward to port 8080 in the Kuard pod. First, run the command below to list the Pod and make note of the Pod name. The Pod name should be *kuard* and should be up and running.
   `kubectl get pods`
4. Run the command below to use port forwarding and expose the Kuard default port, 8080.
   `kubectl port forward kuard 8080:8080`
5. Launch the Kuard website in the browser using the URL http://localhost:8080.

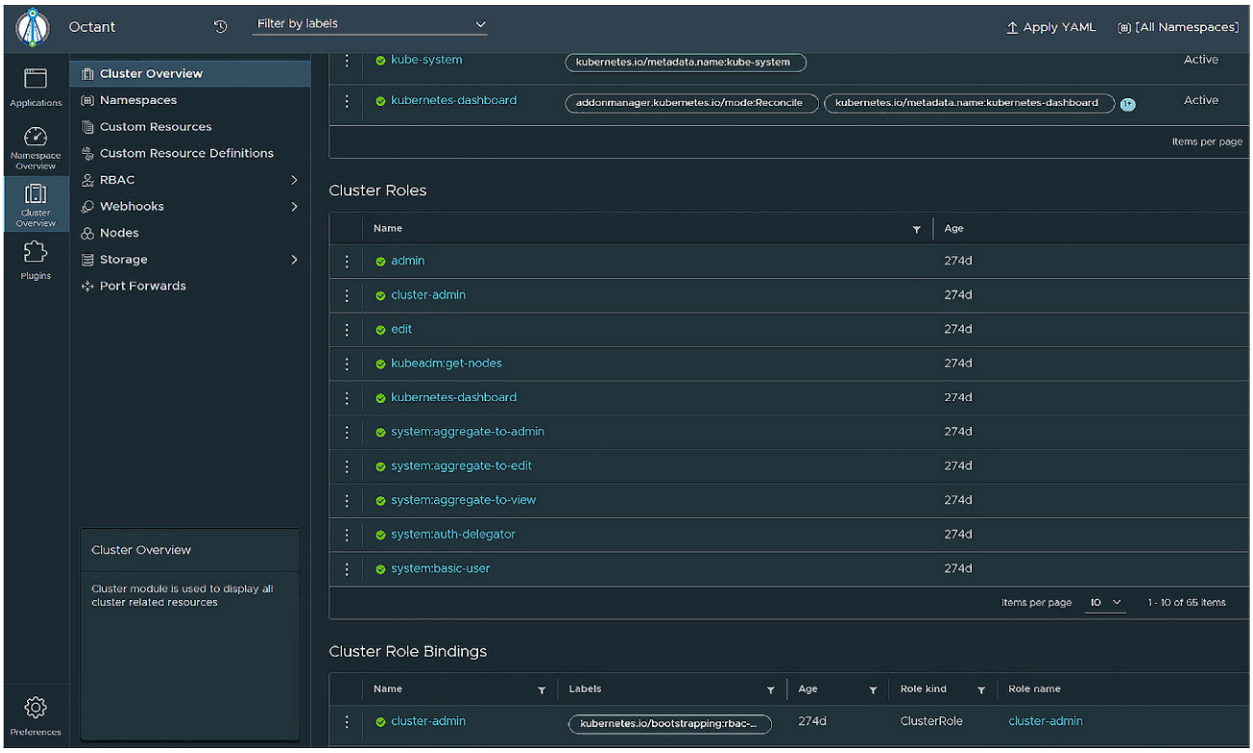


(credit: Kubernetes, under Mozilla Public License 2.0)

Install and Run Octant

Octant is an open-source web interface for Kubernetes that is used to inspect Kubernetes clusters and applications deployed in them.

1. Run the command below to install Octant using Homebrew.
   `brew install octant`
2. Launch Octant by typing the command octant in the AWS shell. A browser window automatically launches.



(credit: Octant, a VMware-backed project)

FaaS Deployment of a Sample Cloud-Native Application

The following example[15] illustrates FaaS deployment of a cloud-native application that consists of two event-driven workloads. The first serves as a simulator that sends data to an event hub. This is the Azure FaaS producer function. A second connects to this event hub to trigger storing the events in a datastore. This is the Azure FaaS consumer function. Both FaaS functions are deployed as Azure Function Apps.

Azure Function App is an event-driven serverless compute platform without provision or managing infrastructure. The datastore used is an Azure Cosmos Database. Dashboards are then used to monitor the performance of the Azure FaaS functions. The architecture of the FaaS deployment of the cloud-native application is shown in Figure 12.40.

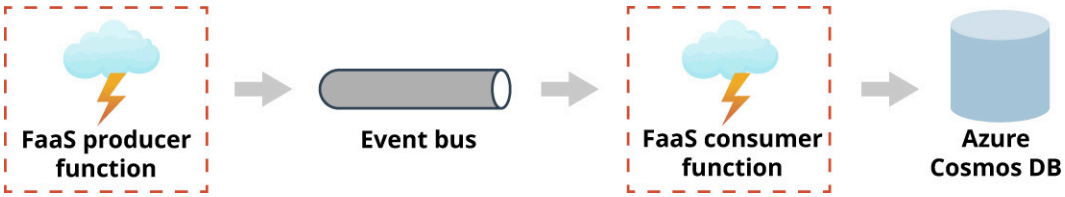


Figure 12.40 FaaS functions of a cloud-native application are deployed in Azure. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

FaaS Environment Setup

Setting specific environment variables makes it easier to run these commands. Run the commands below to set the environment variables. For this example, *rg-atafunctions-westus2* was used for the resource group, *ncafaaseventhub* was used for the event hub namespace, *ncafaaseventhub* was used for the event hub name, *ncafaasauth* was used for the event hub authorization rule, *ncafaasdbusr* was used for the Cosmos database account username, *ncafaasstor* was used for the storage account name, *ncafaasapp* was used for the FaaS function name (for the first Azure function), and *westus2* was used for the location.

15 Sample based off tutorial https://learn.microsoft.com/en-us/training/modules/deploy-real-time-event-driven-app/

```
RESOURCE_GROUP=<value>
EVENT_HUB_NAMESPACE=<value>
EVENT_HUB_NAME=<value>
EVENT_HUB_AUTHORIZATION_RULE=<value>
COSMOS_DB_ACCOUNT=<value>
STORAGE_ACCOUNT=<value>
FUNCTION_APP=<value>
LOCATION=<value>
```

Create a Datastore for the Event-Driven FaaS Cloud-Native Application

The producer function simulates and sends data to an Azure event hub. The consumer function listens for events of a specific namespace in the Azure event hub and processes and stores them in an Azure Cosmos database. The first step is to create an Azure Cosmos DB datastore. Once the datastore is created, the next step is to create and configure an Azure event hub.

1. Run the commands below to create the Azure Cosmos DB datastore.

```
az cosmosdb create\
  --resource-group $RESOURCE_GROUP\
  --name $COSMOS_DB_ACCOUNT

az cosmosdb sql database create\
  --resource-group $RESOURCE_GROUP\
  --account-name $COSMOS_DB_ACCOUNT \
  --name TelemetryDb

az cosmosdb sql container create\
  --resource-group $RESOURCE_GROUP\
  --account-name $COSMOS_DB_ACCOUNT\
  --database-name TelemetryDb\
  --name TelemetryInfo\
  --partition-key-path '/temperatureStatus'
```

Create and Configure an Event Hub

1. Run the commands below to create the event hub namespace, Azure event hub, and event hub authentication rule resources.

```
az eventhubs namespace create\
  --resource-group $RESOURCE_GROUP\
  --name $EVENT_HUB_NAMESPACE

az eventhubs eventhub create\
  --resource-group $RESOURCE_GROUP\
  --name $EVENT_HUB_NAME\
  --namespace-name $EVENT_HUB_NAMESPACE\
  --message-retention 1

az eventhubs eventhub authorization-rule create\
  --resource-group $RESOURCE_GROUP
  --name $EVENT_HUB_AUTHORIZATION_RULE
  --eventhub-name $EVENT_HUB_NAME
  --namespace-name $EVENT_HUB_NAMESPACE \
```

```
  --rights Listen Send
```

2. Run these commands below to create the storage account and function app resources.

```
az storage account create\
  --resource-group $RESOURCE_GROUP\
  --name $STORAGE_ACCOUNT"p" \
  --sku Standard_LRS

az functionapp create\
  --resource-group $RESOURCE_GROUP\
  --name $FUNCTION_APP"-p"\
  --storage-account $STORAGE_ACCOUNT"p" \
  --consumption-plan-location $LOCATION\
  --runtime java\
  --functions-version 4
```

Create, Build, and Deploy the FaaS Producer Function

A few resources need to be created for the FaaS producer function. First, a storage account is created. The FaaS producer function needs to connect to the Azure event hub. Connection strings need to be generated for this purpose. Finally, the FaaS producer function is built as a maven project and then deployed as an Azure Function App.

Set Up Storage for the Consumer Function

1. Run the commands below to create the connection strings that are used to access the storage account for the event hub.

```
AZURE_WEB_JOBS_STORAGE=$(\
az storage account show-connection-string\
  --resource-group $RESOURCE_GROUP\
  --name $STORAGE_ACCOUNT"p"\
  --query connectionString\
  --output tsv)

EVENT HUB_CONNECTION_STRING=$(\
az eventhubs eventhub authorization-rule keys list\
  --resource-group $RESOURCE_GROUP\
  --name $EVENT_HUB_AUTHORIZATION_RULE\
  --eventhub-name $EVENT_HUB_NAME\
  --namespace-name $EVENT_HUB_NAMESPACE\
  --query primaryConnectionString\
  --output tsv)
```

2. Run the commands below to obtain the connection strings that were created in the previous step. Make a note of these connection strings as they will be used later.

```
echo $AZURE_WEB_JOBS_STORAGE
echo $EVENT_HUB_CONNECTION_STRING
```

3. The connection strings that were generated for the Azure Web Jobs Storage and event hub in the previous step need to be added as application settings to the Azure Function App account in the command. Run the command below with these values inserted to create the Function App. A notification to the console indicates that the Function App was built successfully.

```
az functionapp config appsettings set\
  --resource-group $RESOURCE_GROUP\
  --name $FUNCTION_APP"-p" \
```

```
  --settings\
AzureWebJobsStorage=$AZURE_WEB_JOBS_STORAGE\
EventHubConnectionString=$EVENT_HUB_CONNECTION_STRING
```

Build and Deploy the FaaS Producer Function

Now that the Azure resources, such as the event hub, Azure Function App, and Storage account, have been created and configured, the next step is to create an Azure FaaS function project for the FaaS producer function. Maven is used to build the project.

1. Run the command below to create and build the function project. The telemetry-functions-producer/ directory is generated along with the files for the project. A Build Success message indicating the build was successful should appear in the console.
   ```
   mvn archetype:generate -batch-mode\
     -DarchetypeGroupid=com.microsoft.azure\
     -Darchetype ArtifactId=azure-functions-archetype\
     -DappName=$FUNCTION_APP"-p"\
     -DresourceGroup=$RESOURCE_GROUP\
     -DappRegion=$LOCATION\
     -DappServicePlanName=$LOCATION"plan" \
     -Dgroupid=com.learn\
     -DartifactId=telemetry-functions-producer
   ```
2. Run the command below to add application settings from the Azure function into the function project local.settings.json file located in the *telemetry-functions-producer/* root directory.
   ```
   func azure functionapp fetch-app-settings $FUNCTION_APP”-p”
   ```
3. Navigate to the telemetry-functions-producer/src/main/java/com/learn/ directory. Edit the Function.java file and replace all the code in it with the code that follows. The code declares a Function that establishes a connection to the Azure event hub.
   ```
   package com.learn;
   import com.microsoft.azure.functions.annotation.EventHubOutput;
   import com.microsoft.azure.functions.annotation.FunctionName;
   import com.microsoft.azure.functions.annotation.TimerTrigger;
   import com.microsoft.azure.functions.ExecutionContext;

   public class Function {

     @FunctionName("generatesSensorData")
     @EventHubOutput(
     name = "event",
     eventHubName = "", // blank because the value is included in the connection
   string
     connection = "EventHub ConnectionString")
     public Telemetryltem generateSensorData(
     @TimerTrigger(
     name = "timerInfo",
     schedule = "*/10*****") // every 10 seconds
     String timerInfo,
     final ExecutionContext context) {
       context.getLogger().info("Java Timer trigger function executed at:
   "+java.time.LocalDateTime.now()); double temperature = Math.random()* 100;
       double pressure = Math.random() * 50;
   ```

```
    return new Telemetryltem(temperature, pressure);
  }
}
```

4. Create a file named TelemetryItem.java and add the code below. The code declares simulated data items that are pushed to the Azure event hub.

```
package com.learn;
public class TelemetryItem {

  private String id;
  private double temperature;
  private double pressure;
  private boolean isNormalPressure;
  private status temperatureStatus;

  static enum status {
    COOL,
    WARM,
    HOT
  }

  public TelemetryItem(double temperature, double pressure) {
    this.temperature = temperature;
    this.pressure = pressure;
  }

  public String getId() {
    return id;
  }

  public double getTemperature() {
    return temperature;
  }

  public double getPressure() {
    return pressure;
  }

  @Override
  public String toString() {
    return "TelemetryItem={ id=" + id + ",temperature=" + temperature +
",pressure= " + pressure + "}";
  }

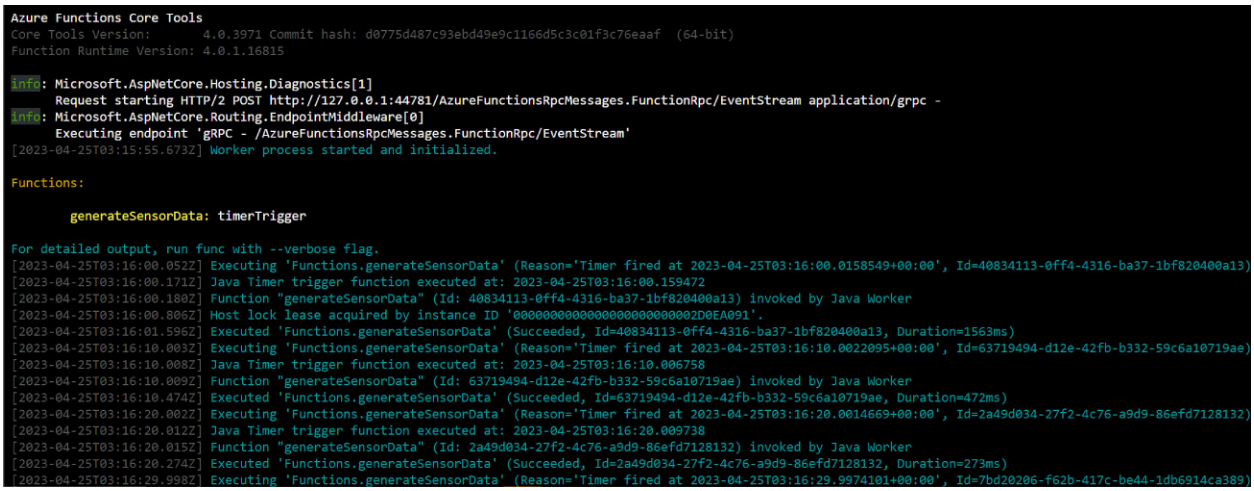
  public boolean isNormalPressure() {
    return isNormalPressure;
  }

  public void setNormalPressure(boolean isNormal) {
    this.isNormalPressure = isNormal;
  }
```

```
  public status getTemperatureStatus() {
    return temperatureStatus;
  }

  public void setTemperatureStatus(status temperatureStatus)
{
    this.temperatureStatus = temperatureStatus;
  }
}
```

5. In the telemetry-functions-producer/ directory, run the command below to build the function. A Build Success message appears in the console indicating the build was successful.
   `mvn clean package`
6. Run the command below to test the function and confirm it runs properly. The output shown indicates the function is running properly.
   `Mvn azure-functions:run`



(attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

7. Run the command below to deploy the FaaS producer function as an Azure Function App. Once the deployment is complete, the HTTP Trigger URLs are provided.
   `mvn azure-functions:deploy`

Create, Build, and Deploy the FaaS Consumer Function

Now that the FaaS producer function is built and deployed, the next step is to create an Azure FaaS function project for the FaaS consumer function. Like the FaaS producer function, a few resources need to be created for the FaaS consumer function. A storage account is created. The FaaS consumer function needs to connect to the Azure event hub. The FaaS consumer function also needs to connect to the Azure Cosmos DB datastore. The FaaS consumer function is built as a maven project and then deployed as an Azure Function App.

1. Run the commands below to create a storage account and the FaaS consumer function.

```
az storage account create\
  --resource-group $RESOURCE_GROUP\
  --name $STORAGE_ACCOUNT"c"\
  --sku Standard_LRS

az functionapp create\
  --resource-group $RESOURCE_GROUP\
```

```
  --name $FUNCTION_APP"-c"\
  --storage-account $STORAGE_ACCOUNT"c"\
  --consumption-plan-location $LOCATION\
  --runtime java\
  --functions-version 4
```

2. Use the commands below to obtain the connection strings for the storage account and the datastore. These values, as well as the datastore information, are required for the consumer function.

```
AZURE_WEB_JOBS_STORAGE=$(\
az storage account show-connection-string\
  --resource-group $RESOURCE_GROUP\
  --name $STORAGE_ACCOUNT"c"\
  --query connectionString \
  --output tsv)

COSMOS_DB_CONNECTION_STRING=$(\
az cosmosdb keys list\
  --resource-group $RESOURCE_GROUP\
  --name $COSMOS_DB_ACCOUNT\
  --type connection-strings\
  --query 'connectionStrings[0].connectionString'\
  --output tsv)
```

3. Run the command below to obtain the event hub connection string.

```
EVENT_HUB_CONNECTION_STRING=$(\
az eventhubs eventhub authorization-rule keys list\
  --resource-group $RESOURCE_GROUP\
  --name $EVENT HUB_AUTHORIZATION_RULE\
  --eventhub-name $EVENT_HUB_NAME\
  --namespace-name $EVENT_HUB_NAMESPACE\
  --query primaryConnectionString\
  --output tsv)
```

4. Run the commands below to display the connection strings that were created in the previous steps. Make a note of these connection strings as they will be used later.

```
echo $AZURE_WEB_JOBS_STORAGE
echo $EVENT_HUB_CONNECTION_STRING
echo $COSMOS_DB_CONNECTION_STRING
```

5. The connection strings that were generated for the Azure Web Jobs Storage, Event Hub, and Azure Cosmos DB datastore in the previous step need to be added as application settings to the Azure Function App account in the command. Run the command below with these values inserted to create the Function App. A notification to the console indicates that the Function App was built successfully.

```
az functionapp config appsettings set\
  --resource-group $RESOURCE_GROUP\
  --name $FUNCTION_APP"-c"\
  --settings\
    AzureWebJobsStorage=$AZURE_WEB_JOBS_STORAGE\
    EventHubConnectionString=$EVENT_HUB_CONNECTION_STRING\
    CosmosDBConnectionString=$COSMOS_DB_CONNECTION_STRING
```

Build and Deploy the FaaS Consumer Function

Now that the Azure resources, such as the event hub, Azure Function App, Storage account, and Azure Cosmos DB, have been created and configured, the next step is to create an Azure FaaS function project for the FaaS

producer function. Maven is used to build the project.

1. Run the command below to create the function project for the FaaS consumer function. The telemetry-functions-consumer/ directory is generated along with the files for the project.
```
mvn archetype:generate -batch-mode\
  -DarchetypeGroupId=com.microsoft.azure
  -DarchetypeArtifactId=azure-functions-archetype
  -DappName=$FUNCTION_APP"-c" \
  -DresourceGroup=$RESOURCE_GROUP
  -DappRegion=$LOCATION\
  -DappServicePlanName=$LOCATION"plan" \
  -Dgroupid=com.learn \
  -DartifactId=telemetry-functions-consumer
```
2. Navigate into the *telemetry-functions-consumer/* directory. Run the command below to update the local settings for local execution with the command. These settings are added to the local.settings.json file.
```
func azure functionapp fetch -app-settings \ $FUNCTION_APP”-p”
```
3. Navigate to the src/main/java/com/learn/ directory. Replace all the code in Function.java with the code shown below. The code declares a Function that establishes connections to the Azure event hub and Azure Cosmos DB.
```
package com.learn;
import com.learn.TelemetryItem.status;
import com.microsoft.azure.functions.annotation.FunctionName;
import com.microsoft.azure.functions.ExecutionContext;
import com.microsoft.azure.functions.OutputBinding;
import com.microsoft.azure.functions.annotation.Cardinality;
import com.microsoft.azure.functions.annotation.CosmosDBOutput;
import com.microsoft.azure.functions.annotation.EventHubTrigger;

public class Function
{
  @FunctionName("processSensorData")
  public void processSensorData(
  @EventHubTrigger(
  name ="msg",
  eventHubName = "", // blank because the value is included in the connection
string
  cardinality = Cardinality.ONE,
  connection = "EventHubConnectionString")
  TelemetryItem item,
  @CosmosDBOutput(
  name = "databaseOutput",
  databaseName ="TelemetryDb",
  collectionName ="TelemetryInfo",
  connectionStringSetting = "Cosmos DB ConnectionString")
  OutputBinding <TelemetryItem> document,
  final ExecutionContext context) {

    context.getLogger().info("Event hub message received: " + item.toString());
```

```
    if (item.getPressure() > 30) {
      item.setNormalPressure(false);
    }
    else {
      item.setNormalPressure(true);
    }

    if (item.getTemperature() < 40) {
      item.setTemperatureStatus(status.COOL);
    }
    else if (item.getTemperature() > 90) {
      item.setTemperatureStatus(status.HOT);
    }
    else {
      item.setTemperatureStatus(status.WARM);
    }
    document.setValue(item);
  }
```

4. Create the file TelemetryItem.java and add the code shown below. The code declares data items that are received from the Azure event bus and stored in the Azure Cosmos DB datastore.

```
package com.learn;

public class TelemetryItem {
    private String id;
    private double temperature;
    private double pressure;
    private boolean isNormalPressure;
    private status temperatureStatus;
    static enum status {
        COOL,
        WARM,"
        HOT
    }

    public TelemetryItem(double temperature,double pressure) {
        this.temperature = temperature;
        this.pressure = pressure;
    }
    public String getId() {
        return id;
    }
    public double getTemperature() {
        return temperature;
    }
    public double getPressure() {
        return pressure;
    }
    @Override"
    public String toString() {
        return \"TelemetryItem={id=" + id + \",temperature="
```

```
                + temperature + ",pressure=" + pressure + "}";
        }
        public boolean isNormalPressure() {
            return isNormalPressure;
        }
        public void setNormalPressure(boolean isNormal) {
            this.isNormalPressure = isNormal;
        }
        public status getTemperatureStatus() {
            return temperatureStatus;
        }
        public void setTemperatureStatus(status temperatureStatus) {
            this.temperatureStatus = temperatureStatus;
        }
    }
```

5. Navigate to the telemetry-functions-consumer/ directory and build the function with the command below. Once the build is complete, a Build Success message is displayed to the console to indicate the build was successful.
   `mvn clean package`
6. Run the command below to test if the function runs properly.
   `mvn azure-functions:run`
7. As mentioned earlier, the FaaS consumer function listens to events of a specific namespace in the Azure event hub and processes and stores them in an Azure Cosmos DB datastore. The events stored in the Azure Cosmos DB datastore can be checked by visiting the Azure Cosmos DB page in the Azure Portal. In the Azure Portal, navigate to the Azure Cosmos DB page. On the left menu, click on Data Explorer, click on the TelemetryInfo tab, and then click on Items expanded from TelemetryInfo to view the data. Data will continue to be sent to the Azure Cosmos DB datastore, which can be viewed in real time.
8. Run the command below to deploy the functions to Azure Functions. Once the deployment completes, a Build Success message appears in the console.
   `mvn azure-functions:deploy`

Test the Deployed FaaS Producer and Consumer Functions

Now that both the FaaS producer and consumer functions are successfully deployed as Azure Function Apps, data continues to be simulated, pushed to the event hub, and then stored in the datastore. The next step is to test them and evaluate their performance. The FaaS producer function continues to send telemetry data to the Azure event hub while the FaaS consumer function continues to process events from the Azure event hub and store them in the Azure Cosmos DB datastore. The activities and performance of these two functions can be viewed in Application Insights in the Azure Portal.

Evaluate the FaaS Producer Function

The first step is to inspect the application map for both FaaS functions. Application maps represent the logical structure of a distributed application. Individual components of the application are identified by their “roleName” property and are independently deployed. These components are represented as circles, called “nodes” on the map. HTTP calls between nodes are represented as arrows connecting the source node to the target node. Application maps help identify performance bottlenecks or failure hotspots across all components via alerts.

The application map (Figure 12.41) displays the logic structure of the FaaS producer function and Azure event bus. It shows the FaaS producer function as a node connected to an Azure event hub, also as a node. The application map for the FaaS producer function shows that there is a dependency between the FaaS producer

function and the Azure event hub. It also shows the FaaS producer function is not dependent on the FaaS consumer function nor the Azure Cosmos DB datastore.

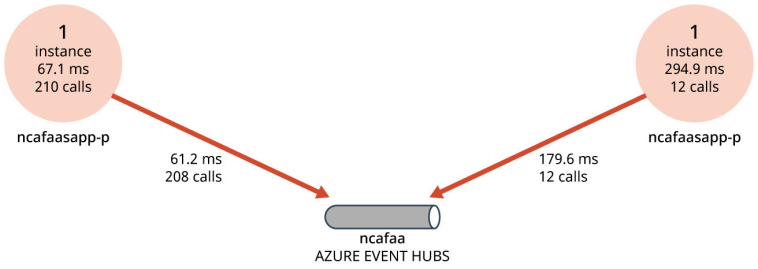


Figure 12.41 This application map shows the FaaS producer function as a node connected to an Azure event hub, also as a node. (Used with permission from Microsoft)

The application map shown in Figure 12.42 displays the logic structure of the FaaS consumer function and Azure Cosmos DB datastore. It shows the FaaS consumer function as a node connected to an Azure Cosmos DB datastore, also as a node. The application map for the FaaS consumer function shows that there is a dependency between the FaaS consumer function and the Azure Cosmos DB datastore. It also shows the FaaS consumer function is not dependent on the FaaS producer function nor the Azure event hub.

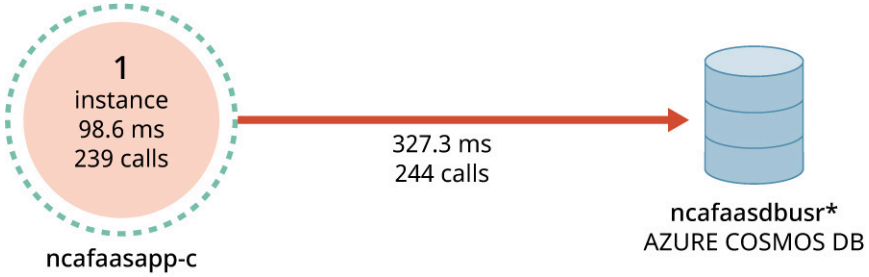


Figure 12.42 This application map shows the FaaS consumer function as a node connected to an Azure Cosmos DB datastore, also as a node. (Used with permission from Microsoft)

Chapter Review

Key Terms

Advanced Package Tool (apt) used to install new packages and update existing packages
Amazon Elastic Compute Cloud (EC2) cloud based, on-demand, compute platform that can be auto-scaled to meet demand
Amazon Web Service (AWS) Amazon cloud computing platform that offers a wide range of services that allow customers to build, deploy, and manage applications and services in the cloud
API gateway used to route traffic between client apps and microservices
AWS CLI command-line interface used to manage Amazon cloud services
AWS Portal web-based console that allows customers to manage their cloud services and subscriptions
Azure cloud computing platform that offers a wide range of services that allow customers to build, deploy, and manage applications and services in the cloud
Azure Function App event-driven serverless compute platform without provision or managing infrastructure
Azure Portal web-based console that allows customers to manage their cloud services and subscriptions
Backend as a Service (BaaS) third-party service that can be integrated with an application where the BaaS code does not need to be managed, and there are no servers
bare metal server high-performance cloud server that is composed of a single-tenant, nonvirtualized physical server
cloud computing delivery of computing services, such as databases, software, analytics, and storage over the Internet
cloud mashup combines multiple cloud applications or services with shared datasets and integrated functionalities
cloud-native application uses microservices architecture that takes full advantage of the benefits of the cloud, such as scalability and on-demand services
community cloud cloud deployment model where resources are only accessible by a selected group of organizations
Container as a Service (CaaS) type of IaaS specifically geared toward efficiently deploying, running, scaling, and managing a single application using a container-based virtualization
container image binary state that is built by "layering" an application and its dependencies on top of the required "layer" of binaries and libraries of the OS where there is a parent-child relationship between image layers
containerization packaging of a standardized unit of software that isolates an application enabling it to run independently of physical resources
continuous delivery automates the deployment stage
continuous integration automates the application compile, build, package, and testing process, enabling it to run independent of physical resources, which allows for a more consistent integration process
continuous integration and continuous deployment (CI/CD) set of DevOps operating principles that enable development teams to focus on rapid and frequent integrations of new code into the application in development as well as fast and frequent delivery or deployment of new iterations of the application
DevOps methodology that combines Agile software development and IT operations to deliver software faster and of higher quality
disaster recovery ability of the application to recover from rare but major incidents: nontransient, wide-scale failures, such as service disruption that affects an entire region
Django open-source web application framework
Docker open-source platform that creates, deploys, and manages containers using a common operating system; it isolates resources allowing multiple containers to use the same OS without contention
Docker Compose tool used for defining, via descriptor files, and managing applications that consist of multiple Docker containers

Docker Engine open-source containerization platform used to build Docker images and manage Docker containers

enterprise service bus (ESB) implements a bus-like communication system between interacting service providers and service consumers

event-driven architecture architecture where functions are invoked by an event

Express.js back-end Node web application framework used to implement RESTFul APIs

Function as a Service (FaaS) compute platform for serverless where developers can run functions that are single units of deployment of the code, which is run by events

GitHub cloud-based developer platform used for software development, collaboration, and security

GitLab cloud-based DevOps platform used to monitor, test, and deploy application code

high availability ability for an application to continue running in a healthy state without significant downtime

high-performance computing (HPC) when a workload is compute- and memory-intensive

Homebrew open-source package management system used to install and manage packages on MacOS and Linux operating systems

hybrid cloud combines private and public clouds, and bare metal or virtual environments

Infrastructure as a Service (IaaS) on-demand access to a cloud-hosted computing infrastructure that includes servers, storage, and network resources

integrated development environment (IDE) software application that provides developers access to resources (e.g., source code editor, build automation tools, debuggers) to design, develop, debug, and test programs

Kuard (Kubernetes Up and Running Demo) basic dashboard highlighting exposed application metrics

Kubectl Kubernetes command-line tool that is used to manage Kubernetes clusters

kubelet responsible for scheduling and making sure applications deployed within the worker nodes are running properly

Kubernetes open-source orchestration platform used for automating deployment, scaling, and management of container-based workloads

monolith application that conforms to a monolithic architecture

monolithic architecture not designed to leverage an orchestration platform and requires the use of load balancers

Next.js open-source React framework used to create full-stack web applications

object-relational mapping (ORM) technique to convert a data object in the Express.js code to a table in the PostgreSQL relational database

Octant open-source web interface for Kubernetes that is used to inspect Kubernetes clusters and applications deployed in them

orchestration platform helps schedule microservices and containers, and optimizes the use of the computing resource

Platform as a Service (PaaS) on-demand access to a cloud-hosted platform for developing, running, and managing applications by prepackaging middleware, language runtimes, and tools as containers

pod small, logical unit that runs a container in a worker node in the Kubernetes cluster

Prisma Node ORM used to map data objects to tables in a relational database

private cloud cloud deployment model where resources are dedicated to one customer such as a single organization for internal use where cloud resources are accessed and managed by an organization

public cloud cloud deployment model where resources are remotely accessible by anyone offered through subscriptions or on-demand pricing plans

remote procedure call (RPC) request-response protocol used by one program to request services from other programs over a network in a distributed computing environment

representational state transfer (REST) architectural style for providing standards between resources so they can communicate with each other over the Web

resiliency ability of a system to recover from failures and continue to function

resource group container that holds related resources used in a cloud solution
Sequelize used as an ORM to convert data objects in the Express.js code to data records in the inventory table the database
serverless computing cloud-native development model where developers can build and run applications but are not responsible for provisioning, maintaining, and scaling the server infrastructure as this is outsourced to the cloud service provider (CSP)
service-oriented architecture (SOA) requires that conforming applications be structured as discrete, reusable, and interoperable services that communicate through an enterprise service bus (ESB)
Software as a Service (SaaS) cloud-hosted, ready-to-use software or application that end users access with a client (e.g., web browser, desktop client, mobile app) via a subscription model
Tanzu modular, application platform that provides a rich set of developer tools and a pre-paved path to production to build and deploy applications quickly and securely on any compliant public cloud or on-premises Kubernetes cluster
Terraform open-source infrastructure as code tool used to build, update, and version cloud and on-premises resources
unikernel single-purpose machine image that is compile-time specialized into a dedicated stand-alone kernel with only the libraries needed to run the application
virtual machine (VM) virtualization of a computer system
web service unit of software that is available as a resource over the Internet
workload application being run on Kubernetes

Summary

12.1 Introduction to Cloud-Native Applications

- Cloud-based applications leverage cloud platforms; however, they do not take full advantage of the inherent characteristics of the cloud like cloud-native applications. Monoliths can be deployed in the cloud as cloud-based applications but can be converted to leverage a microservices architecture as cloud-native applications.
- Microservices are self-contained software units. Components were used before in older architectures such as web service components and SOA. However, the way components between these architectures communicate are quite different.
- There are several challenges associated with implementing a microservices architecture. Building microservices to interoperate with each other, as well as testing, debugging, versioning, deploying, logging, and monitoring microservices add additional challenges.
- Features of cloud-native applications include they are microservices-based, container-based, API-based, and are dynamically orchestrated to scale and optimize the use of computing resources.
- DevOps is a methodology that combines Agile software engineering and IT operations to release higher-quality software faster. CI/CD is a DevOps approach that provides steps to build, package, test, deploy, and release software components with the goal to automate as many steps as possible in the process.
- Some benefits of cloud-native applications are they are cost-effective, easily scalable, portable because they are containerized, reliable, and easier to manage through a DevOps pipeline using CI/CD.
- Best practices for creating cloud-native applications include automate as much of the development life cycle as possible, monitor the development environment as well as the use of microservices, document the services making it easier to integrate with them, and schedule incremental and reversible releases.
- Microservices are containerized. Docker is an open-source platform that creates, deploys, and manages microservices containers.
- Kubernetes is an open-source orchestration platform that is used for automating the deployment, scaling, and managing the health of microservices container-based workloads.
- There are various categories of tools that can be used to develop cloud-native applications, including IaC tools (e.g., Terraform), cloud-based DevOps platforms (e.g., GitLab, GitHub), containerization software products and platforms (e.g., Red Hat OpenShift), and developer tools to develop and deploy applications

quickly (e.g., Tanzu, Node).

12.2 Cloud-Based and Cloud-Native Applications Deployment Technologies

- There are three service models (IaaS, PaaS, and SaaS) and four deployment models (public cloud, private cloud, community cloud, and hybrid cloud) that make up the cloud computing model.
- Cloud service models (IaaS, PaaS, and SaaS) combine and change the way IT resources are consumed by cloud customers with the goal of providing a more economical solution as they can be scaled on-demand at a predictable cost. These cloud service models present a spectrum from complex and high effort of investment to simple and low effort of investment depending on business needs.
- Cloud deployment models (public cloud, private cloud, community cloud, and hybrid cloud) are where cloud services are implemented and made available to end users. Cloud computing deployments work on the principle of virtualizing compute resources. Cloud deployment models can be combined (e.g., hybrid clouds) to offer varying degrees of flexibility in terms of how cloud resources are accessed and managed.
- There are several cloud deployment options to choose from, including bare metal, VMs, unikernels, and containers that are offered in various cloud service models. Serverless computing is another cloud deployment option where developers can build and run applications in the cloud but are not responsible for provisioning, maintaining, and scaling the server infrastructure the application runs on.
- Given there are several cloud deployment technology options, an understanding of the use cases for these technologies can help customers choose the “best fit” options that satisfy business needs. For example, customers who need to have dedicated server with a high level of control would choose bare metal servers, whereas customers who need a solution with an immutable infrastructure would choose unikernels.
- Several tools are available for making implementing services using cloud deployment technologies easier. For example, IaC tools are useful to automate various steps in maintaining operating systems when adopting IaaS. In contrast, tools that automate software development process tasks are more suitable when adopting PaaS.
- There are some key considerations when selecting cloud deployment technologies, which include assessing whether cloud deployment technologies satisfy business needs, understand workload/service requirements, and ensuring they satisfy application and service requirements.
- Some things to consider when assessing whether cloud deployment technology options fit business needs are cost, architectural fit, performance, compliance, elasticity requirements, control requirements, and whether the cloud service provides lock-in.
- Cloud applications and services can be combined to form a cloud mashup. Cloud mashups can give organizations a competitive advantage by bringing competitive solutions to the market quickly.
- Some keys to successfully utilizing cloud deployment technologies include identifying currently available cloud deployment technologies and how they provide a fit to delivering cloud services, determining the use cases for these cloud deployment technologies and identifying tools available to help deliver cloud services, and assessing these cloud deployment technologies and selecting those that best fit business needs.

12.3 Example PaaS and FaaS Deployments of Cloud-Native Applications

- A cloud-native application was created and deployed to a PaaS platform. The cloud-native application consists of two microservices, both of which communicate with a single datastore. The cloud service provider used is Microsoft Azure. Although Microsoft Azure was used in this example, alternatively, other cloud services providers (e.g., AWS, IBM Cloud, GCP) can be used.
- A suite of tools that monitor Kubernetes clusters and deployed applications was configured and deployed in a PaaS platform. The cloud service provider used is Amazon AWS. Although AWS was used in this example, alternatively, other cloud services providers (e.g., Microsoft Azure, IBM Cloud, GCP) can be used.
- A distributed application was created that consists of two FaaS functions, an event bus, and a datastore.

The FaaS functions were deployed in a serverless platform. The cloud service provider used is Microsoft Azure. Although Microsoft Azure was used in this example, alternatively, other cloud services providers (e.g., AWS, IBM Cloud, GCP) can be used.

Review Questions

1. In the four key principles of cloud-native development, what architecture should be used?
   a. monolithic programs
   b. Model-View-Controller (MVC)
   c. microservices
   d. event driven

2. What is one of the four key principles of cloud-native development?
   a. Use a monolithic architecture to ensure all business functions are captured in the software.
   b. Use the waterfall methodology to ensure that the application meets all system requirements prior to initial deployment.
   c. Use containers to package and isolate microservices for deployment.
   d. Use languages that will run on both Windows and Linux environments.

3. What is the difference between monolithic and microservices architectures?
   a. In monolithic architectures, capabilities are loosely coupled, while in microservices architectures, components are highly coupled.
   b. In monolithic architectures, a waterfall software methodology is employed, while microservices are developed in an Agile software methodology.
   c. Monolithic architectures are the most popular cloud application architecture, while microservices is the most popular desktop application architecture.
   d. In monolithic architecture, an application is built as a unified unit to perform multiple business functions and is self-contained, while in microservices architecture, the application is broken down into small, independent components or services that perform a single business function.

4. What is the difference between components in monoliths and microservices?
   a. Components in monoliths are generally libraires that are compiled and linked into a single program or process. Components that are microservices are “out-of-process” components that communicate using a web service request or RPC.
   b. Components in monoliths are individual classes. Components in microservices are system resources.
   c. Components in monoliths are small, individual executables all running in the background and the main program can call them. Components in microservices are libraries that the main program compiles and links in at build time.
   d. Components in monoliths are static libraries and always linked at build time. Components in microservices are dynamic libraries and always linked at runtime.

5. What term is defined as “architectural style for providing standards between resources so they can communicate with each other over the Web”?
   a. remote procedure call (RPC)
   b. representational state transfer (REST)
   c. service-oriented architecture (SOA)
   d. enterprise service bus (ESB)

6. What is the difference between SOA and microservices?

7. What are some of the features of cloud-native applications?

8. What are some of the benefits of cloud-native applications?

9. What are some of the best practices associated with cloud-native application development?

10. What are the various categories of tools used to develop cloud-native applications?

11. What is an example of a cloud service model defined by NIST?
- **a.** Infrastructure as a Service (IaaS)
- **b.** monolithic services
- **c.** microservices
- **d.** Function as a Service (FaaS)

12. What is an example of a cloud deployment model defined by NIST?
- **a.** household cloud
- **b.** private cloud
- **c.** business cloud
- **d.** AWS cloud

13. Which service provides on-demand access to a cloud-hosted platform for developing, running, and managing applications by prepackaging middleware, language runtimes, and tools as containers?
- **a.** Infrastructure as a Service (IaaS)
- **b.** Platform as a Service (PaaS)
- **c.** Software as a Service (SaaS)
- **d.** Function as a Service (FaaS)

14. What is the term for a type of IaaS specifically geared toward efficiently deploying, running, scaling, and managing a single application using a container-based virtualization?
- **a.** Containers as a Service (CaaS)
- **b.** Backend as a Service (BaaS)
- **c.** serverless computing
- **d.** cloud mashup

15. What is high-performance computing?
- **a.** a supercomputer with advanced cutting-edge hardware technologies
- **b.** a computer with multiple processors
- **c.** a program or workload that is compute- and memory-intensive
- **d.** a program that incorporates multithreading to carry out parallel computing to run faster

16. What is the relationship between cloud service models, cloud deployment models, and cloud deployment technologies?

17. Identify and explain the four IaaS deployment options used currently.

18. What is the difference between unikernels and VMs?

19. What is the difference between the PaaS and FaaS deployment options?

20. What are the four deployment options that can be used to deploy cloud-native applications?

21. What aspects of IaaS, PaaS, and FaaS deployment can be automated and how?

22. List five categories of tools used to automate cloud deployment?

23. What are some of the DevOps tools used to automate cloud deployment?

24. What are the key considerations on how cloud deployment technologies are selected?

25. What are cloud mashups and what specific concerns need to be considered when deploying them?

26. What is object-relational mapping?
- **a.** a container that holds related resources used in a cloud solution
- **b.** technique that converts a data object in a programming language to a table in a database

c. tools that enable data objects to be plotted on a front-end map for displaying
d. a software resource that allows a hashmap to be used to find what microservice contains what function calls

27. If the microservices already have a hostname and port, then why are ingress controllers needed?
a. The ingress controllers are what exposes the microservice to the outside world so it can receive requests from clients.
b. The ingress controllers are responsible for reporting system health and reports what microservices are up and running.
c. The ingress controllers are responsible for ensuring clients call the correct microservices.
d. The ingress controllers are what carries out the data transmission in and out of the microservices.

28. What is the AWS Portal?
a. Amazon cloud computing platform that offers a wide range of services that allow customers to build, deploy, and manage applications and services in the cloud
b. an open-source package management system used to install and manage packages on MacOS and Linux operating systems
c. a command-line interface used to manage Amazon cloud services
d. a web-based console that allows customers to manage their cloud services and subscriptions

29. What is Kubectl and what is it used for?
a. It is a command-line interface tool. It is used to interact with a Kubernetes cluster.
b. It is an open-source package management system used to install and manage packages on MacOS and Linux operating systems.
c. It is a basic dashboard highlighting exposed application metrics.
d. It is an open-source web interface for Kubernetes that is used to inspect Kubernetes clusters and applications deployed in them.

30. What application is a tool used to generate containers?
a. Homebrew
b. Octant
c. Docker Engine
d. Function App

31. What kind of information does the Kuard application provide?

32. In the FaaS example that we built, what is the role of the event bus? Why did we need to create this?

33. What does the application map do?

Conceptual Questions

1. Why is it difficult to efficiently allocate resource services in a cloud-native application?

2. Why is the microservices architecture more adaptable to a cloud architecture?

3. Why is it the case that cloud deployment technologies keep evolving? Provide examples to illustrate your answer.

4. What is the difference between BaaS and FaaS? Are FaaS and serverless equivalent in a BaaS context?

5. In the first sample cloud-native application we built, a Dockerfile and a Kubernetes manifest file were created for each microservice. Why do we need both files in this application? Which one is used for what purpose?

6. Focusing our attention on the Kubernetes manifest files, there are two for each microservice: a deployment manifest file and a service manifest file. What are the differences between these two types of

manifest files? What are they used for?

7. In the second example, an AWS EC2 Instance VM was provisioned. Several steps were taken to install package management tools (e.g., Homebrew, build-essentials), the Docker engine, and any other dependencies needed for the application to run. Why did we need to take these steps?
8. In the third example, we built two FaaS functions that were deployed that make up a cloud-native application. Contrast this to the first example we built, where we built microservices that were also deployed in Azure. Although the cloud service providers provide editors in the portal to implement and deploy FaaS functions directly within the portal, in this case we used a build tool, maven, to build the code for the FaaS functions and deploy them remotely. Why is this not the same as the first example? What are the major differences between these two approaches where the first example is using the PaaS deployment option, and the third example is using the FaaS deployment option?

Practice Exercises

1. Explain how a web-based cloud-native application works as opposed to a traditional web application.
2. Give a practical example of a solution implementation that illustrates the differences between SOA and microservices.
3. Explain how CI/CD works in combination with other DevOps tools to support the development of cloud-native applications.
4. Explain how CaaS deployment works.
5. Give a practical example of a solution deployment that illustrates the differences between PaaS and FaaS/serverless.
6. Explain how CI/CD works in combination with other DevOps tools to support the deployment of cloud-native applications using either Containers/CaaS, PaaS, or FaaS.
7. FaaS/serverless is considered to be a deployment technology; however, because it appears that code needs to be written in a different way to specify services as functions, why is FaaS/serverless not considered as a separate architectural style?
8. Follow the AWS tutorial to build a serverless application (https://openstax.org/r/76bldserapp) in AWS.
9. Follow the Google Cloud tutorial to build serverless function (https://openstax.org/r/76bldservfunc) in Google Cloud.
10. Follow the Google Cloud tutorial to build a containerized application (https://openstax.org/r/76bldcontapp) that receives events using a messaging service.

Problem Set A

1. Read this seminal article about microservices (https://openstax.org/r/76artmicroserv) and report on the pros and cons based on its content.
2. Demonstrate how the twelve-factor application principles (https://openstax.org/r/7612factapp) may be used to develop cloud-native applications.
3. Read this article about serverless architectures (https://openstax.org/r/76artservnoarc) and report on the pros and cons of FaaS/serverless based on its content.
4. Read this article documenting several use cases for FaaS/serverless deployment (https://openstax.org/r/76artdocFaaS) and think about when to use it. Perform some research on the Internet and provide additional use cases that make a case for selecting FaaS/serverless deployment technology as opposed to Containers/CaaS or PaaS.

5. In the first example we built, you worked with Kubernetes, which required you to create deployment and service manifest files. These manifest files are YAML files. Read the IBM tutorial (https://openstax.org/r/76IBMtut) to enhance your knowledge on the structure of YAML files.
6. Follow this IBM tutorial (https://openstax.org/r/76tut1kuber) or this IBM tutorial (https://openstax.org/r/76tut2kuber) or to create and deploy an application in Kubernetes.

Problem Set B

1. Convert a monolithic application of your choice to microservices.
2. Demonstrate practically how Chaos Monkey (https://openstax.org/r/76chaosmonk) is used to test a cloud-native application of your choice. Follow the Chaos Monkey setup tutorial (https://openstax.org/r/76chaosmtut) and refer to the Chaos Monkey documentation (https://openstax.org/r/76chaosmdoc) to solve this question.
3. Explain how IaC is used in a practical cloud-native application development context.
4. Deploy a simple cloud-native application using VMWare Tanzu. Document your steps and compare them to the steps you took earlier to deploy a simple application using PaaS without VMWare Tanzu in an earlier question.
5. Obtain the code of the back-end implementation (in Python) of this RESTful microservice (https://openstax.org/r/76RESTmicro) for an order resource. Deploy this cloud-native application on a public cloud of your choice (i.e., either AWS, GCP, IBM Cloud, or Microsoft Azure).
6. Follow this IBM tutorial to gain an understanding of how Kubernetes clusters work (https://openstax.org/r/76kubclust) by debugging and logging applications deployed in Kubernetes.
7. Microservices often utilize message streaming and brokers to share data in real time across the system. Research and explain the basic operations of such tools and how they can be used in microservices applications.

Thought Provokers

1. Consider our start-up company that is 100% committed to leveraging innovative technologies as a business growth facilitator. Describe how it can best take advantage of cloud-native application development to create products or services that can generate business. Give precise examples and explain how the start-up would be able to ensure the scalability of the resulting business (i.e., keep sustaining the cost of doing business while increasing its number of customers).
2. Consider our start-up company that is 100% committed to leveraging innovative technologies as a business growth facilitator. Describe how it can best take advantage of cloud deployment technologies to create products or services that can generate business. Give precise examples and explain how the start-up would be able to ensure the scalability of the resulting business (i.e., keep sustaining the cost of doing business while increasing its number of customers).

Labs

1. Follow the tutorial provided at to develop a sample cloud-native application (https://openstax.org/r/76CLNATapp). This tutorial creates a cloud-native application using Python. With this first tutorial, you should be able to create an Azure function app with an HTTP-triggered function. Then, follow this tutorial to connect the Azure function to a datastore (https://openstax.org/r/76connAzure). Finally, follow this tutorial to extend your cloud-native application so that your function app sends messages to an Azure storage (https://openstax.org/r/76CLappFunc). There are other programming language options (https://openstax.org/r/76AltProgLang) to choose from. It is fine to pick another tutorial available on the Internet to achieve the same goal (see an example (https://openstax.org/r/76Alttutorial)).

2. Build a hospital outpatient department (OPD) cloud-native application based on these approaches (https://openstax.org/r/76approachCN) and code. To get started, follow these instructions (https://openstax.org/r/76instructMSplt) to get the application up and running locally. After the application is up and running locally successfully, use an approach you have learned in this chapter.
3. Perform some research on the Internet and locate a documented example (with code) that illustrates how to migrate a legacy/monolith application to cloud-native. Follow the steps provided in the documented example you found to build and deploy the corresponding cloud-native application.
4. Follow the Microsoft Azure tutorial (https://openstax.org/r/76MSaztutr) to enable rapid development by creating a CI/CD pipeline using GitHub Actions and Azure Pipelines. Refer to documentation about GitHub actions (https://docs.github.com/en/actions) and Azure Pipelines (https://openstax.org/r/76AZpipe) for additional information. Once you complete the tutorial, apply what you learned to create a CI/CD pipeline for the first example you built in this module.



13

Hybrid Multicloud Digital Solutions Development

Figure 13.1 Business requirements determine cloud implementation and deployment strategies, creating an interconnected system that is based on selected application workloads, architecture patterns, and technologies. (credit: modification of “Cityscape” by Romain Guy/Flickr, Public Domain)

Chapter Outline

13.1 Hybrid Multicloud Solutions and Cloud Mashups
13.2 Big Cloud IaaS Mainstream Capabilities
13.3 Big Cloud PaaS Mainstream Capabilities
13.4 Towards Intelligent Autonomous Networked Super Systems

Introduction

By combining different cloud systems or connecting a cloud node with a private network, companies can develop solution architectures that give them all the benefits of clouds and allow them to scale applications without being tied down to one cloud vendor, which also ensures reliability and overall robustness.

TechWorks is a start-up company dedicated to leveraging innovative technologies as part of its repeatable business model and as a growth facilitator. Their vision is to develop next-generation secure, supersociety intelligent autonomous solutions. To achieve this, TechWorks needs a solution architecture that aligns with its goals while staying within budget.

The company faces a choice between building and maintaining a local computing environment or utilizing third-party cloud services. Setting up a local environment requires a substantial initial investment dedicated to building and maintaining infrastructure. This option, while providing full access to develop, scale, and monitor their applications, was deemed too costly for TechWorks.

Instead, TechWorks opted for a third-party cloud system to deploy and maintain their applications. By utilizing a cloud architecture, TechWorks benefits from virtual capabilities such as storage, networking, and computing without the heavy up-front costs. This approach helps TechWorks save on development costs while maintaining flexibility and scalability.

However, cloud-based solutions come with trade-offs, including limited access and control over the provider's infrastructure, which may introduce vulnerabilities and dependencies. TechWorks needs to address several

critical questions: How will they manage their budget to ensure effective application deployment and maintenance? How will they handle rapid growth in customer data traffic? What strategies will they employ to respond to data breaches and system maintenance?

Before the widespread adoption of cloud computing, many large companies such as Samsung and IBM relied on extensive physical infrastructures, including multiple self-developed data centers. The advent of cloud technologies has transformed this landscape, enabling the virtualization of resources and presenting opportunities for companies to transition to cloud environments or hybrid models that combine cloud and physical servers.

For new tech businesses without access to expensive local IT infrastructure, relying entirely on cloud resources is a common strategy. Choosing a well-designed cloud architecture is crucial for long-term cost savings, flexibility, scalability, and agility. Additionally, a secure cloud architecture can help mitigate potential cyber threats.

For instance, relying on a single cloud infrastructure for development, servicing, and deployment exposes companies to significant risks in the event of a data breach. TechWorks decided to adopt a multicloud architecture, distributing responsibilities across different cloud systems for application communications, customer services, access control, and resource management. This approach enhances security and reduces dependency on a single provider.

The previous chapter focused on cloud-native applications that leverage microservices-based architectures. In that case, individual workloads are implemented as services that may or may not reside on the cloud. In this chapter, we will explore how various cloud-based architectures are developed by combining specific infrastructure as a service (IaaS) and platform as a service (PaaS) capabilities and computing environments to accelerate the delivery of desired functionalities and benefits. We will also examine how TechWorks, as a start-up, evaluates and implements cloud architectures to build solutions that meet their needs. Through examples utilizing cloud-based PaaS, such as IoT frameworks for temperature data collection and machine learning services for predictive analysis, we will demonstrate the profound impact of cloud deployment and workload implementation strategies on operational excellence and competitive advantage.

13.1 Hybrid Multicloud Solutions and Cloud Mashups

Learning Objectives

By the end of this section, you will be able to:

- Define and differentiate hybrid and multicloud infrastructure deployments
- Understand the importance of cloud mashups
- Understand how to accelerate the creation of solutions using cloud infrastructure and platform services

Cloud computing provides businesses with more options that help them deploy application workloads more effectively. Application workloads may include clients and servers in traditional client-server architectures, nodes in peer-to-peer (P2P) architectures, or services in microservice architectures. When it comes to cloud architecture, organizations need to determine which application workload should run in the cloud. Organizations taking advantage of cloud computing environments should ensure that it does not introduce delays in solution delivery and maintenance.

Hybrid and Multicloud Solutions

For large-scale applications, businesses look for better ways to manage users and queries without clogging a single cloud system. In this section, we will review different cloud solutions that are built upon combining a public cloud and a private cloud.

THINK IT THROUGH

Cybersecurity and Cloud-Based Architectures

Every year, organizations around the world are hit with major cyberattacks that affect millions and even billions of private accounts and records. To deter cyberattacks, organizations spend millions of dollars per year on cybersecurity. Cloud-based architectures are more difficult to secure and also partially rely on trusting security architectures provided by cloud service providers.

In your opinion, why are organizations becoming more dependent on cloud computing, considering added security risks?

Hybrid Cloud Solutions

One approach to managing cloud storage is to allow localization of infrastructure data while also utilizing third-party services for storing public data, hence implementations of hybrid cloud. Deployment of hybrid cloud infrastructures allows enterprises to manage flexible workloads using both public and private resources, opening the opportunity of having more management rights over sensitive data while also utilizing public clouds to deploy public applications.

To accommodate a broad range of needs, hybrid cloud solutions aim for a dynamic environment by allowing a wide range of options to combat potential issues regarding different deployment, communications, and management. Several approaches that the hybrid cloud offers toward these issues include cloud deployment, application communications, and application and infrastructure management.

Cloud Deployment

When it comes to deployment, organizations look at the structure of their own application's architecture and decide how their features should be deployed among different layers of storage. They can choose to shift their features to the cloud and keep the system on premise or move a fraction of their deployed code to the cloud while keeping cold data or backups on their private infrastructure. Hybrid cloud systems unlock access to on-site backup, which protects organizations from major financial losses or downtime in case of a system failure or a data breach. This also allows easier access to data for a remote workforce, as the data are not tied to a single location. In addition, a shift of deployed code or data system onto the cloud allows corporations to automate application updates and maintenance or scale their applications accordingly, depending on peak times, to maximize efficiency. When using a hybrid cloud system, scaling up during peak times or when demand spikes means simply paying for more cloud resources instead of having to expand local infrastructure. Likewise, a company may have the option to downscale during slower times, enabling a company to pay for only the resources it needs, when it needs them.

Application Communications

The main question when it comes to interconnecting one or several cloud and on-premises systems with varying providers and deployments is how to form effective communication between those systems. Businesses usually tackle the hybrid cloud network by implementing APIs as a means to communicate between different platforms, establishing secure network connections or VPNs to ensure dedicated protected connections, and encrypting data to mitigate data breaches. Note that the more complicated sets of systems the corporation manages require complex API architectures. Developers need to consider interoperability, data integration, and efficiency to design API systems that allow easy communication between systems while maximizing efficiency.

Application and Infrastructure Management

With an interconnected web of cloud and on-premises systems, developers need to find a way to effectively

monitor and keep up the performance of their resources. Because of the investment of multiple cloud systems from different providers, there will be different documentation and complex operational overheads. Developers need a way to simplify administration by centralizing control of all systems either through a unified tool (e.g., Azure Arc and CloudCenter Suite) or robust architecture. Ultimately, with a hybrid cloud system, the company has full control over its data and where they are housed, making it easier to make informed choices about data security. Careful planning of management for a hybrid cloud structure allows developers to easily control the scalability, flexibility, security, and cost of their hybrid system.

The pros and cons of a hybrid cloud solution must be weighed against the needs and priorities of the organization that intends to use the solution. Typically, a hybrid cloud meets a broad range of needs, including flexibility and security, as follows:

- Easier access to data to better support the remote workforce. The organization has the flexibility to provide remote employees with on-demand access to data that are not tied to one central location.
- Reduced costs. When demand spikes, the organization can avoid capital expenditures to expand its infrastructure and instead pay only for the cloud resources it uses.
- Improved scalability and control. Increased automation allows the organization to adjust its cloud settings to respond automatically to changes in demand, as well as optimize performance and efficiency.
- Security and risk management. The organization has control over its data and improves its security by reducing the potential exposure of data. The organization can choose where to house its data and workloads, which makes it easier to implement security measures such as encryption, automation, access control, orchestration, and endpoint security.

INDUSTRY SPOTLIGHT

Leveraging Hybrid Cloud Models for Streaming

Hybrid cloud models are extremely common architectures for big consumer companies. Netflix, for example, in 2008, adopted the hybrid cloud model by combining on-premises database structures to store big movie files and then utilizing Amazon Simple Storage Service (Amazon S3) to distribute data across cloud servers. They can manage their resources through built-in tools such as AWS Local Zones, which bring services closer geographically to a user. It is necessary for Netflix, as a streaming service provider, to adopt hybrid cloud models due to the amount of traffic and geography it covers on a daily basis, as well as to ensure user's smooth movie-watching experience.

Multicloud Solutions

Similar to hybrid cloud structures, a **multicloud solution** involves meshing several different computing environments to form a flexible working environment. Multicloud's main difference is that it is exclusively a combination of more than one public or private cloud system, compared to hybrid cloud, which is a combination of cloud and on-premises. The benefits of multicloud systems include automation and scalability, risk reduction, competitive pricing, and robust security.

Automation and Scalability

One of the cloud system's biggest advantages is the system's reliance on cloud infrastructure, which allows developers to effectively distribute workloads among infrastructures and leverage resources based on geographical patterns and peak usage time. In the long run, collecting data regarding the application's performance also allows complete automation in managing the multicloud infrastructure, which allows for convenience in controlling and optimizing the company's application.

One of the reasons why cloud systems became increasingly popular over the years is because of the low cost of acquiring and maintaining cloud systems. With a multicloud system, an organization is motivated to shop

around and choose multiple vendors for the best price. Different providers offer different services for different prices; there are enough cloud service providers today that an organization typically has many options to choose from.

Risk Reduction

By having access to multiple cloud vendors, the utilization of multicloud structures allows companies to maintain uptime in cases of data breach or system failure. If one vendor fails, the organization can switch to a different one. A vendor can also act as a complete backup in case the system needs a complete reboot.

In a multicloud system, each cloud vendor manages its infrastructure. Services such as AWS and Azure offer access, keys, and secured network management. Multicloud architecture leverages this difference in security features between vendors to act as layers of a secured system, preventing a complete breakdown of the environment.

Challenges of Multicloud Systems

With all the benefits that multicloud solutions offer, some challenges must be considered to mitigate risk and optimize the budget. Similar to hybrid cloud solutions, multicloud system networks can be hard to design due to the increasing complexity of the network as multiple cloud systems are connected. Furthermore, the complete reliance on cloud infrastructure requires dependence on the vendor's services and constraints, making it hard to unify systems or migrate workloads between different clouds. Organizations must consider these problems to mitigate risk and maximize profit when developing multicloud solutions. Typically, users rely on both hybrid and multicloud systems rather than selecting one over the other.

Cloud Mashups

A **mashup** typically describes a web-based application hybrid that combines features from two or more web sources to create a new service. These features communicate with the use of API, which are sets of protocols and tools that serve as a middleman between different software applications, allowing information to transfer between them. Cloud utilization allows complete virtualization of these multiservice web applications through vendors offering centralized tools and operations that support assembling and managing multiple sources; hence, the emergence of cloud mashups.

To understand cloud mashups, consider how news websites function. News providers typically get weather updates from Weather.com (or other sites) and gather other information, such as updates about stocks, shares, currency rates, and even additional news items from sites such as Reuters. The end product is a practical example of a mashup of multiple component parts. Cloud mashups have become increasingly popular over the years because of the plethora of public information that the service offers and the convenience of managing services completely through the cloud.

Cloud mashups effectively allow developers to pick and combine different information from public sources and make their applications. It is extremely beneficial due to how much it enhances users' experience while promoting collaboration between services. However, due to its dependency on the API structure of the service, the discontinuation of a service can cause many applications to break down. This lack of scalability becomes a concern for cloud mashups, and constant upkeep is necessary for applications that depend on using public services as a means to collect and display information.

Leveraging Cloud Services to Implement Cloud Applications' Workloads

Hybrid multiaccess computing ensures network scalability by introducing guidelines that constrain distributed applications to exhibit low latency and consume less power. This can be best achieved by locating and operating workloads at the mobile or network edge, which means maximizing communication performance between the local network or device endpoint onto the Internet. Products surrounding the IoT and computer infrastructures that rely on minimizing latency or collecting real-time data would benefit greatly from this. Furthermore, edge computing embedded in hybrid and multicloud models helps with encrypting data before

sending them onto the cloud network, minimizing risks of data breach and complying with privacy regulations.

Telecom services providers are slowly becoming cloud service providers in an effort to make cloud resources more readily available on demand at the edge of the network. To do so, they are implementing hybrid multiaccess computing solutions and will eventually compete against or coordinate with the big cloud vendors. Businesses and individuals will leverage telecom service providers' edge cloud resources to develop "bring your own cloud" solutions. A **bring your own cloud (BYOC) solution** involves an organization allowing employees or users to freely decide on the cloud vendor that best suits their tasks rather than standardizing a single specific provider. Services such as AWS and Azure offer their edge-specific tools and frameworks to process data locally and deploy and manage edge devices for real-time analytics and machine learning. In the same way as Starbucks (as an example) plans to provide charging station services for electric cars shortly, it will also go beyond providing simple wireless access to its customers by being able to connect them to the cloud resources needed to operate cloud applications (e.g., extended reality applications that the store anchors to objects in the cloud.)

Looking at the near future, it is likely that the next generations of cloud-based solutions will evolve toward multiaccess networking architecture. Note that this will not change the need for hybrid or multicloud workload deployment or the use of cloud infrastructure or platform services. What will change is which organizations will make these services available closer to the edge of the network to maximize performance and enhance users' experience. In fact, machine learning and large language models such as GPT and Bard may still need to be trained in remote big clouds depending on their resource requirements, but the resulting models will be moved and operated at the edge.

This discussion highlights the need to understand how to leverage cloud infrastructure and platform services to support the development and operation of modern applications. In general, cloud services are available today via the big cloud portals (e.g., portal.azure.com) and via software development kits (SDKs) or APIs readily accessible from pretty much any programming language used to develop cloud applications. In the next two sections, we will delve into the details of how to use these services practically to implement cloud applications.

13.2 Big Cloud IaaS Mainstream Capabilities

Learning Objectives

By the end of this section, you will be able to:

- Learn how to use IaaS storage services
- Learn how to use IaaS compute services
- Understand IaaS support services for web and mobile applications
- Relate to IaaS container management services
- Understand IaaS support services for database management

From a business model perspective, start-ups and companies with fluctuating workloads depend on finding a way to effectively deploy and manage their applications. This includes considerations such as cheap, reliable performances that can adapt to unpredictable seasonal demands. This calls for a service model that allows businesses to access cloud resources and adopt agile infrastructures without having to worry about IT management.

One solution to such demand is infrastructure as a service (IaaS). IaaS offers on-demand access to cloud computing services, provides a pay-as-you-go pricing model, and allows the user to take advantage of cloud servers by virtually managing data and servers for their application. IaaS allows the engineering team to optimize the platform from the infrastructure and does not get locked into any cloud provider's settings.

In this section, we will delve deeper into the different layers of operations that IaaS provides. As the infrastructure is provided without any special setup, IaaS, as a framework, requires additional skills and time from the engineering team to set up and maintain their platform. Throughout these sections, we will go

through different common infrastructures that mainstream IaaS providers usually accommodate so that we can determine the best approach to maintain our cloud applications.

Storage Service

A **storage service** is one of the base infrastructure components that a cloud provider would provide that allow the user and the application to read, write, and access storage. These services are elastic storage services, and the word *elastic* is used throughout the chapter to indicate an unlimited number of resources that the user can access from the cloud provider. Some common components in the application that may require storage access are analytical data, logging information, application data, images, and videos. The storage required for these components may grow over time as the application operates, so the users must understand the type of storage service that they choose for their application. It is also common that the user may choose more than one service to use in their application. There are several types of storage services that the user can choose from, depending on the cloud provider. For example, Microsoft Azure allows users to create a storage account that makes it possible for them to use blob storage as explained in the following paragraph, mount Azure-based remote file systems to their local machine, create column-oriented database tables, or create a queue to receive streaming data from a sensor. However, there are three common types of services that all cloud providers provide:

- The first, **file storage**, manages data as files. This is the most common storage service among first-time users as it is the easiest concept to understand. Most users who use computers daily are familiar with File and the way it stores in their local environment. However, as File has a tendency to grow in size and its format can get complicated, the File storage becomes more difficult to manage as the application operates and grows in the Cloud Environment. The structure of the file system can also be carried into each File being stored in this service. Some common file metadata that are managed and tracked by the Cloud Provider are file name, file size, timestamp, and permissions.
- The second, **object storage** (or blob storage), manages data as blobs, with each blob representing any data format, such as a file in a local filesystem. A blob can contain any type of data, such as a small value, a document, an image, a video, or a collection of such. Cloud providers allow access to the information stored and its associated metadata via an API/SDK or via direct web links in the case of pictures for example. Common metadata includes name, size, timestamp, and custom-tag. In some cases, this approach helps manage the storage of items and also makes them available to anyone who is given access to them in the cloud, which is not possible when storing data items in a local file system. This service is commonly used by consumer applications and allows them to access and retrieve data items as objects using their names and tags.
- The third, **block storage**, manages the data as blocks or physical range of storage in the physical device (such as a hard disk drive [HDD] or Non-Volatile Memory Express [NVMe]). Each block address can range from a few kilobytes or several megabytes in size. Because this service can provide a way for the user to directly access the data using a physical address, it does not need to manage the entire set of data objects. The most common usage for this service would be a system application such as an operating system or database where they need access to files much quicker, and they can keep track of and manage how the data are changed.

To access the storage services from the application, the developer would have more than one way to access the data. Figure 13.2 shows a simple view of different access patterns on how the user and application can get access to the storage service.

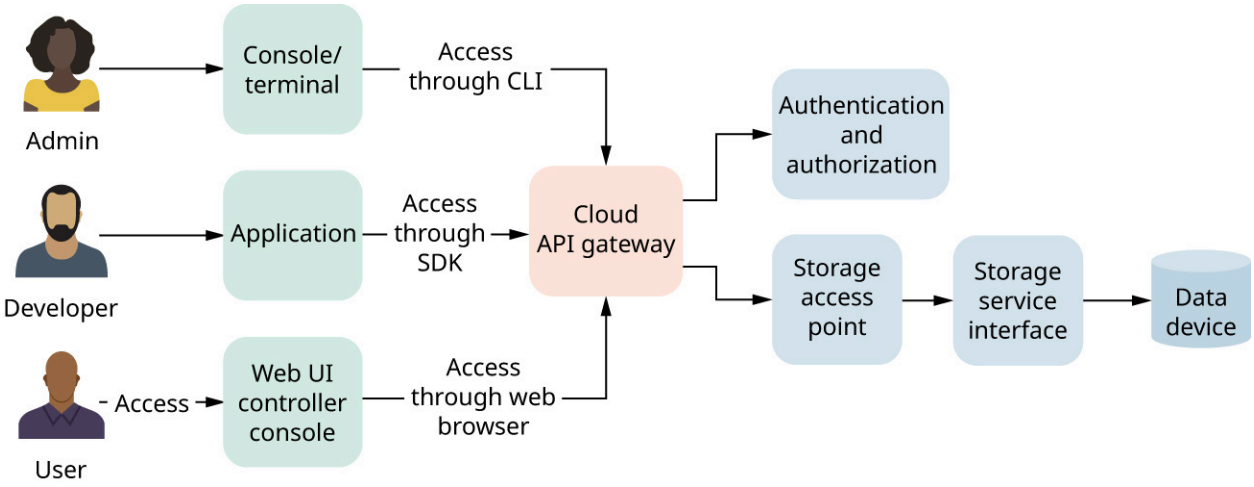


Figure 13.2 As the diagram shows, users and applications can access storage service through different patterns, including CLI or command line interface. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Most cloud providers will provide different tools to support different environments that the user may use to access their infrastructure and services. The storage service will usually be provided as a location in the cloud for the user to access. If the user is a developer, they can access the storage directly through their code using a **software development kit (SDK)**, which is used to access the storage needed to read, write, and store data. This kit is usually provided by the cloud provider and works with different programming languages. Some applications running on the cloud environment also use this SDK to write operational data, such as logs, or consume different configurations that were stored in the cloud environment. If a user is a cloud administrator, they can also access the storage service through a computer terminal using command line interface (CLI). This is also a tool provided by the cloud provider, and this tool will allow users to access the storage service and manage their data. Finally, the user can also access the storage service through the web interface console through their browser. By doing it this way, the user can access their data interactively through their browser. Generally, cloud resources are available either via a cloud portal interface, an SDK that is accessed programmatically, or a CLI that enables users to invoke SDK constructs on the command line. Figure 13.3 shows the web interface for an Azure storage account container (aka, blob) accessible via the Azure object storage service's web console. On the Amazon AWS cloud, blobs can be created as buckets within the Simple Storage Solution (S3).

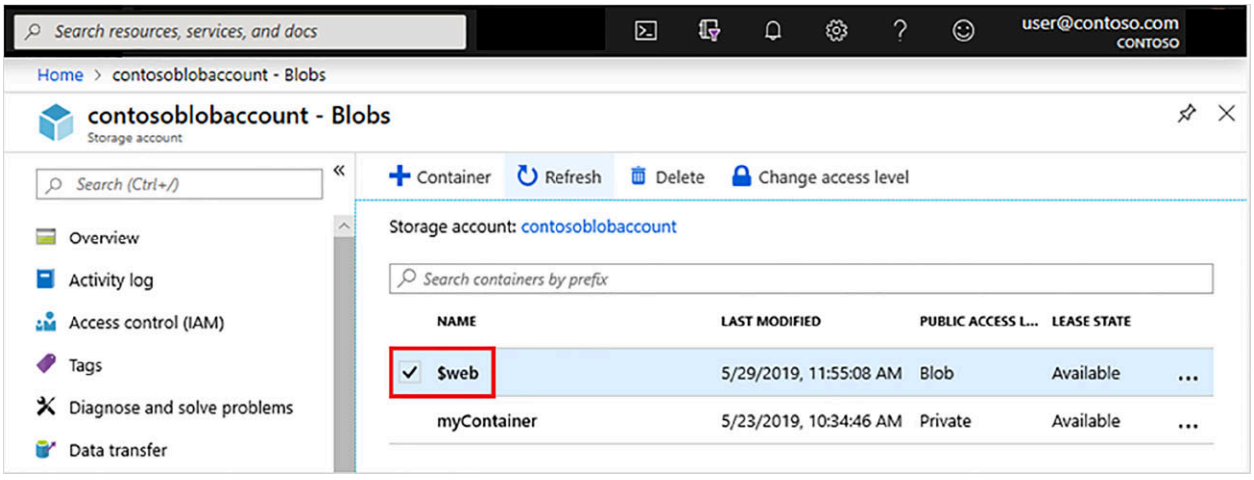


Figure 13.3 This image from Azure object storage service's web console provides an additional example of the web interface used to store data containers (i.e., blobs) in Azure within a storage account. (Used with permission from Microsoft)

Another important component in the cloud storage service is the **storage access point**. The access point is one of the critical components in cloud architecture that will minimize the latency that the user can consume the data. Based on the user's location, the cloud provider can provide the available access point closest to the

user's location to eliminate long I/O latency. However, this is also one of the key problems of leveraging storage service when the application is scaled out. Depending on the cloud provider, there are a fixed number of access points in an area for each account to consume. Usually, it would not be an issue for a small- or medium-performance application. However, for real-time and high-performance applications, this restriction may become a bottleneck for the application to operate correctly. This issue can be resolved when the user moves to a hybrid cloud solution where they can spend a bit more up front to pay for backing up data devices and hardware that can be located in their local area.

Compute Services

The **compute service** provides the ability for the user to obtain access to a private computing environment and is another base infrastructure component that a cloud provider would need to provide. This is also one of the first things that the user will try when they start using cloud services. This will provide a similar experience to a remote computing service that a user may experience in their local environment. The user can access this environment to develop or run any tasks that they cannot run in their local environment. The performance of this service can vary based on the user request, or it can be based on how the application and tasks are required during runtime. Depending on the computer hardware specification, when they need to run, and how they scale based on the application requests, there are various services the user can select. The following are three common compute services the user can see from a cloud provider:

- A **virtual compute service (VCS)**, which enables the user to request an environment to do some tasks and then shut it down to release the resource back to the cloud provider. Depending on when it was requested and how long the user keeps it running, the price for this service varies.
- A **spot/not urgent compute**, which enables the user to get a task done but keeps costs low by allowing the cloud provider to run the task without urgency when the time is convenient and cost-effective.
- A **virtual functional and serverless compute service**, which is an application that runs in the compute environment, is executed as a function, and is then shut down when the task is completed. The user will get charged based on the number of tasks or requests that the service completes. Because this service requires a different backend service from the cloud provider to operate, the cost per time unit for this service will be higher than other services. This service is common for cloud microservice architectural applications where the user may host different components and functions of their application as separate components. This will allow each component to be scaled independently and avoid a bottleneck to one large instance of an application needing to be maintained.

Similar to storage service, the user can have more than one way to access compute service. However, because the user needs to interact with the compute service to complete a task, they usually access through two main channels: CLI and Web UI controller console. Figure 13.4 shows how different users' roles can access the compute service.

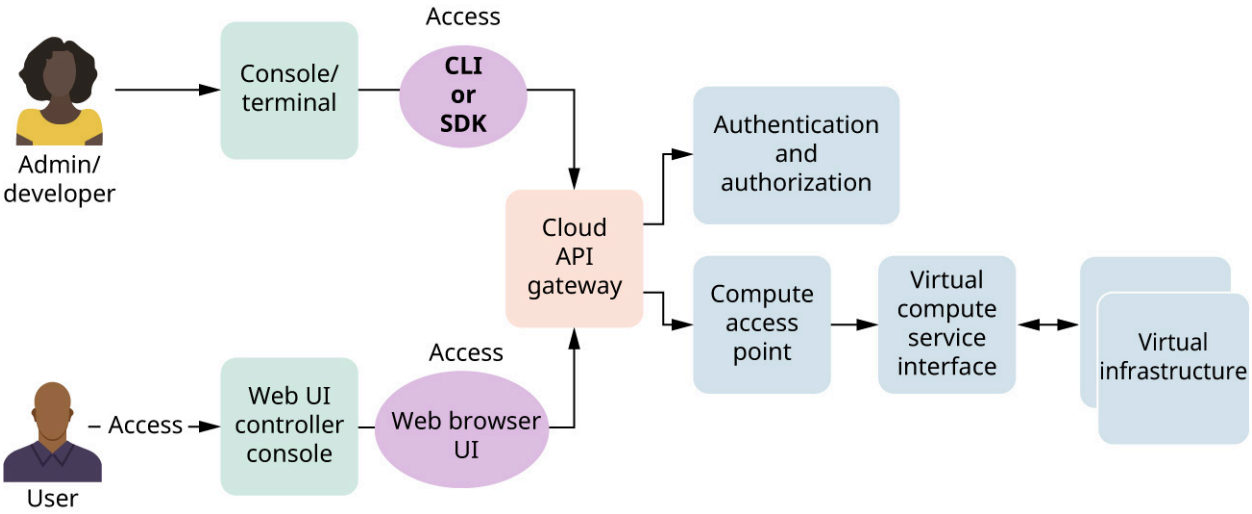


Figure 13.4 The access pattern for a compute service may differ depending on a user's role. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

One thing to note is that we ignore the access pattern where either the user uses a local integrated development environment (IDE) to access the computing environment or when the user accesses the IDE that is hosted in the virtual environment. In both cases, the user has to configure the computing environment differently to bypass the provided tools and interfaces.

Another challenge in compute service that is similar to storage service is the limited number of access points in an area or cloud account that the user can use. However, this is not as big of a challenge as storage computing because most of the computing tasks can be scheduled, and it is easier to provide minimum compute service and operate with minimal performance that provides storage access with minimal I/O latency. The high I/O latency can destroy the user experience, and it can occur unpredictably.

LINK TO LEARNING

Compute service is a critical component of cloud solutions. One of the largest providers of compute services is Amazon Web Services (AWS), which supports millions of customers, including small businesses as well as larger organizations. AWS's website provides an overview of compute (https://openstax.org/r/76Compute) and explains how these services benefit cloud computing.

Web and Mobile App Services

Among all applications and workloads in the cloud, two common workloads have emerged in recent years. Those are web and mobile workloads where the user wants to provide real-time access to all users across the globe. This is one of the most critical reasons why the user wants to move into the cloud environment because the cost to scale the global infrastructure is extremely high.

A content delivery network (CDN) is an important cloud capability that accelerates web and mobile workloads access globally. A CDN is a network of servers and associated networking infrastructure that is spread across the globe and allows access to web and mobile workloads from anywhere. It is configured to prioritize and cache common data and content (e.g., videos) in different geographical areas so it can increase access and processing speeds of mobile and web applications in the global network, which results in improved user experience and reduced energy costs. On-demand streaming services (e.g., Netflix, Hulu, Tubi) use CDNs to direct users to the closest server (i.e., a network edge server located at the edge of the network closer to their location) from which they can stream their movies. The web clients/apps provided by these vendors allow dynamic adaptive streaming over the Web (via the HTTP web protocol) to ensure high-quality streaming of

media content over the Internet delivered from CDN servers. Figure 13.5 shows a high-level architecture of how CDN is set up on Azure Cloud Service.

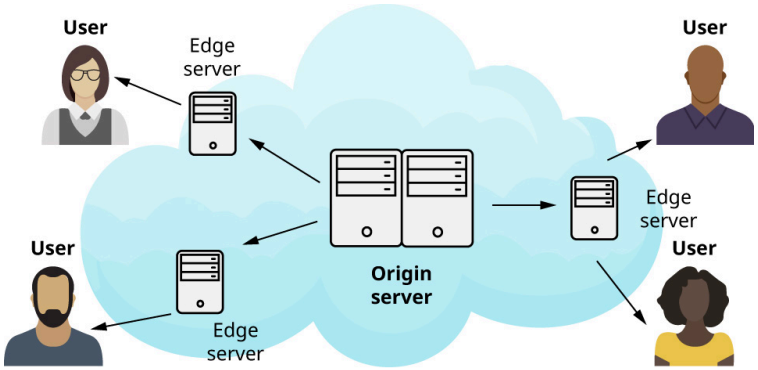


Figure 13.5 Microsoft Learn's content delivery network on Azure Cloud Service is set up using a high-level architecture. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Outside of the infrastructure, the cloud provider is also providing other common patterns and services that most web and mobile applications need to use to provide an optimal experience to the end consumer. These patterns and services are pre-implemented and optimized by the cloud provider and have an easy integration with the compute and storage services where the application runs. The following are two common web and mobile cloud components:

- The component that stores and manages sensitive information securely while allowing the application to be scaled and deployed in different environments on the cloud is **secret and configuration management**. Different environments will require different configurations and secret details, such as database access or language settings. The best practice to handle complex environmental configurations and secrets is using a global cloud service to inject those details during runtime so the application code can be free of custom implementation and the application deployment and version control would be much simpler. Most cloud providers will allow the user to manage their custom configuration and secrets and update them during application runtime.
- The component that allows developers to centralize all logging data and provide a comprehensive view of all events happening with an application at any moment is **logging and monitoring management**. As the application grows and scales in different environments, the ability for the developer to know and understand how their application runs becomes more and more important. The administrator's ability of the administrator to identify and mitigate the issue during the runtime is an essential requirement for any cloud-based application. For this reason, most cloud providers provide logging and monitoring services together with the compute service so that they can centralize all logging data and provide a comprehensive view of all events that are happening with the application at any moment. The user can leverage this service to obtain some monitoring capabilities out of the box when they deploy their application to the cloud environment. However, based on different applications and requirements, the user may expand these capabilities by adding custom logging logic or notification configurations to match their needs. Figure 13.6 shows a simple monitoring dashboard on Azure Cloud Service.

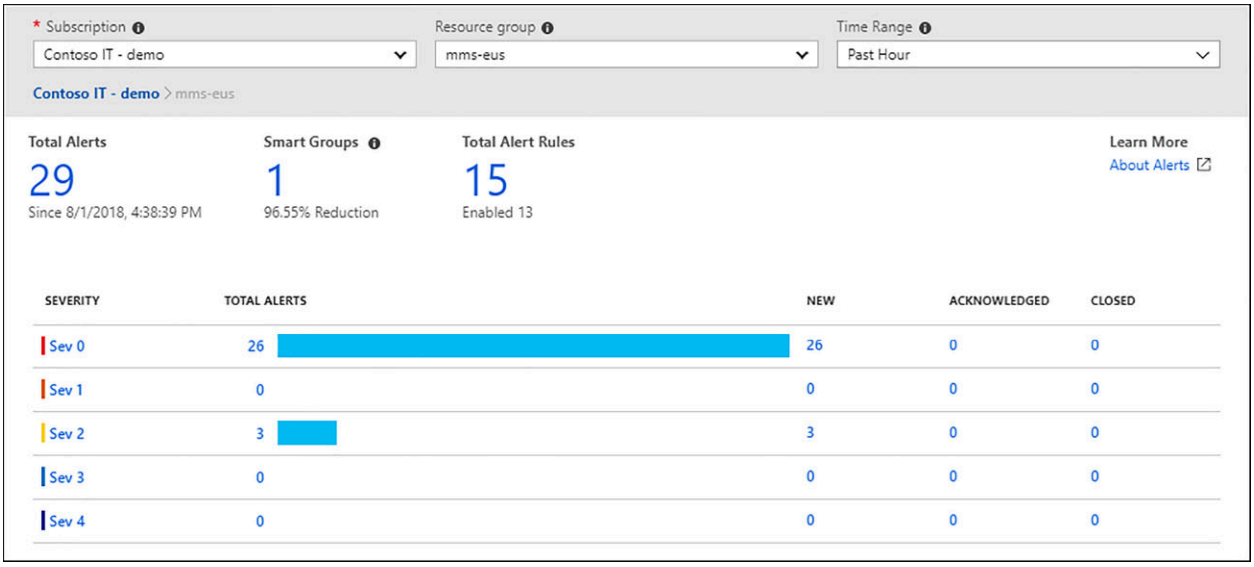


Figure 13.6 This picture provides an example of a simple monitoring dashboard on Azure Cloud Service. (Used with permission from Microsoft)

Container Management Services

In recent years, **container management services**, which encapsulate an application with the necessary operating system libraries for it to operate, have become one of the key innovations that have transformed the use of cloud infrastructures. In a nutshell, the container is a way to encapsulate an application with any operating system's library that is required for the application to operate. By using a container, the developer is not worried about the environmental mismatch between the environment where the application was developed and the environment where the application is run. This container can be run on top of any Linux kernel that provides support to the container runtime. Figure 13.7 shows how a container runs in a regular compute environment. It is also important to point out that the container technology is different from the virtualization technology used in virtual machines.

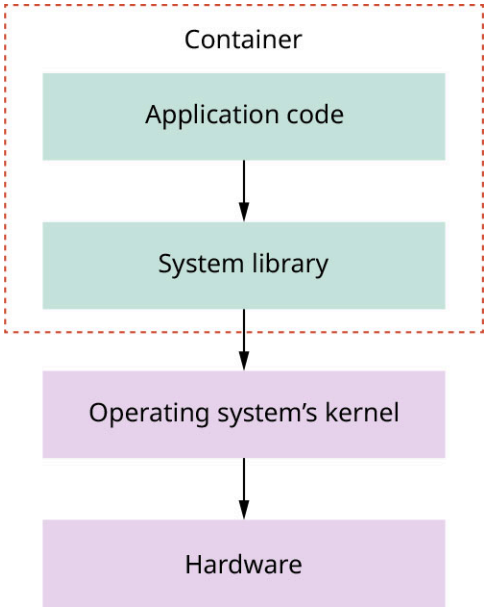


Figure 13.7 This diagram shows the high-level view of how a container runs in a regular compute environment. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

On the cloud environment, the container management service allows the user to deploy, operate, and scale any containerized application. It also includes several necessary components that help the user manage their

containerized application better, such as the following:

- The **container registry (CR)**, or the registry where the user can version control each container image that they deployed into the cloud environment. The user can usually manage the container using image metadata such as namespace, image name, or image tags.
- The **base container image**, which is the foundational layer of a container provided by the cloud provider to build an application. The image is mostly up-to-date with secure patches and packages that allow the user to update their application with the latest security update.
- The Kubernetes environment, or Kubernetes (K8S) service, which is the most popular container orchestration system. This is a managed service that is usually provided by the cloud provider to allow the user to scale their containerized application in a Kubernetes environment. The K8S environment is one of the key systems to run a hybrid cloud environment where the user can run applications from both their local and cloud environment.

Figure 13.8 shows a simple workflow to deploy a containerized application into Azure Cloud Environment. Architecting a hybrid cloud solution requires strong technical expertise in application development and containerized application, and it also requires deep knowledge of cloud solutions. The details on how to do it correctly will not be covered in this section. However, the hybrid solution should be the target for any user on their journey of migrating their application into the cloud.

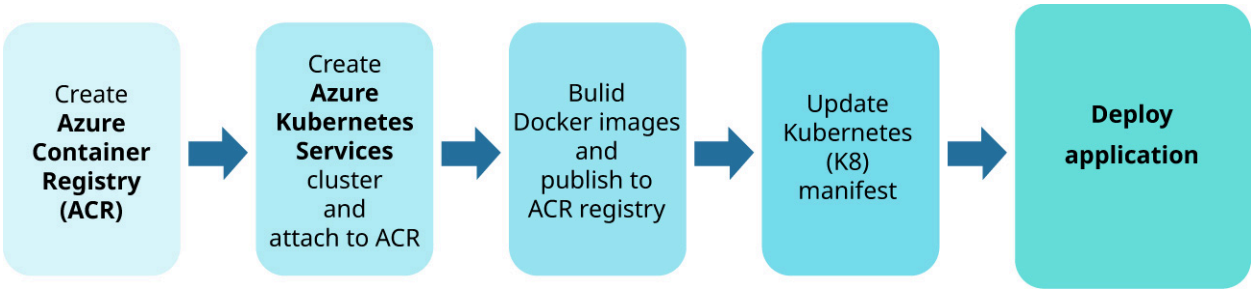


Figure 13.8 On Azure, K8S can be part of a simple containerized application deployment workflow. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

CONCEPTS IN PRACTICE

Containers and Virtual Machines

For deployment, organizations can use containers or virtual machines. How do you decide which technology is more appropriate? Containers have more scalability and are practical if you're working with multiple environments and you want to package and run your applications in a manner that is predictable with repetition from one environment to the next. Virtual machines (VM) provide more environmental control and are practical if you need to use the same physical machine to install more than one operating system, as well as create more than one environment. Another consideration is the speed of software development. It is faster and easier to build and test new features on containers compared to VMs. However, VMs provide better security. While there's a lot to consider when deciding on containers versus VMs, both technologies offer important benefits, providing organizations with critical resources for deployment.

Database Management Services

Today, database or any data management system is the heart of any application. However, managing a database is usually a difficult task as the data can be growing oversized both in size and in complexity. For this reason, most cloud providers provide several managed database services for any application that runs inside and outside of the cloud environment. There are two popular database management services:

- A **relational database service (RDS)**, in which the relationship between data is strictly managed. The data can be managed under a predefined data schema. This is the most common type of database that most developers are familiar with. Some popular examples of this type of database are Oracle DB, MySQL, or PostgreSQL.
- A **NoSQL database**, in which the relationship between data is not strictly managed. The data can be managed under key/value pair. This type of database has become popular in recent years with monitoring applications where the data needs to be written and captured quickly. Some popular examples of this type of database are MongoDB Atlas and Cassandra DB.

The access pattern for these services is similar to how any application access to any managed database. Different secure authentication and authorization methods can be configured through the CLI or web console by the user and the application can access the database from inside or outside of the cloud environment.

13.3 Big Cloud PaaS Mainstream Capabilities

Learning Objectives

By the end of this section, you will be able to:

- Learn how to use Internet of Things cloud PaaS services
- Learn how to use shallow and deep machine learning cloud PaaS services
- Learn how to use blockchain cloud PaaS services
- Understand PaaS services support for extended reality applications
- Understand PaaS services support for 3-D/4-D printing services
- Relate to PaaS services for cloud application development

For businesses that look for all existing features in IaaS, as well as a service that provides a complete platform to develop, test, and launch their applications, platform as a service (PaaS) has become one of the top options. On top of servers, network, and security that IaaS provides, PaaS also includes middleware such as operating systems and development tools that enable quick application development and market launch. As the cloud provider manages more layers, PaaS will require less time and skill from the engineering team to manage their infrastructure. It will provide freedom for the company to focus on developing applications and providing services. However, because of those abstracted services, PaaS may have higher costs with dependency on the particular platform or vendor. The higher cost may come later when the organization explores different technical decisions and cloud vendors.

In this section, we will explore various PaaS services that enable organizations to effectively launch and maintain large-scale applications. This includes AI incorporations or XR platforms that allow for operations of scripting and modeling. Through real-life examples, we can analyze how companies slash costs while also accelerating time to market and application development.

Internet of Things Services

A 5G network enables mobile computing at the edge of modern telecommunication networks with support for a variety of IoT devices, including laptops, smartphones, and smartwatches. 5G has higher radio frequencies, which transfer considerably more data over the air at faster speeds while reducing congestion and lowering latency. Thanks to 5G, more IoT devices can be used simultaneously within the same geographic area. As a result, today's dynamic information networks consist of interconnected sensors, actuators, mobile phones, robotics, and smart devices.

IoT network traffic falls broadly into two categories: telemetry and telecommand. One category, **telemetry**, aggregates data generated by sensors and devices and sends them to a server. The other category, **telecommand**, sends commands across a network to control devices or sensors. Figure 13.9 illustrates the typical flow of IoT data generated by mobile edge devices and data processing and storage via cloud PaaS services.

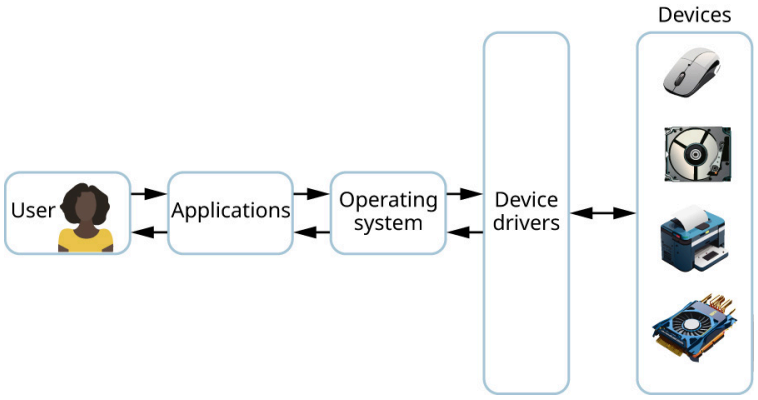


Figure 13.9 Generally, this is how IoT data flow via mobile edge devices, data processing, and storage via cloud PaaS services. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

To serve the purpose of IoT, several application-layer protocols have been developed, such as Message Queuing Telemetry Transport (MQTT), Advanced Message Queuing Protocol (AMQP), Constrained Application Protocol (CoAP), Extensible Messaging and Presence Protocol (XMPP), and Simple Text Orientated Messaging Protocol (STOMP). This was necessary because application layer protocols, such as HTTP, are not suitable for IoT telemetry and telecommand applications. HTTP is designed for one-to-one communication rather than one-to-many communication between many sensors and one server and is a reliable protocol for web applications. However, HTTP supports unidirectional synchronous request-response communication and cannot send data in both directions simultaneously. IoT sensors cannot work efficiently in a synchronous manner. HTTP also is not designed for event-based communication. Generally, HTTP's scalability is achieved by loading the server, which puts a heavy load on sensor devices connected to multiple other devices. In addition, HTTP uses high power consumption, making it unsuitable for advanced wireless sensor networks.

To understand how IoT application-layer protocols work, consider MQTT, which is an example of a lightweight application-layer messaging protocol. It is based on the publish/subscribe (pub/sub) model typically used for message queuing in telemetry applications. Multiple clients, or sensors, can connect to a central server, known as a broker, and subscribe to topics that interest them. Through the broker, which is a common interface (router) for sensor devices to connect to and exchange data, these clients also have the option of publishing messages regarding their topics of interest. To make the communication reliable, MQTT uses a TCP connection on the transport layer for connections between sensors and the broker.

Big cloud vendors use IoT protocols such as MQTT to provide IoT PaaS services and related frameworks that facilitate the interactions of IoT devices with the cloud. The use of these services makes it possible to build and deploy innovative IoT solutions without developing and managing IoT frameworks on local computers. In particular, Microsoft Azure IoT manages cloud services that can interconnect, monitor, and control billions of IoT assets. An IoT solution is typically made up of one or more IoT devices that communicate with one or more back-end services hosted in the cloud. IoT devices can be constructed as circuit boards with sensors attached that use Wi-Fi to connect to the Internet (e.g., presence sensors in a room). Devices may be prototyped using a Raspberry Pi or the Microsoft MXChip IoT DevKit, which has built-in sensors for temperature, pressure, and humidity, as well as a gyroscope, accelerometer, and magnetometer. Microsoft also provides an open-source IoT device SDK to help build apps on devices. In addition, Microsoft's IoT Edge and IoT Hub frameworks can be used to facilitate the operation of IoT applications at the edge and the collection and transfer of data to the cloud via common communication protocols such as MQTT and AMQP.

The technologies, PaaS services and solutions provided by Azure IoT are summarized in Table 13.1. AWS, GCP, and IBM Cloud also provide equivalent IoT PaaS services and related capabilities.

IoT Central Application Templates	IoT Solutions	Azure Services for IoT	IoT and Edge Device Support
Retail Health Energy Government	Azure IoT central-managed application platform	Azure IoT Hub Azure IoT Hub Device Provisioning Service Azure Digital Twins Azure Time Series Insights Azure Maps	Azure Sphere Azure IoT Device SDK Azure IoT Edge Azure Data Box Edge
	Reference architecture and accelerators (PaaS)	Azure Stream Analytics Azure Cosmos DB Azure AI Azure Cognitive Services Azure ML Azure Logic Apps	Windows IoT Azure Certified for IoT—Device Catalog Azure Stream Analytics Azure Storage
	Dynamics connected field service (SaaS)	Azure Active Directory Azure Monitor Azure DevOps Power BI Azure Data Share Azure Spatial Anchors	Azure ML Azure SQL Azure Functions Azure Cognitive Services

Table 13.1 IoT Technologies, Services, and Solutions Available through Microsoft Azure

Shallow and Deep Machine Learning Services

IoT services provided by big cloud vendors include both **shallow machine learning**, which has few neuron layers, and **deep machine learning**, which has many neuron layers. PaaS services enable application developers to leverage machine learning capabilities on the cloud. Developers can build and deploy innovative machine learning solutions without using local computers to set up and manage machine learning frameworks, such as Apache Hadoop or Spark, along with related libraries and/or tools. This includes building and deploying solutions that require using and streaming data analytics.

In addition to Microsoft, AWS, GCP, and IBM Cloud also provide shallow and deep PaaS services and related capabilities.

Big Data Analytics Services

The process of analyzing big data to find correlations, consumer preferences, market trends, and related information is referred to as big data analytics. It is important to help organizations with decision-making processes. Big data analytics processes usually require training models using data sets with a manageable number of descriptive features. As such, big data analytics requires shallow rather than deep machine learning services. Tools for big data analytics include big data analytics frameworks, machine learning libraries, and analytics machine learning tools.

Big Data Analytics Frameworks

Various cloud vendors, including Amazon, Cloudera, Dell, Oracle, IBM, and Microsoft, offer an implementation

of the Apache Hadoop or Spark stacks that are useful to support big data analytics projects. Other cloud data analytics frameworks include Amazon Elastic MapReduce (EMR), Amazon Athena, Azure HDInsight, Azure Data Lake, and Google Cloud Datalab. EMR is a useful framework to host Spark. HDInsight is similar to EMR in power, and it supports Spark, Hive, HBase, Storm, Kafka, and Hadoop MapReduce. HDInsight guarantees 99.9% availability and integrates various programming tools, such as Visual Studio, and supports various programming languages like Python, R, Java, and Scala, as well as .NET languages. As illustrated in the Azure Data Lake conceptual view (Figure 13.10), HDInsight includes all the usual Hadoop and Yarn components, such as Hadoop File System (HDFS) as well as tools that integrate other Microsoft business analytics tools such as Excel and SQL Server.

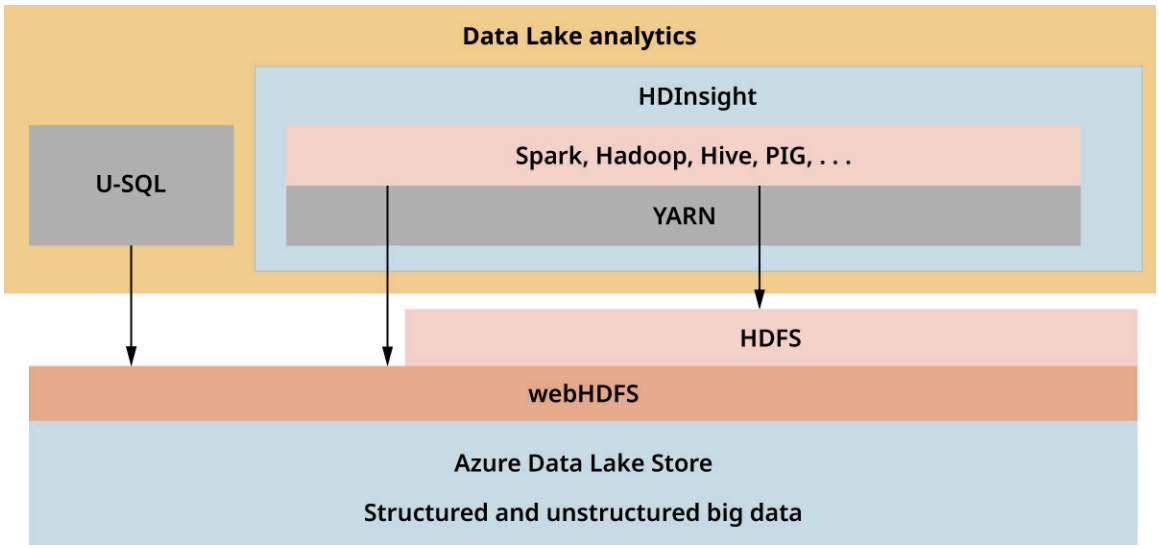


Figure 13.10 This graphic shows the big data analytics tools available in HDInsight with Azure Data Lake. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Big Data Machine Learning Libraries

Machine learning libraries have algorithms and functions in place to develop machine learning models needed for big data analytics. For example, Spark includes the MLlib scalable machine learning library and the GraphX API for graphs and graph-parallel computations. MLlib offers classification, regression, clustering, and recommender system algorithms. MLlib was originally built directly on top of the Spark RDD abstraction. It provides an API to specify dataframes, transformers, estimators, and a high-level API for creating ML pipelines. GraphX is a Spark component that implements programming abstractions based on the RDD abstraction. GraphX comes with a set of fundamental operators and algorithms to work with graphs and simplify graph analytics tasks. MLlib and GraphX are available to application developers via Azure ML and ML programming offerings from other cloud vendors.

Big Data Analytics Machine Learning Tools

Azure Machine Learning (ML) is a cloud portal for designing and training machine learning cloud services. Azure also provides Databricks, which is an Apache Spark–based analytics platform optimized for the Microsoft Azure cloud environment, as well as the ML.NET open-source and cross-platform machine learning framework. Amazon's AWS machine learning service (i.e., Amazon SageMaker), like Azure ML, can be used to create a predictive model based on the training data you provide. It requires much less understanding of ML concepts than Azure ML. Its dashboard presents a previous list of experiments, models, and data sources that enables developers to define data sources and ML models, create evaluations, and run batch predictions. As illustrated in Figure 13.11, it is based on a drag-and-drop component composition model. You build a solution to a machine learning problem by dragging solution parts from a palette of tools and connecting them into a workflow graph. You then train the solution with your data. When you are satisfied with the results, you ask Azure to convert your graph into a running web service using the model you trained. The tool supports

customized machine learning as an on-demand service.

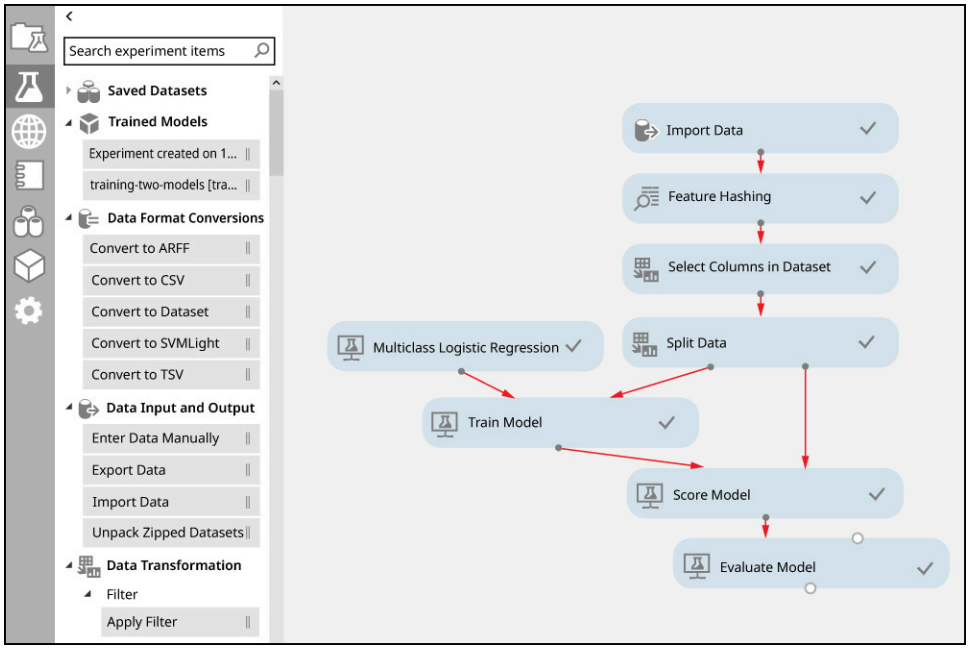


Figure 13.11 Azure ML is based on a drag-and-drop component composition model that enables you to build a solution to a machine learning problem. (Used with permission from Microsoft)

This is another example of serverless computation. It does not require you to deploy and manage your VMs; the infrastructure is deployed as you need it and if your web service needs to scale up because of demand, Azure scales the underlying resources automatically.

Amazon also provides a portal-based tool, Amazon Machine Learning, that allows you to build and train a predictive model and deploy it as a service. In addition, both Azure and Amazon provide pre-trained models for image and text analysis in the Azure Cognitive Services and the Amazon ML platform, respectively.

THINK IT THROUGH

Using Avatars in Virtual-Reality Environments

Avatars are a popular way to represent ourselves graphically online.

Can avatars in the metaverse's virtual-reality environment make use of big cloud PaaS capabilities to operate semi-autonomously? If they're semi-autonomous, they'll be able to act independently, with limited control from users. What ethical consequences could this pose?

Streaming Big Data Analytics Services

Vendors provide various services to facilitate streaming big data analytics in the cloud, such as Spark Structured Streaming, Amazon Kinesis Data Firehose and Kinesis, Azure Stream Analytics, and Google Dataflow (based on Apache Beam). Spark Structured Streaming also provides another high-level concept called the DStream, which represents a continuous stream of data as a sequence of RDD fragments with windowed computations. Spark Structured Streaming leverages Spark Core and its fast scheduling engine to perform streaming analytics.

Amazon Firehose, which is designed for extreme scale, can load data directly into Amazon S3 or other Amazon services. Kinesis Data Analytics for SQL applications provides SQL-based tools for real-time analysis of streaming data from Kinesis Data Streams or Firehose. Kinesis Data Streams provides ordered, replayable,

real-time streaming data. Various open-source frameworks, such as Kafka, Storm, RisingWave, Apache Spark, Apache Flume, Apache Beam, and Apache Flink, are also available to process streaming data on local machines. Table 13.2 compares Firehose to Kinesis Data Streams. The two primary components of the Azure Stream Analytics are the Azure Event Hubs service and Stream Analytics engine.

Feature	Firehose	Kinesis Data Streams
Purpose	Service for transferring data into third-party tools	Streaming service
Provisioning	Fully managed and has no administration	Managed but requires shards configuration
Scaling	Automated scaling based on demand	Manual scaling
Data storage	Data storage not included	Offers data storage that can be configured from 1 to 365 days
Replay capability	No, replay capability is not supported	Yes, replay capability is supported
Message propagation	Almost real-time, depending on buffer size or time	Real-time

Table 13.2 Firehose versus Kinesis Data Streams

Apache Beam is the open-source release of the Google Cloud Dataflow system. Beam treats the batch and streaming cases uniformly and supports pipelines to encapsulate computations, as well as PCollections, which represent data as they move through a pipeline. Beam enables computational transformations that operate on PCollections, produces PCollections, and relies on sources and sinks, from which data are read and to which data are written, respectively.

Deep Learning and Generative AI Services

All the big clouds today provide integrated machine learning services that use various techniques to create models that leverage prior experience and make it possible to improve the ability of machines to perform tasks. These services may be used as part of big data analytics and streaming big data analytics techniques, as explained in the previous subsections. Deep learning (DL) is another technique that leverages artificial neural networks (e.g., RNNs, CNNs, GANs) to create models that perform predictive tasks requiring special training and the ability to relate to a vast combination of labels/patterns (e.g., image recognition, speech recognition, language translation). There are various types of deep learning improvements that include reinforcement learning and transfer learning among others. Artificial intelligence (AI) is also a technique that leverages ML to enable computers to mimic human intelligence. Generative AI (GenAI) is a subset of AI that uses techniques such as deep learning and transformers to generate new content (e.g., create images, text, or audio) that matches a request. Transformers are special types of ML model architectures that are suited for solving problems that contain sequences such as text or time-series data.

Transformers have been used recently to solve natural language processing problems (e.g., translation, text generation, question answering, text summarization), and various transformer implementations have been quite successful, including the bidirectional encoder representations from transformers (BERT) and the generative pre-trained transformers (GPTs). ML models that support GenAI are referred to as large language

models (LLMs) and/or foundation models (FMs). LLMs are typically tuned toward specific conversational applications and require more parameters and data-intensive training. Foundation models (FMs) are more general-purpose than LLMs and less data-intensive. Examples of LLMs include OpenAI's ChatGPT, Google Gemini, Meta M2M-100 and LlaMA, IBM's Granite model, Anthropic's Claude models, Mistral AI's models, and many others. LLMs (and FMs) can be augmented using retrieval augmented generation (RAG) AI frameworks that supplement the FMs internal representation of information to improve the quality of the LLM-generated responses.

Cloud vendors provide DL services in the form of programming frameworks that help implement deep learning applications using differentiable programming. GenAI services provided by the big clouds and other vendors are prompt interfaces that are specially engineered to allow users to get the most out of an LLM by including a sufficient amount of information in the prompts they create.

The following provides a nonexhaustive list of DL services provided by some of the big clouds:

- Amazon AWS deep learning services:
  - Amazon Deep Learning AMIs (DLAMIs). Amazon DLAMIs are customized machine images that may be used for deep learning in the cloud. They can be deployed on various types of Amazon VMs (i.e., EC2 instances), including CPU-only instances or the latest high-powered multi-GPU instances. DLAMIs come preconfigured with NVIDIA CUDA and NVIDIA cuDNN and the latest releases of the most popular deep learning frameworks. Amazon released the Amazon Deep Learning AMI with Conda, which uses Conda virtual environments to isolate each framework, allowing you to switch between them at will without their dependencies conflicting. The full list of supported frameworks by Amazon Deep Learning AMI with Conda includes PyTorch, TensorFlow 2, and Apache MXNet (now retired but can still be accessed and used). Starting with the v18 release, Amazon Deep Learning AMI with Conda no longer includes the CNTK, Caffe, Caffe2, Theano, Chainer, or Keras Conda environments. Configuring the Amazon DLAMIs to use Jupyter is easy; go to the Amazon Marketplace on the EC2 portal and search for "deep learning." You will find the DLAMIs, then select the server type you would like to use. If you simply want to experiment, it works well with a no-GPU option; when the VM comes up, log in with ssh, and configure Jupyter for remote access.
  - Amazon Lex. Amazon Lex is an IA chatbot that allows users to incorporate voice input and conversational interfaces into applications. It is an extension of Amazon's Echo product, a networked device with a speaker and microphone that you can ask questions through the Alexa service (e.g., questions about the weather, event scheduling, news, and music). It is possible to associate Echo voice commands with the launching of an Amazon Lambda function that executes a cloud application.
  - Amazon Polly. Amazon Polly is the opposite of Lex; it turns text into speech for dozens of languages with a variety of voices and uses the Speech Synthesis Markup Language (SSML) to control pronunciation and intonation.
  - Amazon Rekognition. Amazon Rekognition is at the cutting edge of deep learning applications. It takes an image as input and returns a textual description of the items that it sees in that image. This includes objects, landmarks, dominant colors, activities, and faces. It also performs detailed facial analysis and comparisons and can identify inappropriate content that appears in images.
- Microsoft Azure deep learning services:
  - Azure Data Science VMs (DSVMs). Azure DSVMs are Azure Virtual Machine images, preinstalled, configured, and tested with several popular tools that are commonly used for data analytics, machine learning, and AI development and training.
  - Azure Machine Learning (ML). Azure ML is a cloud service that is designed to help accelerate and manage machine learning project life cycles. It can be used to train and deploy ML models and manage machine learning operations (via MLOps). You can create a model in Microsoft ML or use a model built from an open-source platform, such as PyTorch, TensorFlow, or scikit-learn. MLOps tools help you monitor, retrain, and redeploy models. As noted earlier, Azure also provides the ML.NET machine

learning framework.
    - Azure AI services. Azure AI services are APIs/SDKs that can be used to build applications that support natural methods of communication (i.e., see, hear, speak, understand, and interpret user needs). These services include support for vision (e.g., object detection, face recognition, optical character recognition), speech (e.g., speech-to-text, text-to-speech, speaker recognition), languages (e.g., translation, sentiment analysis, key phrase extraction, language understanding), and decision (e.g., anomaly detection, content moderation, reinforcement learning).

- Google deep learning services:
    - Deep learning VMs. Deep learning VM images are virtual machine images optimized for data science and machine learning tasks. All images include preinstalled ML frameworks and tools and can be used on VM instances with GPUs to accelerate data processing tasks. ML frameworks supported include TensorFlow and PyTorch.
    - Google machine learning APIs. Google provides various APIs to services that can be used to build applications that support natural methods of communication, including Cloud Vision to understand the content of an image, Cloud Speech-to-Text to transcribe audio to text, Cloud Translation to translate an arbitrary string to any supported language, and Cloud Natural Language to extract information from text.

The following provides more information related to GenAI services provided by the big clouds:

- AWS GenAI services. Amazon AWS provides various GenAI tools, including the Amazon Q AI-powered assistance and the Amazon Bedrock suite of LLMs, FMs, and generative AI tools. Amazon SageMaker may be used to build, train, and deploy FM models at scale.
- Microsoft Azure GenAI services. The Azure OpenAI service and the Azure AI studio can be used to create custom copilot and generative AI applications. Microsoft has partnered with OpenAI, the company that is developing ChatGPT. It also provides the Phi family of small language models (SLMs) that are low-cost and low-latency alternatives to LLMs in some cases.
- Google GCP GenAI services. Vertex AI, Generative AI Studio, and Vertex AI Model Garden are various solutions that Google provides to support the creation of generative AI applications. Google also provides the Gemini family of generative AI models that are capable of processing information from multiple modalities, including images, videos, and text.
- IBM Cloud GenAI services. IBM Watsonx.ai AI studio brings together generative AI capabilities that are powered by FMs and ML. It provides tools to tune and guide models based on enterprise data as well as build and refine prompts. IBM also develops custom Granite AI foundation models that are cost-efficient and enterprise-grade.
- Other GenAI services. In addition to the GenAI services and tools mentioned here, many other vendors focus on the creation of LLMs, SLMs, and FMs. Here are a few of them:
    - OpenAI's ChatGPT
    - Meta LlaMA
    - Anthropic Claude
    - Mistral AI

ML Toolkits Performance

ML toolkits can be used for various tasks, such as scaling a computation to solve bigger problems. One approach is the SPMD model of communicating sequential processes by using the message passing interface (MPI) standard model. Another is the graph execution dataflow model, used in Spark, Flink, and the deep learning toolkits. You can write ML algorithms using either MPI or Spark. You should be aware that MPI implementations of standard ML algorithms typically perform better than the versions in Spark and Flink. Often, the differences are factors of orders of magnitude in execution time, but the MPI versions are harder to program than the Spark versions.

Blockchain Services

Blockchains use a distributed ledger system to store data and transactions in an open-source database that enables you to build applications that allow multiple parties to securely and transparently run transactions and share data without using a trusted central authority. With blockchain 2.0, developers have a mechanism that allows programmable transactions, which are modified by a condition or set of conditions. Blockchain 2.0 is not limited to supporting transactions. It can also handle microtransactions, decentralized exchange, and creating and transferring digital assets. Blockchain 2.0 also has the ability to handle smart contracts, which are scripts executed in a blockchain 2.0 environment. The codes of smart contracts are accessible to the public, and anyone can verify the correctness of code execution. The actual verification is carried out by miners in the blockchain environment, and this ensures honest execution of the "contract." Smart contracts rely on cryptography to secure them against tampering and unauthorized revisions.

A **blockchain network** is a peer-to-peer network that allows people and organizations who may not know one another to trust and independently verify the records of financial and other transactions. This improves the efficiency and immutability of transactions for business processes such as international payments, supply chain management, land registration, crowdfunding, governance, financial transactions, and more.

Ethereum.org implements a blockchain 2.0 decentralized computing platform for Web3 and provides a language to write transaction scripts. Web3 applications are referred to as DApps and may be deployed on blockchain 2.0 decentralized computing platforms. Ethereum.org provides its own development environment (i.e., Remix) and programming languages, such as Solidity, to develop and deploy contracts. Public test networks such as Goerli may be used to develop and test contracts before deploying them on the Ethereum platform. Web3 APIs are available for various programming languages, such as JavaScript, Python, Haskell, Java, Scala, and PureScript, to facilitate the creation of applications that interact directly with the blockchain 2.0 platform.

Using PaaS blockchain services on AWS, Oracle, GCP, and IBM big clouds, it is possible to create a blockchain decentralized computing platform to facilitate the deployment of Web3 DApps. Big clouds provide clusters of VMs that may be leveraged as P2P nodes within a blockchain 2.0 decentralized platform implementation, such as Hyperledger. While Microsoft Azure offered PaaS services to create blockchain platforms, it has retired these services and partnered with ConsenSys and other companies to provide that support. Azure does provide products and services, Web3 developer tools, and security capabilities to help create Web3 applications and deploy them on partner platforms.

LINK TO LEARNING

To learn more about blockchain PaaS services available in the big cloud, you can visit multiple websites and explore blockchain features. For example, you can find information about AWS blockchain (https://openstax.org/r/76AWSBlockchain) online.

AWS Blockchain Services

AWS provides blockchain templates that help you create and deploy blockchain networks on AWS using different blockchain frameworks. AWS Managed Blockchain, which is shown in Figure 13.12, is used to configure and launch AWS CloudFormation stacks to create blockchain networks. The AWS resources and services used depend on the AWS blockchain template selected and the options to specify the fundamental components of a blockchain network.

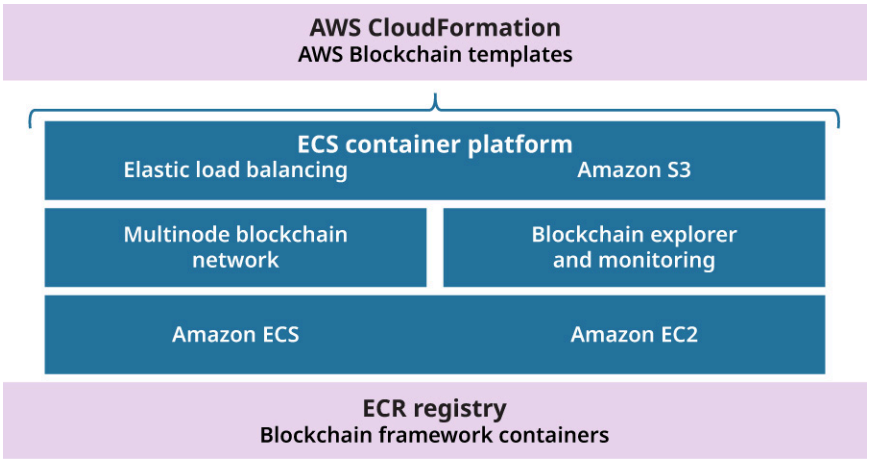


Figure 13.12 This graphic shows AWS blockchain templates, which are used to configure and launch AWS CloudFormation stacks to create blockchain networks. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Amazon Managed Blockchain is a fully managed service for creating and managing blockchain networks that supports the Hyperledger Fabric open-source framework. You can use Managed Blockchain to create a scalable blockchain network quickly and efficiently using the AWS Management Console, the AWS CLI, or the Managed Blockchain SDK. Managed Blockchain scales to meet the demands of thousands of applications running millions of transactions. Once the blockchain network is functional, Managed Blockchain simplifies network management tasks by managing certificates, making it easy to create proposals for a vote among network members, and tracking operational metrics such as computing resources, memory, and storage resources.

Figure 13.13 shows the basic components of a Hyperledger Fabric blockchain running on AWS. A network includes one or more members with unique identities. For example, a member might be an organization in a consortium of banks. Each member runs one or more blockchain peer nodes to run chaincode, endorse transactions, and store a local copy of the ledger. Amazon Managed Blockchain creates and manages these components for each member in a network and also creates components shared by all members in a network, such as the Hyperledger Fabric ordering service and the general networking configuration.

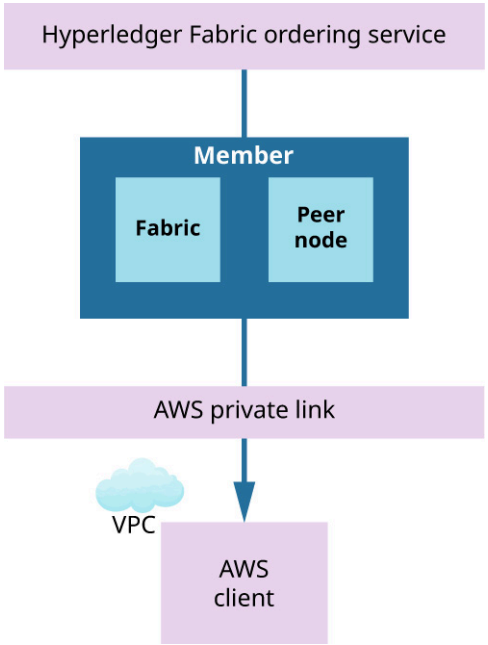


Figure 13.13 This graphic shows the basic components of a Hyperledger Fabric blockchain running on AWS via a Managed Blockchain network. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

When creating a Managed Blockchain network, the creator chooses the blockchain framework and the edition of Amazon Managed Blockchain to use, and this determines the capacity and capabilities of the network as a whole. The creator also must create the first Managed Blockchain network member. Additional members are added through a proposal and voting process. There is no charge for the network itself, but each member pays an hourly rate (billed per second) for their network membership. Charges vary depending on the edition of the network. Each member also pays for peer nodes, peer node storage, and the amount of data that the member writes to the network. The blockchain network remains active as long as there are members. The network is deleted only when the last member deletes itself from the network. No member or AWS account, even the creator's AWS account, can delete the network until they are the last member and delete themselves.

IBM Blockchain Services

IBM Blockchain Platform (IBP) is an IBM Cloud offering that is built on Fabric, a blockchain infrastructure provided by the open-source Hyperledger project. IBP provides an integrated developer experience with smart contracts that can be easily coded in Node, Golang, or Java. You can use the new IBM Blockchain VS Code extension to write client applications, based on the IBP console's integration of the Fabric SDK. IBP offers the possibility of deploying only the necessary components to connect to multiple channels and networks, while you maintain control of identities in your environment. Flexible and scalable, IBP can be run in any environment that IBM Cloud Private (ICP) supports, including LinuxONE. IBP simplifies the development and management of a blockchain network. It lets you accomplish the following tasks with just a few clicks in the easy-to-use interface:

- automated deployment of Fabric
- creation of custom governance policies
- initial development
- deployment of the application into production, including the creation of channels and deployment of chaincode
- inviting new members into the network and managing identity credentials over time

LinuxONE is engineered for high-performance, large-scale data and cloud services. A single LinuxONE platform consolidates hundreds of x86 cores. The platform's dedicated I/O processors allow you to move massive amounts of data while maintaining data integrity. The option to have dedicated cryptographic processors that supplement the standard CPUs means encryption for data at rest and for data in transit. Partitions within IBM's Secure Service Container (SSC) technology help to protect data and applications from internal and external threats.

The IBM Blockchain solution leverages Kubernetes (K8s), which is an open-source system for the automation of deployment, scaling, and management of containerized applications. The Kubernetes framework runs distributed systems resiliently and takes care of scaling requirements, failover, deployment patterns, and so forth. Kubernetes restarts containers that fail, replaces containers, kills containers that do not respond to your user-defined health check, and does not advertise them to be used until they are ready to serve. The key aspects of Kubernetes include the following:

- service discovery and load balancing
- storage orchestration
- automated rollouts and rollbacks
- automatic bin packing
- self-healing
- secret configuration management

Other components of the IBM Blockchain solution include IBM Cloud Private (ICP), GlusterFS, MIBM Secure Service Container, and the IBM Blockchain Platform. ICP is a private cloud platform for enterprises to develop and run workloads locally. It consists of PaaS and developer services that are needed to create, run, and

manage cloud applications. GlusterFS is a scalable network file system suitable for data-intensive tasks such as cloud storage and media streaming. It aggregates various storage servers into one large parallel network file system. IBM Secure Service Container (SSC) provides the base infrastructure on LinuxONE for container-based applications, either for hybrid or private cloud environments. This secure computing environment delivers tamper-resistant installation and runtime operations.

Oracle Blockchain Services

Oracle also provides a blockchain platform. As illustrated in Figure 13.14, Oracle's blockchain components include a network of validating nodes (i.e., peers), a distributed ledger (i.e., linked blocks, world state, and history database), an ordering service for creating blocks, and membership services for managing organizations in a permissioned blockchain.

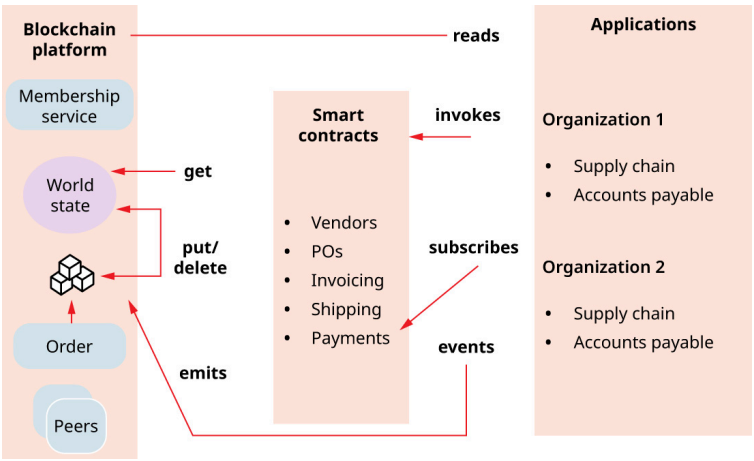


Figure 13.14 As this diagram shows, Oracle's blockchain components include a network of validating nodes, a distributed ledger, an ordering service for creating blocks, and membership services for managing organizations in a permitted blockchain. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The smart contracts (chaincode) layer consists of chaincode programs that contain the business logic for updating the ledger, querying data, and/or publishing events. Chaincodes can read the ledger data to verify conditions as part of any proposed updates or deletes and trigger custom events. Updates and deletes are proposed or simulated and are not final until transactions are committed following consensus and validation protocols. New or existing applications can register/enroll organizations as members, submit transactions (invoke smart contracts) to update or query data, and consume events emitted by the chaincodes or by the blockchain platform.

The Oracle Blockchain Platform is based on the Hyperledger Fabric project from the Linux Foundation, and it extends the open-source version of Hyperledger Fabric. Preassembled PaaS, Oracle's Blockchain Platform, includes all the dependencies required to support a blockchain network such as compute, storage, containers, identity services, event services, and management services. The Oracle Blockchain Platform includes the blockchain network console to support integrated operations.

GCP Blockchain Services

Google offers Blockchain Node Engine, a fully managed node hosting service for Web3 development that minimizes the need for node operations. Web3 companies that require dedicated nodes can relay transactions, deploy smart contracts, and read or write blockchain data. Blockchain Node Engine supports Ethereum, enabling developers to provision fully managed Ethereum nodes with secure blockchain access.

With blockchain, Google's focus is on cryptocurrency and blockchain analytics tools that provide deep blockchain transaction history data sets and deeper sets of queries that enable multichain meta-analysis and integration with conventional financial record processing systems. Google is particularly interested in providing transaction history for cryptocurrencies that have similar implementations, such as Bitcoin,

Ethereum, Bitcoin Cash, Dash, Dogecoin, Ethereum Classic, Litecoin, and Zcash. Google also offers machine learning tools that may be used to search for patterns in transaction flows and provide basic information on how a crypto address is used.

Extended Reality Services

When discussing extended reality services, there are generally two types: **virtual reality (VR)**, which enables a computer-generated, interactive, 3-D environment in which a user is immersed, and **augmented reality (AR)**, which supplements the real world with virtual (computer-generated) objects that appear to coexist in the same space as the real world. The key distinction between VR and AR is that VR is meant to immerse the user in a virtual environment, while AR introduces virtual elements to the real world. A VR system typically uses a headset in combination with a variety of sensors to track the user's movement and relay the appropriate images and feedback, creating the sensation of interacting with the virtual world. An AR system typically relies on clear lenses or a pass-through camera that allows users to see the world around them in real-time while virtual elements are projected on the lenses or rendered on the camera output.

There is also **extended reality (XR)**, a reality service that involves both real and virtual environments. Microsoft introduced the term **mixed reality (MR)**, which is a form of XR. XR encompasses both virtual and real environments and often integrates cloud services like IoT, ML, and blockchain. Three components are needed to make an XR system functional: a head-mounted display, a tracking system to recognize and follow physical objects, and mobile computing power. Today, XR is enabled via the use of headsets and haptic gloves, which are IoT devices. Instead of accessing applications with traditional computers, XR makes it possible to access applications through head-mounted hardware or gloves that interact directly with humans' senses, such as vision, hearing, and touch. In addition, ML may be needed within a mixed reality environment to do things such as generate prediction and perform recognition in a way similar to what we use ML for in the real world. In some sense, XR is the base technology for Webx.0 (i.e., the real metaverse, not Meta's Metaverse). Many XR commercial headsets are available for use, including Microsoft HoloLens, Google Cardboard, and Meta Quest.

Creating virtual reality scenes for VR or virtual objects or avatars that can be viewed via XR headsets requires the use of a 3-D engine tool such as Unity or Unreal Engine. Unity is a widely used cross-platform 3-D engine and integrated development environment (IDE). Its uses include developing 3-D content and games for different platforms, such as PCs, consoles, mobile devices, AR/VR target devices, and the Web.

Unity is a complex system with a steep learning curve. Successful deployment of applications also requires development frameworks and plug-ins, such as Microsoft Mixed Reality Toolkit (MRTK) or OpenXR. OpenXR can be accessed by 3-D engines (e.g., Unity, Unreal), the WebXR device API, as well as XR applications running on base stations to facilitate deployment to or integration with various devices including 3-D head mounted displays (e.g., Microsoft HoloLens, Apple Vision Pro, Meta Quest), trackers (e.g., body, hand, object, eye), haptic devices, and cloud/5G infrastructure.

XR applications provide controlled and repeatable scenarios rehearsing muscle memory and situational awareness. VR applications make it possible to explore places otherwise inaccessible and also have the potential to provide access to resources that may be prohibitively expensive or otherwise inaccessible. VR and AR applications provide innovative ways to visualize and manipulate data.

Azure MR Services and Related PaaS Services

Microsoft and the Azure Cloud provide various services for its Kinect and HoloLens products. This includes the Azure Kinect Sensor SDK, a developer kit with advanced AI sensors for building computer vision and speech models. Azure Kinect is a cutting-edge spatial computing developer kit with sophisticated computer vision and speech models, advanced AI sensors, and a range of powerful SDKs that can be connected to Azure cognitive services.

Microsoft also offers HoloLens 2, which is a set of smart glasses, and the HoloLens Emulator, both of which allow users to test holographic applications on a PC without a physical HoloLens 2 or HoloLens 1, including the HoloLens development toolset. Using the HoloLens emulator requires learning keyboard and mouse commands to facilitate walking in a given direction, looking in different directions, and making controlling gestures and hand movements.

The emulator uses a Hyper-V virtual machine, which means human and environmental inputs being read by HoloLens sensors are simulated from a keyboard, mouse, or Xbox controller. Users do not need to modify projects to run on the emulator because the apps do not recognize that they are not running on a real HoloLens. Users can join the HoloLens developer program, and learn to develop and deploy their own 3-D models.

Alternative development environments for HoloLens include Unreal Engine and BuildWagon. BuildWagon provides an online code editor that allows users to write code in JavaScript and view the results on the same screen or directly on the HoloLens. A HoloLens device is not required, and code is hosted on the cloud to allow multiple developers to collaborate on the same project from different locations. BuildWagon's HoloBuild library provides ready-made components to expedite creation processes and access HoloLens' special features.

The Microsoft Mixed Reality Toolkit (MRTK) is a Microsoft-driven project that provides a set of components and features used to accelerate cross-platform MR app development in Unity. It provides the cross-platform input system and building blocks for spatial interactions and UI, enabling rapid prototyping via in-editor simulation that enables users to see changes immediately. It operates as an extensible framework that provides developers with the ability to swap out core components while supporting a wide range of platforms.

Microsoft provides a detailed set of guidelines to assist with the development of mixed reality applications, covering application ideation, design, development, and distribution.

CONCEPTS IN PRACTICE

Innovation and Big Cloud Paas

Innovation is a great concept, and the use of big cloud PaaS services enables it. Before big cloud PaaS services became available, it was difficult for companies to put in place the services and related infrastructure needed to develop innovative solutions. All of the PaaS services that are covered in this chapter require frameworks and resources, which are provided by the big clouds; therefore, it is possible for companies to focus on applying these services to develop innovative solutions. As an example, you can simply go to azure.portal.com and type "Data Science Virtual Machine" in the search bar. You will then be provided with a choice of Linux or Windows VMs that come fully packed with all the framework and related APIs needed to implement the service. Microsoft Azure provides IoT Edge and IoT Hub frameworks that can be used to collect data from sensors located at the edge (e.g., weather sensors that measures temperature and humidity) and propagate the corresponding data to the Azure cloud so it can be analyzed to generate weather predictions. Therefore, big cloud PaaS services can be used today to enable and accelerate innovation.

Also available on the Azure Cloud are a number of PaaS services, including the following:

- Azure storage services may be used to store 3-D models.
- Azure Remote Rendering (ARR) is a service that lets you render highly complex 3-D models in real-time and stream them to a device. ARR is generally available and can be added to your Unity or Native C++ projects targeting HoloLens 2 or Windows desktop PC.
- Azure Object Anchors (AOA) is a mixed reality service that helps you create rich, immersive experiences by automatically aligning 3-D content with physical objects. It makes it possible to gain a contextual

understanding of objects without the need for markers or manual alignment. It also saves significant touch labor, reduces alignment errors, and improves user experiences by building mixed reality applications with Object Anchors.

- Azure Spatial Anchors (ASA) is a cross-platform service that allows you to build spatially aware mixed reality applications. With ASAs, you can map, persist, and share holographic content across multiple devices at a real-world scale. In particular, ASAs are used to create free-world anchors that persist across multiple application sessions.
- Azure Speech service is a speech resource that may be used to recognize speech, synthesize speech, get real-time translations, transcribe conversations, or integrate speech into your bot experience.
- Azure AI Vision is a cloud-based computer vision API that provides developers with access to advanced algorithms for processing images and returning information. When a user uploads an image or specifies an image URL, Microsoft Computer Vision algorithms can analyze visual content in different ways based on inputs and user choices.

GCP XR and Related PaaS Services

Google provides various XR and related PaaS services, including both Google AR and VR. The AR services include Google Lens, which can recognize things in images. AR in Google search lets you bring 3-D objects and animals into the world you see. Live View in Google maps changes how the world looks to add directions and other information. AR Stickers let you drop objects into photos taken with a Google Pixel camera. Google makes it possible for developers to develop AR applications using its ARCore Geospatial API. Google also provides VR capabilities, including Cardboard, which is a cardboard VR headset that uses a phone as a virtual world generator unit. DeepDream is a computer vision program that uses a convolutional neural network to find and enhance patterns in images to create dream-like appearances. Google also makes it possible for developers to develop their own VR applications.

Other XR and Related PaaS Services

Google isn't the only provider of XR and related PaaS services. For example, Meta Quest offers all-in-one VR headsets that developers can use to create a range of VR experiences, including mixed reality, designed for both work and play.

In another example, with the Amazon Sumerian Platform, Amazon develops XR tools and uses XR technology to support its retail businesses. For example, Amazon Sumerian makes it possible to create and run VR, AR, and 3-D applications quickly and easily without requiring any specialized programming or 3-D graphics expertise. It runs on popular hardware such as Meta Quest and Google Cardboard, as well as on Android and iOS mobile devices. Amazon Sumerian makes it possible to create virtual classrooms that let you train new employees around the world or enable people to tour a building remotely.

The NVIDIA Omniverse platform is an easily extensible platform for 3-D design collaboration and scalable multi-GPU, real-time, true-to-reality simulation. Omniverse revolutionizes the way individuals create, develop, and work together as teams, bringing more creative possibilities and efficiency to 3-D creators, developers, and enterprises.

YouTube also offers a VR experience through videos recorded with 360 or 3-D cameras. Through YouTube VR, users can experience things such as skydiving, snowmobiling, and a hot air balloon ride. To ensure it is VR, look for the compass icon in the upper left of a video.

3-D/4-D Printing Services

The process of **3-D printing**, formally known as additive manufacturing, is a process of designing static objects in three dimensions through additive processes in which successive layers of material are laid down under computer control. It is being used in applications such as medical prosthetics, aerospace components, and defense equipment. A 3-D modeling program, such as AutoCAD, is used for designing the objects. Various

cloud vendors are making 3-D printing technology available on the cloud today, such as Craftcloud, which allows users to essentially create an order for a custom 3-D printed part without having to actually own a 3-D printer. Users can upload their 3-D models to the Craftcloud platform, select their specifications, and receive their custom 3-D printed part in the mail.

The process of **4-D printing**, also supported as a platform as a service (PaaS) on some clouds, provides the capability of programming the fundamental materials used in 3-D printing by creating objects that can change their form or function after fabrication. It is scalable and can use cloud-based environments that streamline the development and deployment of smart materials and dynamic structures. These services offer advanced computational resources and specialized software tools needed for designing, simulating, and controlling 4-D printing processes, enhancing efficiency and innovation in creating adaptable and self-transforming products. 4-D printing adds the elements of time and interactivity to 3-D printing. 4-D printing creates objects with dynamics and performance capabilities, so they are able to change their form or function after fabrication. These objects can be assembled, disassembled, and then reassembled to form macroscale objects of desired shape and multifunctionality. This technology is based on three key capabilities: the machine, the material, and the geometric program. As an example, using this technology, the Stratasys material research group developed a new polymer that could be expanded 150% when submerged in water.

INDUSTRY SPOTLIGHT

3-D and 4-D Printing in Health Care

In health care, 3-D and 4-D printing has revolutionized imaging technology, improving processes such as mammography, radiation therapy, bronchoscopy, and ultrasounds. With benefits such as three-dimensional imaging, better delivery processes for drugs, tissue engineering, and more sophisticated medical devices, 3-D and 4-D technology improves the quality of images and enables health professionals to provide more accurate diagnoses and better targeted treatments, improving patient care and often leading to better outcomes.

Provide a specific example of how you think 3-D and 4-D printing are likely to improve health care in the next five years.

Applications Development Services

Various application development support capabilities are provided as PaaS services on the big clouds. These services help organizations improve operations and include integration management, identity and security management, application life cycle management, monitoring, and management and governance.

Integration Management

Integration management is a PaaS service that supports project management with tools for communication, project coordination, efficiency, and even conflict resolution. AWS, GCP, and IBM Cloud also provide integration management PaaS services and related capabilities.

Identity and Security Management

With identity and security management supported by PaaS, organizations can help ensure that only authorized users have access to their systems and applications. AWS, GCP, and IBM Cloud also provide identity and security management PaaS services and related capabilities.

Application Life Cycle Management

Application life cycle management is a tool that guides the software application process from planning until the software is decommissioned and retired. Various application life cycle management capabilities, including

DevOps and migration, are provided as PaaS services on the big clouds.

DevOps combines people, processes, and products to enable continuous delivery of value to end users. DevOps enables you to build, test, and deploy any application, either to the cloud or on premise. AWS, GCP, and IBM Cloud also provide DevOps PaaS services and related capabilities.

Migration services minimize the time and resources required to migrate an on-premises environment to the cloud. AWS, GCP, and IBM Cloud also migration services and related capabilities.

Monitoring

Typical monitoring services include application log analytics to drive resource autoscaling. Monitoring ensures that organizations realize when they have application issues that need immediate attention and areas where applications can perform better. Monitoring also provides data about applications that are underutilized and overloaded. AWS, GCP, and IBM Cloud also provide monitoring PaaS services and related capabilities.

Management and Governance

Generally, management and governance capabilities include recovery, cost management and billing, and other services. AWS, GCP, and IBM Cloud also provide management and governance PaaS services and related capabilities.

13.4 Towards Intelligent Autonomous Networked Super Systems

Learning Objectives

By the end of this section, you will be able to:

- Analyze specific applications of AI through XR technology
- Understand the impact of the development of supersociety capabilities, including nanotechnology, robotics, and supercomputers
- Discuss the advantages and challenges faced by the development of IANS and supersystems

Recent advances with superintelligent AI allow for the seamless vision of incorporating networked autonomous systems into reality. These systems, known as **intelligent autonomous networked supersystems (IANS)**, are becoming the next major development for chained computing, where intelligent chains of autonomous machines work together as a system to make decisions and take action. IANS are highly interconnected AIs, forming complex networks that collaborate while utilizing quick and extended exchange of data to promote growth.

In this section, we will analyze current applications of IANS and supersystems that are implemented and traffic large-scale systems and businesses with their applications. By going through real-life examples, we can evaluate the potential and critically examine the current challenges and limitations in the development and deployment of extensive IANS and similar supersystems that use intelligence to improve industries such as health care.

Web Platforms and Smart Ecosystems Applications

Today's web incorporates hybrid multiclouds that continuously evolve. As shown in Figure 13.15, these hybrid multiclouds power myriad innovative technology components that make it possible to create innovative solutions as the Web continues to evolve.

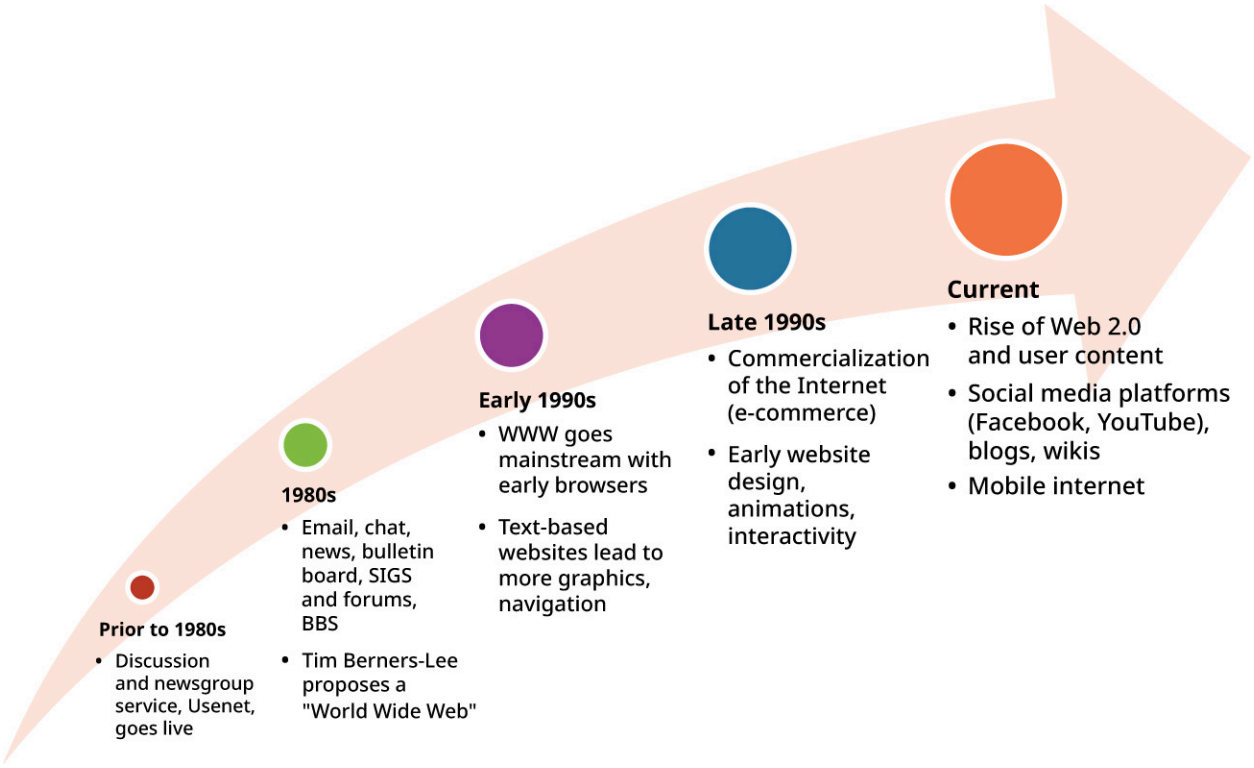


Figure 13.15 As this diagram shows, the Web has evolved and experienced many breakthroughs and disruptions. (credit: modification of "History of online service" by Viviensay/Wikimedia Commons, CC0)

In particular, recent advances in mobility and networking, such as 5G, have made it possible to minimize the latency of traditional web and mobile applications. This has led to a proliferation of social networks that enable efficient access to various types of content and global instant communication and collaboration. In addition, virtualization technology has made it possible to create powerful cloud platforms that facilitate access to infrastructure and platform services, as described earlier in this chapter.

This progress is leading to global acceptance of the next-generation hybrid Web 3.0, which makes it possible to combine traditional Web 2.0 applications with blockchain 2.0 capabilities. Blockchain is characterized by real-time transactions, scalability, and unlimited decentralized storage. Further improvements are on the horizon to provide a more scalable, fast, unlimited, and completely secure blockchain infrastructure via Blockchain 3.0/4.0.

To build on this, Web 4.0 and 5.0 are already on the way as the metaverse is being positioned as the successor to today's Internet. Metaverse is a concept that originated in the 1992 novel Snow Crash, in which people use the metaverse as an escape from a dystopian world (an idea also later explored in the novel and film *Ready Player One*). Metaverse embodies a unified immersive digital world that is tightly connected to the physical world. In the metaverse, people can interact without physical or geographic constraints and enjoy a compelling sense of social presence.

The metaverse is characterized by two key features. It is persistent, with its collective network of 3-D-rendered virtual elements and spaces available throughout the world 24/7. It is also shared, giving a vast number of users simultaneous access and the ability to use the metaverse to interact. The metaverse functions with six key layers that include

- infrastructure (e.g., chips and processors, cloud infrastructure),
- access/interface (e.g., haptics, headsets, smartglasses),
- virtualization tools (e.g., 3-D design engines, avatar development),
- virtual worlds (centralized and decentralized),
- economic infrastructure (e.g., payments, crypto wallets, and non-fungible token (NFT) marketplaces using

NFTs with a digital signature that cannot be exchanged or equated to another item), and
- experiences (e.g., gaming, virtual real estate/concerts).

Industry Applications

As of 2024, the online worlds promoted by metaverse proponents are not fully formed and functional just yet, but platforms and games like Second Life, Roblox, and Decentraland are indicators of the future once the metaverse is fully operational. The metaverse is expected to rely on technologies such as virtual reality headsets, advanced haptic feedback, and 3-D modeling tools to power immersive digital environments. To understand the potential of the metaverse and mixed reality technology, let's consider their applications in the areas of education and health care.

Enhanced Learning Experiences

Mixed reality can provide visuals and relatable examples to help students perceive theoretical information in complex topics such as biology, anatomy, physics, and math. Medical students can learn anatomy and practice examining the body with XR apps that represent the human body inside and out. Chemistry students can conduct experiments using different chemical combinations and see results with no harm to students and school property. Nursing students can use simulations to learn and prepare for unique situations that they'll encounter in clinical settings. Students can also take field trips to museums, exhibitions, and theaters all over the world without leaving the classroom.

In-Person Patient Care Applications

In operating rooms, clinics, hospital wards, and medical training settings, mixed reality speeds up diagnoses, increases access to health-care facilities, cuts down on infection transmissions, and improves medical care outcomes. XR enables holographic overlaying of images and data onto real-life situations such as surgical operations, remote consultation, and treatment, opening new avenues in health care.

In cardiology, initial XR health-care applications created interactive visualizations that enabled pediatric cardiologists to virtually demonstrate complex congenital heart problems to their students and patients. XR techniques reduce the time required to diagnose cardiology issues, and surgeons who use Microsoft HoloLens headsets during procedures can interact with the hologram using hand movements and benefit from a wider field of view, which improves surgical outcomes. In one application, XR technology enables surgeons to see patients' 3-D computed tomography (CT) and magnetic resonance imaging (MRI) scans directly. This will make it easier for surgeons to identify the exact area of the patient's body that needs surgery. This could be especially beneficial for emergency surgeries that must be performed as quickly as possible to save a patient's life. Preoperative simulations are made easier with XR, which creates customized 3-D models for each patient and visualizes the inside anatomy in a fully immersive environment. In complex surgical operations such as reconstructive surgeries, holographic overlays can substantially help surgeons examine the bones and determine the flow of blood arteries. XR may also be helpful for pain relief. In Denmark, Aalborg University researchers studied the potential of XR to provide pain relief for phantom limbs. It may be possible to delude the brain of a person with an amputation into thinking it still controls their missing limb, which may assist in reducing the agony associated with phantom limbs.

Telemedicine

Using XR headsets and 3-D XR, medical practitioners are able to review patient histories vocally, discuss with medical specialists, and record patient records. XR-powered headsets can eliminate the need for doctors to review written reports, analyze patient data, and deliver findings in real time, resulting in faster and more precise diagnostics.

XR may also provide paramedics with remote support to address emergencies. With XR, paramedics can remotely get support from senior medical professionals. This can help them make more accurate and faster medical decisions, efficiently provide emergency medical aid, and improve patient outcomes.

Lastly, XR can help provide remote care to patients with mobility issues. It can also visually project simulations for different situations, offer ease of access to facilitators remotely, increase patient engagement by providing a safe and controlled immersive environment, and leverage telehealth appointments.

Therapeutic and Mental Health Applications

The Autism Glass Project at Stanford University's medical school used XR to help children with autism manage their emotions and identify related facial expressions. There are many other possible applications of the same technology, including the use of VR psychotherapy to address mental health conditions and disorders and treat the cause rather than the effect of such disorders by combining VR technology with big data analytics, cloud computing, machine learning, IoT, and blockchain. Affective Interaction through Wearable Computing and Cloud Technology (AIWAC) technology provides a full-stack solution aiming at effective remote emotional health-care assistance. The solution components allow collaborative data collection via wearable devices, enhanced sentiment analysis and forecasting models, and controllable affective interactions.

Other Applications of XR Technology

As the metaverse evolves, XR-driven technology may be applied in various industries for uses such as the following:

- equipment assembly, maintenance, and repair
- engineering and architectural design (e.g., experiencing a virtual building before it is built)
- market research (e.g., experiencing a virtual product that does not yet exist)
- entertainment (e.g., cinema, music, and sports)
- product advertising and promotion
- computer games

Mixed reality technology also has the capacity to promote social good. For example, applications that combine XR with other innovative technologies to support people with disabilities have already been developed, including technology that alerts people who are blind when rapidly moving objects are headed in their direction.

LINK TO LEARNING

Microsoft offers mixed reality guidance and allows users to try the technology for free to learn how it can improve workplace processes. According to Microsoft, mixed reality technology can help solve problems while improving productivity and maximizing efficiency. For example, companies such as PCL Construction have successfully utilized Microsoft Mixed Reality, including HoloLens and Dynamics 365 Guides, to enhance project management and on-site operations.

PCL visualized 3-D blueprints directly on-site and provided interactive training through mixed reality, resulting in improved project accuracy, a 30% reduction in completion times, and a 25% decrease in costs. The technology also enhanced safety protocols, ensuring compliance and minimizing on-site accidents. You can read more about PCL's experience with mixed reality technology (https://openstax.org/r/76PCLexperience) and explore other examples.

Evolving Considerations for Standards and Guidelines

The use of immersive applications in the metaverse has made it necessary to update the standards and guidelines used to develop applications. Traditional usability guidelines were designed for web and mobile applications that forced users to access applications with computers. Immersive applications force designers to understand how human beings sense things, reason, and plan actions, as well as what motivates people to use available solutions. Designing for these applications requires paying attention to usability, accessibility,

and inclusion.

The process of designing products to be effective, efficient, and satisfying is called usability; per the World Wide Web Consortium (W3C) standards, it also calls for accessibility and inclusion. It subsumes solution qualities (e.g., effectiveness, efficiency) that are appealing to humans and motivate the use of such solutions. The concept of **accessibility** addresses discriminatory aspects related to equivalent user experience for people with disabilities. It pertains to solution qualities that make these solutions usable by people with disabilities. For example, web pages that display pictures should have "alt-text" HTML tags associated with the pictures that can be used to read what the pictures contain so that people who are blind can navigate to pages and understand what is displayed within pictures they cannot see. In general, people with disabilities should be able to perceive, understand, navigate, interact, and leverage websites and related tools in much the same way as all people do. The concept of **inclusion** ensures that diverse communities can make use of solutions regardless of their location, culture, and other differentiating traits, habits, or interests.

LINK TO LEARNING

Read these current usability guidelines (https://openstax.org/r/76UsablityGuide) to learn how to create websites that are accessible to everyone, accessibility, usability, and inclusion are important concepts.

An important aspect of designing applications is human-computer interaction (HCI), the science that studies interactions between people and computers and evaluates whether computers can successfully interact with humans. As illustrated in Figure 13.16, "HCI is concerned with understanding the influence technology has on how people think, value, feel, and relate and using this understanding to inform technology design."[1]

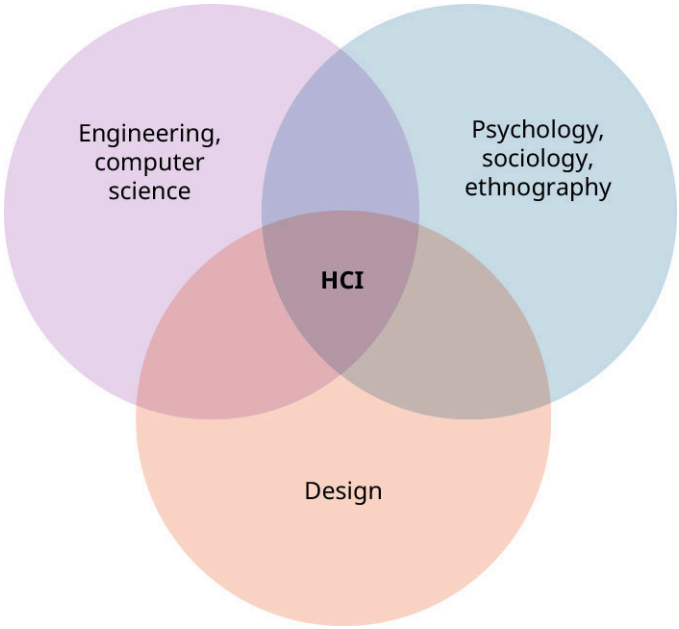


Figure 13.16 Human-computer interaction (HCI) studies the interaction of humans with technology to understand how technology influences human behavior. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The focus of HCI is to ensure that solutions are usable by humans. Usable solutions should be easy to use as well as effective, safe, efficient, and fun for the user. HCI also focuses on the creation of methods that may be used to measure and otherwise evaluate usability, as well as on the definition of usability guidelines and standards. Applying HCI as part of the design of computing systems requires considering human physical and mental capabilities (e.g., attention, memory) as well as the needs of humans (e.g., functional, emotional, social)

1 P. C. Wright and J. C. McCarthy, "Empathy and experience in HCI," Conference: Proceedings of the 2008 Conference on Human Factors in Computing Systems, CHI 2008, 2008, Florence, Italy, April 5-10, 2008. http://dx.doi.org/10.1145/1357054.1357156

as constraints at the same level as machine physical constraints such as processor speed and networking capabilities.

A usable system must understand the various roles of humans. This includes the following:

- Humans as sensory processors. Usability results when the system fits within human sensory limits for vision, hearing, touch, smell, and taste.
- Humans as interpreters/predictors. Usability results when the system fits with human knowledge. This includes the ability to process information via perception and cognitive processes such as selective attention, learning, problem-solving, and language processing.
- Humans as actors in environments. Usability results when the system fits within task and social contexts, such as gender and ethnic backgrounds.

LINK TO LEARNING

HCI guidelines (https://openstax.org/r/76HCIguidelines) are an important tool to ensure that technology is user friendly, as well as efficient and accessible.

Human Computer Interaction (HCI) Guidelines for Immersive Solutions

The most important solution development steps in HCI are to define the context. This includes the type of uses and applications—such as industrial, commercial, and exploratory—as well as the market and the customer. The context is not the specific local environment, but rather the larger type of world that the system needs to exist in. This includes the users' physical attributes, physical workspaces, perceptual abilities, cognitive abilities, personality and social traits, cultural and international diversity, and special abilities or disabilities. It also includes task analysis to understand what users need and want to do with technology. Other steps involve function allocation, system layout/basic design, mockups and prototypes, usability testing, iterative testing and redesign, and update and maintenance.

Similar to mobile solutions, immersive products must go through a series of prototypes to ensure stabilization, feasibility (a single logic path), alpha prototype (minimum viable product), beta prototype (largely complete), and release candidate (all required functionality) ready for product owner review. Quality measurement checklists and design best practices are different for web, mobile, and immersive solutions.

THINK IT THROUGH

VR Accessibility

Ricardo's friends are having fun using virtual reality and avatars to explore castles in Europe. But Ricardo has a disability that prevents him from joining in the fun using the website his friends have selected.

Why is this important? How could HCI guidelines help the developers of this website make virtual reality accessible to Ricardo?

Supersociety Digital Solutions

While smart ecosystem solutions focus on providing insights to their users so they can adapt to change and optimize their activities to guarantee success, supersociety applications go one step beyond by replacing humans in certain mechanical activities and allowing them to focus on activities that machines are not able to perform on their behalf. The set of supersociety capabilities that are being developed keep evolving with capabilities supported by innovative technology components powered by the hybrid multiclouds that are an inherent part of our evolving web infrastructure.

Supersociety Capabilities

Noteworthy supersociety capabilities being developed today include technology at the molecular level, robotics and advanced robotics, supercomputers, and intelligent autonomous networked systems and supersystems.

Nanotechnology

The field of **nanotechnology** focuses on matter at the molecular level to create structures and devices about 1 to 100 nm in size with fundamentally new organization, properties, and performance. Nanotechnology can reduce the size of storage devices available via hybrid multiclouds, making it possible to drastically increase the volume of information available on the Web. Some researchers have also suggested a quite futuristic Internet of Thoughts in which neural nanorobots could be used to connect the neocortex of the human brain (i.e., the smartest conscious part of the brain) to a "synthetic neocortex" in the cloud. If doable, this could enable the creation of a future "global superbrain" that would connect networks of individual human brains and AIs to enable collective thought.

Challenges associated with nanoscale science and technology include making nanomaterials (e.g., self-assembly, top-down vs. bottom-up), characterizing nanostructures (e.g., imaging and measuring small things), understanding properties ("nanoland" lies between macroworld and single atoms and molecules), and nanosystems integration and performance (i.e., how we assemble nanostructures into systems). To better understand these challenges, consider Figure 13.17.

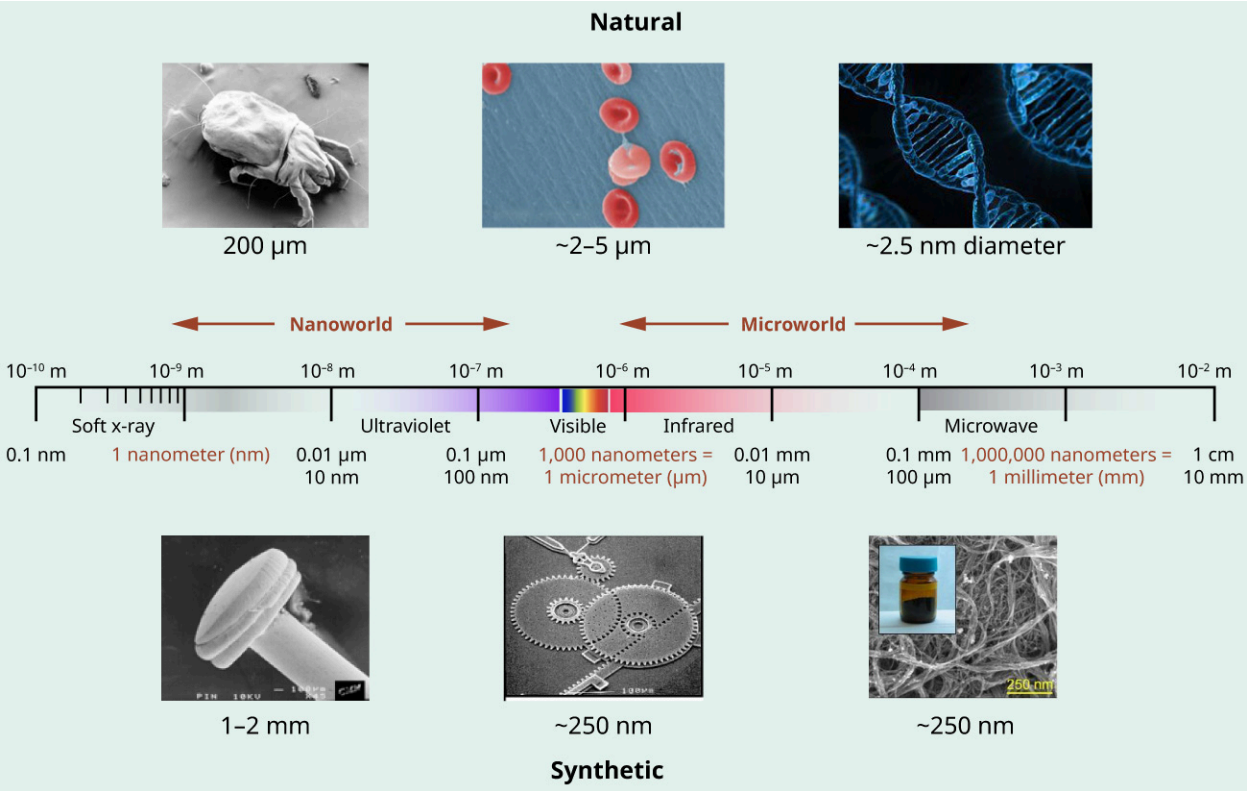


Figure 13.17 Nanotechnology faces many challenges as matter is used at the molecular level to create structures and devices that are ~1 to 100 nm in size with fundamentally new organization, properties, and performance. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit ruler: modification of "The Scale of Things - Nanometers and More" by NIST/nist.gov, Public Domain; credit top left image: modification of "CSIRO ScienceImage 11085 A scanning electron micrograph of a female dust mite" by Matt Colloff/Wikimedia Commons, CC BY 3.0; credit top middle image: modification of "This digitally-colorized scanning electron micrograph (SEM) revealed some of the ultrastructural morphology displayed by red blood cells" by CDC/Public Health Image Library, Public Domain; credit top right image: modification of "Dna-163466" by PublicDomainPictures/Wikimedia Commons, CC0; credit bottom left image: modification of "Head of a pin" by NIST/nist.gov, Public Domain; credit bottom middle image: modification of "Model of a MEMS Safety Switch," Courtesy Sandia National Laboratories, SUMMiT™ Technologies, www.sandia.gov/mstc; credit bottom right image: modification of "Carbon Nanotube Reference Materials" by NIST/nist.gov, Public Domain)

Robotics and Advanced Robotics

Robotics and advanced robotics are joint disciplines that include computer science and mechanical and electrical engineering. The field of **robotics** focuses on the design, development, functioning, and application of robots, as well as the computer systems needed to control the robots, provide sensory feedback, and process information. Swarm robotics emphasizes a large number of robots and promotes scalability. A **cyborg** is a biological human with parts replaced with machinery, while machines with biological parts added are considered to be an **artificial human**. Google's Cloud Robotics Core is an open-source platform that facilitates the management of robot fleets as well as the creation and operation of robotics-packaged solutions that automate business tasks. Other big cloud platforms also provide support for robotics and advanced robotics.

Software Robots

Software robots, or bots (e.g., web crawlers, chatbots), are computer programs that operate autonomously to complete a virtual task. They are not physical robots; instead, they exist only within a computer.

Another field of robotics, **cognitive robotics**, is a field that creates robots that can think, perceive, learn, remember, reason, and interact. It focuses on creating robots that mimic human perception, reasoning, and planning abilities. One subspecialty, **biomimetic robotics**, focuses on the design of robots that leverage principles common in nature, such as what can be learned from the evolution and development of intelligence in animals and humans. Recent progress and directions in AI, machine learning, and cognitive science drive the focus of the next generation of robotic systems. Figure 13.18 shows how these areas overlap.

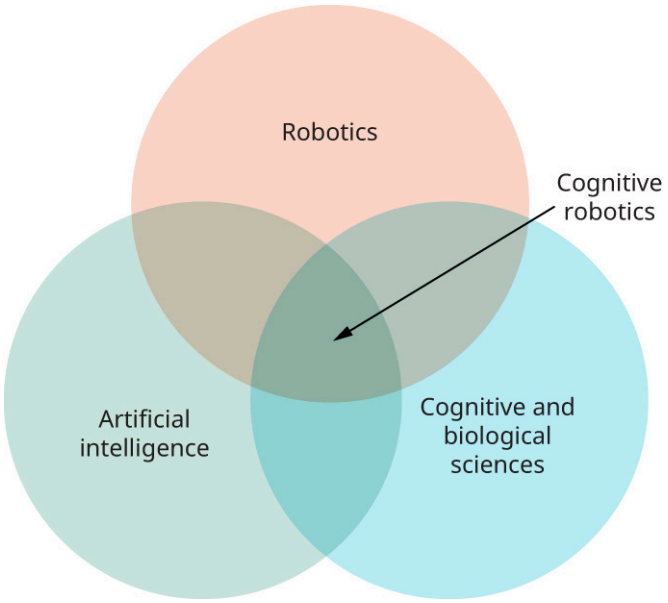


Figure 13.18 Cognitive robotics draws from the fields of cognitive and biological sciences and artificial intelligence, as well as robotics. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The intent of cognitive robotics is to replace humans in dangerous environments or manufacturing processes or resemble humans in cognition, enabling robots to do jobs that are hazardous to people. Cognitive robotics aims to improve robots' perception capabilities as they navigate and manipulate objects in a given environment and interact with people. It also makes it possible for robots to perform tasks by predicting the actions of people around them as well as their own. Cognitive robots can also perceive how people see the world, predict what they need, and anticipate their actions. This explains how these robots can execute daily tasks while interacting safely with people. They are capable of direct interactions, such as assisting customers, as well as indirect interactions, such as sweeping the floor while customers are shopping in a store.

Intelligent mobile robots that can move independently were introduced during the Second World War. Following the implementation of artificial intelligence (AI) in robotics, they became autonomous or more

intelligent. Figure 13.19 provides a list of components and architecture of modern intelligent robots.

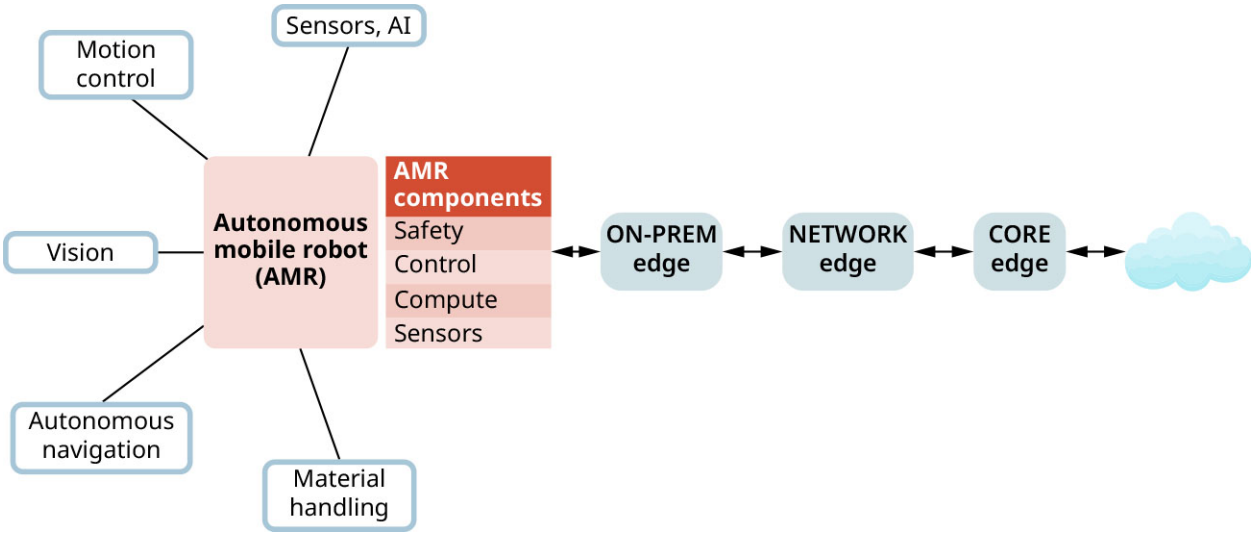


Figure 13.19 Modern, intelligent robots rely on various components and architecture to function as intended. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Robot Operating System

The Robot Operating System (ROS) is a meta-operating system specially designed for robots. It is open-source and supports a variety of services to control robotics hardware, provide hardware abstractions, and perform common tasks. It can also help manage software packages and pass messages from one process to another. ROS also includes various libraries and tools to facilitate the selection, development, and operation of software modules across various computers.

Robot Manipulators and Mobile Robots Characteristics

Robots include robot manipulators and mobile robots. A **robot manipulator** is a physical tool that operates at a fixed location to catch and move items. A **mobile robot** is one that can navigate from one position to another. Robot manipulators face the challenge of being able to pick and place objects with a sufficient degree of precision, while mobile robots must be able to estimate relative and absolute robot positions and navigate on a map.

Mobile robots are used in applications such as medical treatment, mail delivery, infrastructure inspections, and passenger travel. For example, Nao is a humanoid robot that is specially designed to interact with humans. It is loaded with sensors that enable it to mimic emotions. It can recognize people's faces as well as objects and can speak, walk, and dance. Nao was created by Aldebaran Robotics, which was acquired by SoftBank in 2015. Sixth-generation Nao robots are used in research as well as in the health-care and education industries.

Atlas is one of the most agile robots in existence. It uses whole-body skills to move quickly and balance dynamically. While Atlas can lift and carry objects such as boxes and crates, the robot can also run, jump, and do backflips.

LINK TO LEARNING

Learn more about mobile robots by exploring this website about Atlas (https://openstax.org/r/76Atlas) from Boston Dynamics. Atlas is one of several humanoid robots you can learn about.

Zipline is an autonomous fixed-wing aircraft drone used to carry blood and medicine from a distribution center to wherever it is needed. It can launch within minutes and travel in any weather.

As these examples show, mobile robots have many applications. Figure 13.20 provides an overview of the fields and industries that find mobile robots useful.

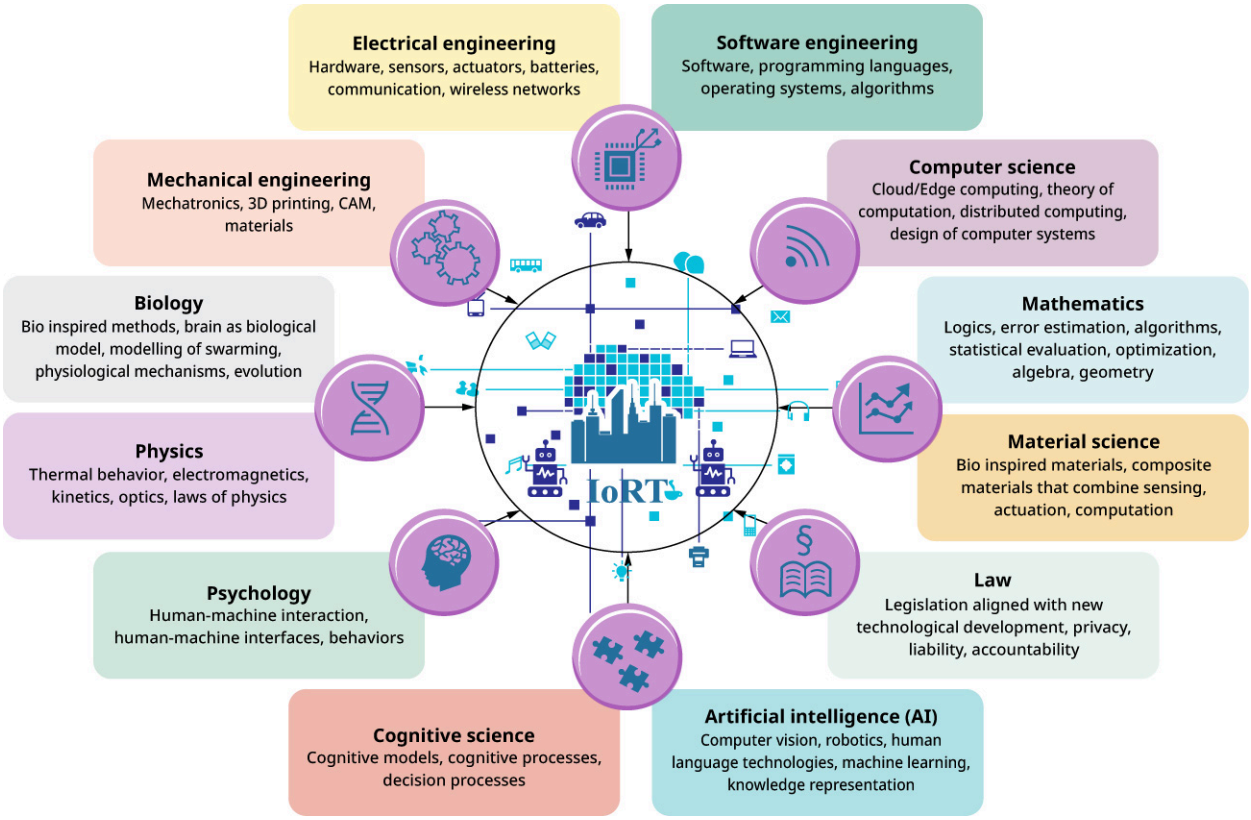


Figure 13.20 Mobile robots have a variety of applications in various fields and industries, including engineering specialties, science, mathematics, and law. (credit: Vermesan, O., Bahr, R., Ottella, M., Serrano, M., Karlsen, T., Wahlstrøm, T., Sand, H. E., Ashwathnarayan, M., & Gamba, M. T. (2020). Internet of Robotic Things Intelligent Connectivity and Platforms. Frontiers in robotics and AI, 7, 104. https://doi.org/10.3389/frobt.2020.00104)

Computer Vision and Cognitive Robotics

Computer vision advances have facilitated the positioning and navigation of mobile robots. Computer vision is achieved using optics and sensors, which involve image acquisition, image representation, and image processing.

LINK TO LEARNING

The Open Source Computer Vision Library (OpenCV) is a machine learning software library (https://openstax.org/r/76MachineLearn) built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in commercial products.

Artificial Cognitive Systems

Robots need artificial cognitive systems that simulate human thought processes to supplement human cognition. Artificial cognitive systems are developed using algorithms in artificial intelligence and technologies such as machine learning, deep learning, speech recognition, and object recognition.

Supercomputers

Various supercomputers are being developed based on neuromorphic computing and quantum technology. The supercomputers will eventually be available on the big clouds to help optimize the performance and

throughput of supersociety applications. IBM already provides quantum compute access plans for its cloud, and Intel is building neuromorphic computing hardware that can be leveraged on cloud supercomputers.

Cognitive Sciences and Neuroinformatics

The computing approach that combines the use of AI and cognitive science is called cognitive computing. The study of how to build a computer that can mimic basic human brain functions is called **neuroinformatics**. This requires robots' ability to handle ambiguity as well as uncertainty. Robots that are equipped with these capabilities can mimic humans' ability to memorize information, learn, reason, react, and show emotions.

Table 13.3 outlines related areas in cognitive sciences and technology support.

Subject Area	Brief Description	Technology Support
Artificial intelligence	Study of cognitive phenomena to implement human intelligence in computers	Pattern recognition, robotics, computer vision, speech processing
Learning and memory	Study of human learning and memory mechanisms to build them on future computers	Machine learning, database systems, memory enhancement
Languages and linguistics	Study of how linguistics and language are learned and acquired, and how to understand novel sentences	Language and speech processing, machine translation
Perception and action	Study of the ability to take in information via the senses such as vision and hearing; haptic, olfactory, and gustatory stimuli fall into this domain	Image recognition and understanding, behavioral science, brain imaging, psychology, and anthropology
Neuroinformatics	The intersection of neuroscience and information science	Neurocomputers, artificial neural nets, deep learning, aging, disease control
Knowledge engineering	The study of big data analysis, knowledge discovery, and the transformation and creativity process	Datamining, data analytics, knowledge discovery, and system construction

Table 13.3 Cognitive Science and Technology Support

Desired features of cognitive computing systems include the following:

- adaptive learning, which is learning as information changes and as goals and requirements evolve, resolving ambiguity and tolerating unpredictability
- interaction with users, allowing users—which may include other processors, devices, and cloud services, as well as people—to define their needs as a cognitive system trainer
- ability to be iterative and stateful, which means the system may remember previous interactions and may redefine a problem by asking questions or finding additional source input if a problem statement is ambiguous or incomplete
- contextual in information discovery, which means the system may understand and extract meaning, syntax, time, location, appropriate domain, regulations, user profiles, process, tasks, and goals, and respond to sensory inputs with visual and gestural effects

The real-world applications of cognitive systems include speech understanding, sentiment analysis, facial recognition, election insights, autonomous driving, and deep learning applications. Deep learning, in particular, requires specific hardware, including graphic processors, digital signal processors, field programmable logic devices, systems on a chip, custom microchips, and application-specific integrated circuits. Some of the providers of deep learning hardware include the Google Neural Machine Translation System (GNMT), Google Cloud's Tensor Processing Unit (TPU), TensorFlow, Cambricon's Neural Processing Unit (NPU), Intel's Movidius Neural Compute Stick (NCS), the Intel Movidius Neural Compute SDK, and Intel's Movidius Vision Processing Unit (VPU).

Neuromorphic Computing

Cognitive science is interdisciplinary in nature. It covers the areas of psychology, artificial intelligence, neuroscience, and linguistics. It spans many levels of analysis, from low-level machine learning and decision mechanisms to high-level neural circuitry to build brain-modeled computers. It applies software libraries on clouds or supercomputers for machine learning and neuroinformatics studies. It uses representation and algorithms to relate the inputs and outputs of artificial neural computers. It also designs hardware neural chips to implement brain-like computers referred to as neuromorphic computers, as illustrated in Figure 13.21.

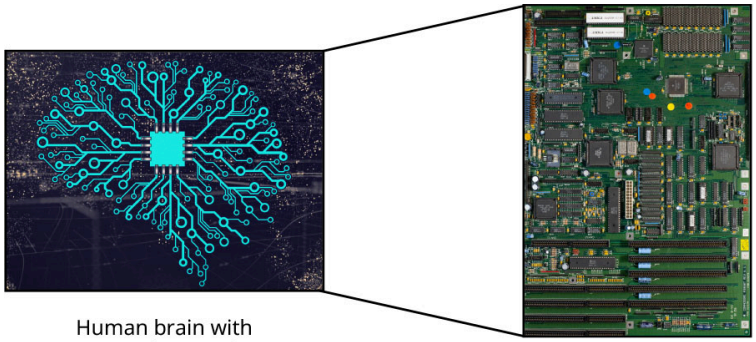


Figure 13.21 As a field, cognitive science has made many advances, including the design of hardware neural chips, such as those pictured here, which are used to implement brain-like computers known as neuromorphic computers. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit left image: modification of "Artificial Neural Network with Chip" by mikemacmarketing/Wikimedia Commons, CC BY 2.0; credit right image: modification of "Amiga 3000T motherboard without annotations" by Podstawko/Wikimedia Commons, CC0)

In 1990, Carver Mead, the Gordon and Betty Moore Professor Emeritus of Engineering and Applied Science for the California Institute of Technology, introduced the term **neuromorphic computing**, which relies on a hardware architecture that models how the human brain uses neurons. This provides the potential for faster, more complex computations while remaining power efficient. This field of computing emerged to compete with traditional computer architectures. Machine learning became popular, and as it advanced, neuromorphic computing became the best platform for machine learning algorithms. Comprised of a network of neurons and synapses, neuromorphic computing relies on hardware architecture modeled after the human brain. A neuron is a function that operates on an input. A synapse, which processes neuron output and passes a state to another neuron, can be trained to know how to convert neuron output to states. A memristor is a component that remembers the charge of an electric current. Memristors are great for neuromorphic computing as they provide neuroplasticity. Neurons pulse electric signals as input. Output is based on the path taken through the network, and it is a highly connected and parallel architecture that uses side-by-side memory and processing while consuming a low amount of power. Various neuromorphic computing models have been developed on this type of hardware, including the Neuroscience-Inspired Dynamic Architecture (NIDA), Dynamic Adaptive Neural Network Arrays (DANNAs), and Memristive Dynamic Adaptive Neural Network Arrays (mrDANNAs). Intel Labs developed the Loihi 2 neuromorphic chip that is used for research along with an open-source framework known as Lava. Intel's goal is to facilitate the adoption of neuromorphic computing. Intel Labs' second-generation neuromorphic research chip, codenamed Loihi 2, and Lava, an open-

source software framework, will drive innovation and adoption of neuromorphic computing solutions.

Quantum Computing

Quantum computing has applications in experimental physics. It provides a theory that is more fundamental than Newtonian mechanics and electromagnetism and can explain phenomena that these theories cannot tackle.

A **quantum computer** is a machine that performs calculations based on the laws and principles of quantum mechanics, in which the smallest particles of light and matter can be in different places at the same time. Information is stored in a physical medium and manipulated by physical processes. Designs of "classical" computers are implicitly based in the classical framework for physics and can only deal with bits, not qubits. This classical framework has been replaced by the more powerful framework of quantum mechanics. In a quantum computer, one qubit (a quantum bit) could be both 0 and 1 at the same time. So, as Figure 13.22 shows, with three qubits of data, a quantum computer could store all eight combinations of 0 and 1 simultaneously. That means a three-qubit quantum computer could potentially process information more efficiently than a classical three-bit digital computer, depending on the algorithm.

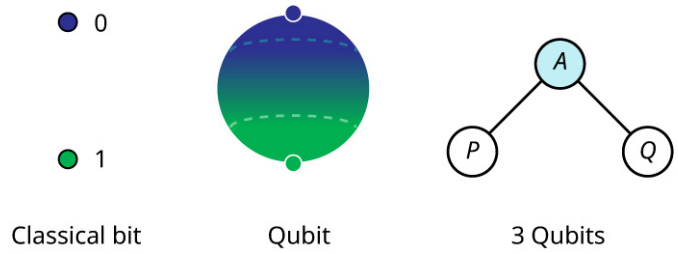


Figure 13.22 In a quantum computer, one qubit, or quantum bit, could be both 0 and 1 at the same time, enabling a quantum computer to store all eight combinations of 0 and 1 simultaneously. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Typical personal computers today calculate 64 bits of data at a time. A quantum computer with 64 qubits would be 2^{64} faster, or about 18 billion times faster. A bit of data is represented by a single atom that is in one of two states, denoted by |0> and |1>. A single bit of this form is known as a qubit. A physical implementation of a qubit could use the two energy levels of an atom. An excited state represents |1>, and a ground state represents |0>. A single qubit can be forced into a superposition of the two states denoted by the addition of the state vectors:

$$|\psi> = \alpha|0> + \alpha|1>$$

where α and α are complex numbers and $|\alpha| + |\alpha| = 1$.

A qubit in superposition is in both of the states |1> and |0 at the same time.

In general, an *n* qubit register can represent the numbers 0 through 2^n–1 simultaneously. Entanglement is the ability of quantum systems to exhibit correlations between states within a superposition. Imagine two qubits, each in the state |0> + |1> (a superposition of the 0 and 1). We can entangle the two qubits such that the measurement of one qubit is always correlated to the measurement of the other qubit.

However, if we attempt to retrieve the values represented within a superposition, the superposition randomly collapses to represent just one of the original values. This means that a wave function changed and instead of the superposition state continuing, the superposition has collapsed into a single state that has a defined value representing only one of the original values.

In the classical computing model, a probabilistic Turing machine (PTM) is an abstract model of the modern (classical) computer. The strong Church-Turing thesis states that a PTM can efficiently simulate any realistic model of computing, meaning that if a problem is difficult for a PTM, it must also be difficult for any other reasonable computing model. For example, factoring is believed to be hard to perform on a Turing machine (or any equivalent model). We do not know whether there is some novel architecture on which factoring is

easy. Because we lack certainty, we assume that certain computational problems, such as factoring, possess inherent complexity regardless of the effort put into finding an efficient algorithm.

In the early 1980s, Richard Feynman noted that it appeared unlikely for a PTM to efficiently simulate quantum mechanical systems. Because quantum computers operate as quantum mechanical systems, the model of quantum computing appears to challenge the strong Church-Turing thesis.

Possible applications of quantum computing include efficient simulations of quantum systems, phase estimation, improved time-frequency and other measurement standards such as GPS, factoring and discrete logarithms, hidden subgroup problems, and amplitude amplification. Possible implementations of quantum systems include optical photon computers, nuclear magnetic resonance (NMR), ion traps, and solid-state quantum. The optical photon computer operates through the interaction between an atom and a photon inside a resonator, while another approach employs optical devices such as a beam splitter and mirror. NMR represents qubits using the spin of atomic nuclei, with chemical bonds between these spins manipulated by a magnetic field to simulate gates. The spins are initialized by magnetization, and measurement is achieved by detecting induced voltages. Currently, it is believed that NMR will not scale beyond about twenty qubits. However, in 2006, researchers reached a 12-coherence state, showing that scalability up to 12 qubits is feasible using liquid-state nuclear magnetic resonance quantum information processors. Ion traps form qubits using two electron orbits of an ion (charged atom) confined in a vacuum by an electromagnetic field. Additionally, there are two widely recognized solid-state implementations of qubits:

1. A qubit formed through a superconducting circuit using a Josephson junction, which establishes a weak link between two superconductors. A Josephson junction consists of two superconductors separated by a very thin insulating barrier.
2. A qubit formed using a semiconductor quantum dot, a nanostructure ranging from ten to several hundred nanometers in size, designed to confine a single electron.

Many papers have explored various aspects of quantum computing, including detailed language specifications. For more information about any of the following examples, perform some further research.

- Quantum computation language (QCL) by Bernhard Ömer: a C-like syntax and very complete
- Quantum Guarded-Command Language (qGCL) by Paolo Zuliani and others: a high-level imperative language for quantum computing
- Quantum C by Stephen Blaha: currently just a specification

GLOBAL ISSUES IN TECHNOLOGY

Quantum Computing in Global Operations

Quantum computing is impacting industries throughout the world and improving global operations in fields such as banking, health care, manufacturing, and transportation. For example, quantum computing has been used to develop new drugs, design aircraft that are safer and more efficient, provide more robust encryption and online security, and predict the weather with greater accuracy. Quantum computing has the ability to solve problems faster and more effectively compared to classical computing.

In 2023, the value of quantum computing's global market size was $885.4 million (USD), and it is expected to increase to $12.6 billion by 2032, representing a growth of 34.8%. While North America currently holds the largest share of the quantum computing market, other parts of the world, particularly Europe and Asia, are expected to grow over the next few years.

Intelligent Autonomous Networked Systems and Supersystems

Intelligent autonomous networked systems and supersystems leverage the various supersociety technologies

discussed so far. However, one of the missing links appears to be the lack of understanding of what drives human reasoning and planning, which has led to the inability to mimic it to create systems based on artificial general intelligence (AGI). Also, the current approach to deep learning requires using a very large amount of data to train machine learning algorithms and create usable models, which are extremely time-consuming, as well as requiring tremendous resources such as data and processing power. Researchers are striving to come up with solutions to these problems.

IANS have the potential to leverage the innovative capabilities of AI, ML, edge computing, and virtualization to offer better human experiences in interconnectivity. Compared to automated networks, which have explicitly defined inputs and outputs in predictable environments, autonomous networks improve operations when functioning in unpredictable environments with conditions lacking inputs and outputs that can be tested in advance. With IANS, the systems can learn and adapt as conditions change, adjusting to meet whatever needs arise. AGIs are particularly useful in this environment, as they have the ability to work independently, adapting and making adjustments as needed when conditions change.

Knowledge management and reuse of processes and information are being looked into as well. A possible approach to facilitate reasoning in the AGI grand scheme consists of using an open world decentralized hybrid multicloud repository that can be accessed by swarm cognitive robots. These robots could reuse/share their knowledge and adapt their individual behavior in real time according to the context in which they operate.

Chapter Review

Key Terms

3-D printing (also: *additive manufacturing*) process of designing static objects in three dimensions through additive processes in which successive layers of material are laid down under computer control

4-D printing provides the capability of programming the fundamental materials used in 3-D printing by creating objects that can change their form or function after fabrication

accessibility addresses discriminatory aspects related to equivalent user experience for people with disabilities

artificial human machine with biological parts added

augmented reality (AR) interactive experience that supplements the real world with virtual (computer-generated) objects that appear to coexist in the same space as the real world

base container image foundational layer of a container provided by the cloud provider to build an application

biomimetic robotics focused on the design of robots that leverage principles that are common in nature such as what can be learned from the evolution and development of intelligence in animals and humans

block storage manages data as blocks or physical range of storage in a physical device such as a hard disk drive (HDD) or Non-Volatile Memory Express (NVMe)

blockchain network peer-to-peer network that allows people and organizations who may not know one another to trust and independently verify the records of financial and other transactions

bring your own cloud (BYOC) solution structure in which organizations allow employees or users to freely decide on the cloud vendor that best suits their tasks rather than standardizing a single specific provider

cognitive robotics field of creating robots that can think, perceive, learn, remember, reason, and interact

compute service infrastructure component that enables users to obtain access to a private computing environment

container management services way to encapsulate an application with any operating system's library that is required for an application to operate

container registry (CR) registry where users can control each container image they deploy into the cloud environment

cyborg biological human with parts replaced with machinery

deep machine learning type of machine learning that has many neuron layers

extended reality (XR; also: mixed reality [MR]) reality service technology that involves both real and virtual environments

file storage storage service that manages data as files

inclusion ensures that diverse communities can make use of solutions regardless of their location, culture, and other differentiating traits, habits, or interests

intelligent autonomous networked supersystems (IANS) intelligent chains of autonomous machines that work together as a system to make decisions and take actions

logging and monitoring management common web and mobile cloud component that allows developers to centralize all logging data and provide a comprehensive view of all events happening with an application at any moment

mashup web-based application that combines features from two or more sources to present a new service

mixed reality (MR; also: extended reality [XR]) reality service technology that involves both real and virtual environments

mobile robot robot that can navigate from one position to another

multicloud solution mesh of several different computing environments to form a flexible working environment

nanotechnology focuses on matter at the molecular level to create structures and devices that are about 1 to 100 nm in size with fundamentally new organization, properties, and performance

neuroinformatics study of how to build a computer that can mimic basic human brain functions
neuromorphic computing relies on a hardware architecture that models how the human brain uses neurons
NoSQL database database management service in which the relationship between data is not strictly managed, often under key/value pair
object storage (also: *blob storage*) manages data as blobs, with each blob representing any data format
quantum computer machine that performs calculations based on the laws and principles of quantum mechanics
relational database service (RDS) database management service in which the relationship between data is strictly managed, often under a predefined data schema; most common type of database and includes Oracle DB, MySQL, and PostgreSQL
robot manipulator physical tool that operates at a fixed location to catch and move items
robotics science that focuses on the design, development, operation, and use of robots along with the computers that facilitate their control, monitor their sensing abilities, and process related information
secret and configuration management stores and manages sensitive information securely, while allowing the application to be scaled and deployed on different environments on the cloud.
shallow machine learning type of machine learning that has few layers of neurons
software development kit (SDK) used to access storage needed to read, write, and store data
spot/not urgent compute service enables the user to get a task done, but keeps costs low by allowing the cloud provider to run the task without urgency when the time is convenient and cost-effective
storage access point critical component in cloud architecture that ensures latency to enable users to consume data
storage service base infrastructure components that a cloud provider would provide that allow the user and the application to read, write, and access storage
telecommand category of IoT network traffic that sends commands across a network to control devices or sensors
telemetry category of IoT network traffic that aggregates data generated by sensors and devices and sends it to a server
virtual compute service enables the user to request an environment to do tasks and then shut it down to release the resource back to the cloud provider
virtual functional and serverless compute service application that runs in the compute environment, is executed as the function, and is then shut down when the task is completed
virtual reality (VR) simulated experience that immerses a user in a computer-generated, interactive, 3-D environment

Summary

13.1 Hybrid Multicloud Solutions and Cloud Mashups

- Hybrid cloud solutions provide a way for businesses to transform their infrastructure without putting all their resources and data into the cloud by allowing the coexistence and management between on-premises systems and cloud services. This solution provides a way for the business to keep some infrastructure and data on premise and enjoy the scalability that cloud computing provides.
- Multicloud solutions allow businesses to connect and explore with multiple cloud vendors at once. This solution provides flexibility for the business to select the best service on every cloud provider and only use the most cost-effective component in their infrastructure.
- Cloud mashup web-based application architecture allows a business to use different components coming from the multiple cloud providers in their application.
- Future cloud applications aim to deploy and manage applications at the mobile and network edge to maximize performance and reduce latency. This advancement will be a big stepping stone to enhancing user experience for IoT and other real-time data-collecting services.

13.2 Big Cloud IaaS Mainstream Capabilities

- Infrastructure as a service (IaaS) provides a pay-as-you-go pricing model and allows users to take advantage of cloud servers by virtually managing data and servers for their applications.
- Storage service provides elastic storage services that allow users and applications to read, write, and access as needed. Common components that may require storage access are analytical data, logging information, application data, images, and videos. Three common types of storage services that all cloud providers provide include file, object, and block storage.
- Compute services provide users with access to a private computing environment to develop or run tasks that they cannot run in their local environment. Common compute services include virtual compute service, spot/not urgent compute service, and virtual functional and serverless compute service.
- Content delivery network (CDN) is the network of servers and networking infrastructure across the globe that allows fast access from any place in the world into the network. Two common web and mobile cloud components are secret and configuration management and logging and monitoring management.
- Container technology encapsulates an application with any operating system library required for it to operate. With containers, developers are not concerned about environmental mismatches between the environment where the application was developed and the environment where the application runs.
- Kubernetes (K8S) service is the most popular container orchestration system. The K8S environment is one of the key systems for running a hybrid cloud environment, where users can run applications from both their local and cloud environments.
- Data management systems are the heart of applications. However, managing a database is usually difficult with oversized and complex data. To address this, most cloud providers provide several managed database services.

13.3 Big Cloud PaaS Mainstream Capabilities

- Platform as a service (PaaS) is an option for organizations that need the features in IaaS plus a platform to develop, test, and launch applications.
- IoT network traffic falls broadly into two categories—telemetry and telecommand. Telemetry aggregates data generated by sensors and devices and sends them to a server. Telecommand sends commands across a network to control devices or sensors. To serve the purpose of IoT, several application-layer protocols have been developed, such as MQTT, AMQP, CoAP, XMPP, and STOMP. This was necessary because application layer protocols, such as HTTP, are not suitable for IoT telemetry and telecommand applications.
- IoT services provided by big cloud vendors include both shallow machine learning, which has few layers of neurons, as well as deep machine learning, which has many neuron layers. Big data analytics is important to help organizations with decision-making processes. Big data analytics requires shallow machine learning services. Tools for big data analytics include Hadoop or Spark stacks, which are machine learning libraries with algorithms and functions to develop machine learning models and machine learning tools.
- For deep learning, cloud vendors provide services in the form of programming frameworks that help implement deep learning applications using differentiable programming.
- Blockchains use a distributed ledger system to store data and transactions in an open-source database. With blockchain 2.0, developers have a mechanism that allows programmable transactions, which are modified by a condition or set of conditions. Blockchain uses open-source frameworks that enable you to build applications that allow multiple parties to securely and transparently run transactions and share data without using a trusted central authority.
- Virtual reality (VR) enables a computer-generated, interactive, 3-D environment in which a user is immersed. Augmented reality (AR) supplements the real world with virtual (computer-generated) objects that appear to coexist in the same space as the real world. The key distinction between VR and AR is that VR is meant to immerse the user in a virtual environment, while AR introduces virtual elements into the real world.
- Microsoft Azure offers various PaaS services relating the extended reality applications, including Azure

storage services that may be used to store 3-D models, Azure Remote Rendering (ARR), Azure Object Anchors (AOA), Azure Spatial Anchors (ASA), Azure (cognitive) Speech Service, and Azure (cognitive) Vision Service. Google and other vendors, such as Amazon, also offer such services.

- 3-D printing, formally known as additive manufacturing, is used to design static objects in three dimensions through additive processes in which successive layers of material are laid down under computer control. Various cloud vendors are making 3-D printing technology available on the cloud today. 4-D printing adds the capability of programming the fundamental materials used in 3-D printing. It creates objects with dynamics and performance capabilities that can change their form or function after fabrication. These objects can be assembled, disassembled, and then reassembled to form macroscale objects of desired shape and multifunctionality.
- Various application development support capabilities are provided as PaaS services on the big clouds. These services may improve the management and operation of cloud applications. Services provided include integration management, identity and security management, application life cycle management, monitoring, and management and governance.

13.4 Towards Intelligent Autonomous Networked Super Systems

- Intelligent autonomous networked supersystems (IANS) of highly connected AIs are becoming the next major development for chained computing, where intelligent chains of autonomous machines work together as a system to make decisions and take actions.
- The metaverse is persistent, with its collective network of 3-D-rendered virtual elements and spaces available throughout the world, and it is shared, giving a vast number of users simultaneous access.
- With unlimited possible applications to promote social good, 3-D Mixed reality technology includes technology such as one that alerts people who are blind when rapidly moving objects are headed in their direction.
- Usability, accessibility, and inclusion, along with human-computer interaction (HCI) are important issues as standards and guidelines are updated to support the metaverse.
- Supersociety applications replace humans in certain mechanical activities and allow them to focus on activities that machines are not able to perform on their behalf.
- Neuroinformatics strives to build a computer that can mimic brain functions, handling ambiguity as well as uncertainty.
- Cognitive robotics aims to replace humans in dangerous environments and manufacturing processes or resemble humans in cognition, enabling robots to do jobs that are hazardous to people.
- A quantum computer can perform calculations by following quantum mechanics laws.
- Intelligent autonomous networked systems (IANS) are positioned to leverage supersociety capabilities such as nanotechnology, robotics, and supercomputing, but limitations on artificial generalized intelligence have made practical applications difficult.

Review Questions

1. What solution do organizations usually select when they need to store sensitive information and/or function as central repositories for data syncing?
   a. public cloud
   b. hybrid cloud
   c. private cloud
   d. mashup cloud

2. Which cloud resources are provided by third parties and are fully managed services with multiple tenants?
   a. public cloud
   b. hybrid cloud
   c. private cloud
   d. mashup cloud

3. Usually, what means do organizations use to communicate in the hybrid cloud network?
   a. cold data backups
   b. application programming interfaces (APIs)
   c. on-premises systems
   d. centralized control tools

4. Which cloud solution allows developers to pick and combine different information from public sources and create their own application?
   a. public cloud
   b. hybrid cloud
   c. multicloud
   d. mashup cloud

5. What cloud service allows an organization's employees or users to freely decide their own cloud vendor that best suits their tasks rather than standardizing a single specific provider?
   a. bring your own cloud solution (BYOCS)
   b. hybrid cloud
   c. private cloud
   d. public cloud

6. Which storage service manages data as blobs?
   a. file storage
   b. object storage
   c. application storage
   d. block storage

7. Which infrastructure component of cloud computing provides users with the ability to gain access to a private computing environment?
   a. compute service
   b. file storage
   c. software development
   d. storage service

8. Which cloud computing component accelerates the web and mobile workload globally?
   a. storage service
   b. compute service
   c. content delivery network
   d. application storage

9. Which innovation in cloud computing addressed the issue of an environmental mismatch between the application development environment and the environment where an application runs?
   a. object storage
   b. container management services
   c. content delivery network
   d. compute service

10. What service allows data to be managed under key/value pair?
    a. content delivery network
    b. relational database service
    c. container management services
    d. NoSQL database service

11. What is telecommand?
    a. a feature of 5G that creates higher radio frequencies, enabling data transfers at faster speeds

b. a category of IoT network traffic that sends commands across a network to control devices or sensors
c. a feature of 5G that enables multiple IoT devices to be used simultaneously within the same geographic area
d. a category of IoT network traffic that aggregates data generated by sensors and devices and sends it to a server

12. What type of machine learning does big data analytics require?
a. shallow machine learning
b. telecommand
c. telemetry
d. deep machine learning

13. What feature ensures that smart contracts are secured against tampering and unauthorized revisions?
a. deep machine learning
b. IoT
c. ML toolkits
d. cryptography

14. What type of reality immerses a user in a computer-generated, interactive, 3-D environment?
a. augmented reality (AR)
b. virtual reality (VR)
c. extended reality (XR)
d. mixed reality (MR)

15. How does 4-D printing enhance 3-D printing?
a. 4-D printing adds deep machine learning onto 3-D printing's shallow machine learning.
b. 4-D printing adds three-dimensional capabilities to 3-D printing's two-dimensional properties.
c. 4-D printing adds the capability of programming the fundamental materials used in 3-D printing.
d. 4-D printing adds the capability of using both telemetry and telecommand to enhance 3-D printing.

16. Which PaaS service can support organizations with project management needs?
a. integration management
b. application life cycle management
c. monitoring
d. management and governance

17. What technology is best positioned to replace today's web technology?
a. quantum computing
b. extended reality (XR)
c. blockchain 4.0
d. the metaverse

18. Which technology can produce holographic overlaying of images and data onto real-life contexts?
a. neuroinformatics
b. extended reality (XR)
c. quantum computing
d. virtual reality (VR)

19. What concept focuses on designing products to be effective, efficient, and satisfying?
a. inclusion
b. cognitive computing
c. usability
d. accessibility

20. Which type of robotics uses principles discovered in nature as the approach to design robots?
    a. cognitive robotics
    b. software robotics
    c. biomimetic robotics
    d. manipulative robotics

21. Which field is striving to build a computer that can mimic basic human brain functions?
    a. neuroinformatics
    b. biomimetic robotics
    c. robotic manipulators
    d. nanotechnology

22. How does the environment for automated networks differ from the environment for autonomous networks?
    a. It leverages AGI.
    b. It tends to be predictable.
    c. It leverages edge computing.
    d. It tends to be unpredictable.

Conceptual Questions

1. Explain the difference between public and private clouds.
2. How do organizations make decisions about cloud deployment?
3. What should developers consider when they design API systems that allow easy communication between systems while also maximizing opportunity costs?
4. How does a multicloud solution differ from a hybrid cloud solution?
5. What are some of the disadvantages of cloud mashups?
6. What does it mean to operate workloads at the mobile or network edge?
7. Explain how file storage, object storage, and block storage differ.
8. How does spot/not urgent compute service differ from virtual compute service?
9. Explain how the content delivery network functions.
10. What is Kubernetes (K8S) service and how does it benefit users?
11. Describe a relational database service and give at least two examples of this type of database.
12. Explain the difference between shallow machine learning and deep machine learning.
13. What is blockchain 2.0 capable of?
14. Explain the differences among virtual reality (VR), augmented reality (AR), and extended reality (XR).
15. What can be done with 4-D printing that is not possible with 3-D printing?
16. Name at least two application development support capabilities provided as PaaS services, and explain why these services are important.
17. What two features characterize the metaverse?
18. Provide at least two examples of how XR technology is being applied in health care.
19. Under human-computer interaction guidelines, a usable system must understand the various roles of humans. What are those roles?

20. What is nanotechnology and how can it benefit people who use the Web?
21. What is neuromorphic computing?

Practice Exercises

1. You have been tasked with developing the cloud architecture for your organization. You need to deploy and manage your organization's applications for thousands of concurrent customers, as well as store data that are sensitive. What type of cloud architecture will meet your organization's needs?
2. As your organization grows, you need to update the cloud architecture. The organization needs a flexible working environment that offers competitive pricing and robust security, as well as automation and scalability. What type of cloud architecture will meet your organization's needs at this point?
3. Your organization recently experienced a system failure that resulted in downtime and cost the organization more than a million dollars. To prevent this situation from occurring again, you have recommended that the organization implement a multicloud solution. Why are you recommending this?
4. To convince your organization to implement a multicloud solution, you explain that one advantage is robust security. Why is robust security an advantage of multicloud solutions?
5. You need storage services that provide quick, direct access to files and give you the ability to track and manage how your data are changed. Which type of storage service will best meet your needs and how does this storage service manage data?
6. If you have a limited budget and more flexibility in the timing of your tasks, what type of compute service are you likely to use and why?
7. What is one of the most critical issues that organizations resolve by using web and mobile application services?
8. Your colleague wants to build an application. You suggest using a container management service. What are the next steps to do this?
9. Your colleague wants to implement a cloud solution but is concerned because their business handles a large database with data that must be written and captured quickly. What advice do you offer them about cloud solutions?
10. Turn your smartphone into an IoT device by implementing and documenting a solution based on one of the following tutorials:
    - Azure IoT Samples for iOS Platform (https://openstax.org/r/76AzureSamples)
    - Turn Your Smartphone into an IoT Device (https://openstax.org/r/76IoTPhone)
11. Unity Barracuda is a lightweight cross-platform neural networks inference library for Unity. Barracuda can run neural networks both on GPU and CPU. Follow the instructions for installing Barracuda (https://openstax.org/r/76Barracuda), read the documentation (https://openstax.org/r/76Documentation) and the starter kit documentation (https://openstax.org/r/76StarterKit). Get the BlazeFaceBarracuda project (https://openstax.org/r/76BlazeFace) to run and deploy.
12. You have hired a developer to create a video game. Prepare a flowchart of what you expect the video game to do, noting which features should be VR, AR, and/or XR.
13. Perform research on the Internet and document applications of XR/the metaverse that are being applied in industries other than health care.
14. Perform research on the Internet and document the vendor solutions that are currently available to support the six key layers of the metaverse, namely: infrastructure, access/interface, virtualization tools,

virtual worlds, economic infrastructure, and experiences.

15. Research and document the feasibility of nanorobots.

Problem Set A

1. Your organization implemented a hybrid cloud network and is experiencing communication problems between its different platforms. As your organization's technological manager, how will you resolve this issue?
2. Your organization relies on a private, on-premises cloud to manage its applications and data. But as the organization grows, this system has become expensive to maintain and is no longer cost effective. The organization needs a cheaper option that also provides a flexible working environment with robust security. How will you resolve this issue?
3. Most of your organization's employees work remotely. As such, you suggest that the organization utilize BYOC (bring your own cloud) solutions. Why do you recommend this?
4. Follow this tutorial (https://openstax.org/r/76AmazonS3) to mount an Amazon S3 bucket as a drive on Linux, MacOS, or Windows (your choice) to make files stored on that drive shareable on the cloud.
5. Run the following notebooks by bringing up Jupyter in your own Linux data science virtual machine on Microsoft Azure or by running the following tutorial container:

   ```
   docker run -it -p 8888:8888 dbgannon/tutorial
   ```

   You may need to use a Spark cluster, and you may use Microsoft Azure HDInsight, if that is the case.

   - Get familiar with machine learning using Azure ML; execute and document the following exercise: Azure ML sample (https://openstax.org/r/76AxureML), which uses Azure ML to build a document classifier as a web service. Download the notebook file (https://openstax.org/r/76Notebook).
   - Get familiar with deep learning using TensorFlow; execute and document the following exercise: tensorflow (https://openstax.org/r/76Tensorflow), which illustrates leveraging TensorFlow to build a very simple logistic regression analyzer that can be used to make simple predictions of graduate school admissions. Download the notebook file (https://openstax.org/r/76NotebookFile).

6. Install ROS Melodic (https://openstax.org/r/76RosMedic) and its associated development environment and experiment with some of the tutorials (https://openstax.org/r/76RosMedTutor) provided.
7. Install the CoppeliaSim robotics simulator integrated development environment (https://openstax.org/r/76Coppelia) and experiment with some of the tutorials provided to illustrate its use in combination with ROS.

Problem Set B

1. You own a small business and have decided to expand your operations. To support the expansion, you need a cloud solution that will allow you to keep your customers' data private, while also providing them with access to multiple applications. Because your budget is limited, your cloud solution must have a competitive price, and you want the option of having a backup vendor to ensure your system doesn't experience downtime. Describe the cloud solution that you will implement and explain why you selected this option.
2. As you expand your business, you want to offer employees more flexibility in how they structure their work processes. You hope this will make employment with your business more attractive to highly qualified job seekers. Explain how you will use BYOC solutions to make your business' approach to work more attractive and entice employees to work for you.

3. You are the news director for your university's website, and you want to provide students with university news as well as weather updates and news from local and national sources. How can a cloud mashup help you accomplish this?
4. Mount a Microsoft Azure file share using this tutorial for Linux (https://openstax.org/r/76Linux), this tutorial for MacOS (https://openstax.org/r/76MacOS), or this tutorial for Windows (https://openstax.org/r/76Windows) (your choice).
5. Navigate to the various big cloud portals (i.e., Amazon AWS, Google GCP, IBM Cloud, and Microsoft Azure) and create basic VMs on each one of these clouds. Delete the VM instances after you create them unless you plan to use them again in the near future (in which case make sure to stop them to limit the consumption of cloud resources). Follow this tutorial (https://openstax.org/r/76BasicVMs) to build and deploy a simple web application on the Microsoft Azure big cloud.
6. You will need to work with this tutorial (https://openstax.org/r/76Medium) for this problem.

   The application is a casino-like solution where users are able to bet money for a number between 1 and 10 and if they're correct, they win a portion of all the ether money staked after 100 total bets.

   A. Follow the tutorial to create and deploy the DApp in Ethereum (off-cloud for now).
   B. Review and summarize the blockchain offerings and tutorials available on the various big clouds at the following locations:
      - blockchain on Microsoft Azure Cloud (https://openstax.org/r/76AzureCloud) (note: Microsoft is now working with Consensys as the main technology partner for blockchain cloud services); create a free account (https://openstax.org/r/76FreeAccount) and refer to the tutorials (https://openstax.org/r/76Consensys)
      - blockchain on AWS (https://openstax.org/r/76BlockchainAWS) and the tutorial (https://openstax.org/r/76AWSTutor)
      - blockchain on IBM Cloud Platform (https://openstax.org/r/76IBMCloud) and the tutorials (https://openstax.org/r/76IBMTutor)
      - blockchain on Oracle Cloud (https://openstax.org/r/76Oracle) and the tutorials (https://openstax.org/r/76OracleTutor)
      - blockchain on Google Cloud (https://openstax.org/r/76GoogleBlockCh)

   Implement/port the tutorial application created in part A so it operates one of the big cloud platforms just listed.

7. Install scikit-learn (https://openstax.org/r/76SciKit) and explore machine learning. Name at least three ways that machine learning can help you study and pass this class.
8. Follow the ROS Tutorial (https://openstax.org/r/76ROSTutor) and the CoppeliaSim's additional tutorials (https://openstax.org/r/76CoppSimTutor) in their comprehensive user manual.
9. Follow these instructions (https://openstax.org/r/76FrankaSim) to experiment with the Franka simulator.

Thought Provokers

1. In the beginning of this chapter, we noted that as TechWorks' organizers made decisions about their system architecture, they asked these questions: How will they manage their budget so that the application can be effectively deployed and maintained? What if the application has a lot of customer data to traffic daily and it is expected to grow rapidly? Is deploying a single cloud or local environment infrastructure enough to execute now and in the future? How will they respond to data breach and systems maintenance? Ultimately, TechWorks decided to choose the best multicloud architecture that allows them to divide their responsibilities onto different cloud systems: application communications, customer services, access control, and resource management. Do you think TechWorks made the best

choice for their cloud solution? If you were advising TechWorks, what would you do differently and why?

2. Describe how TechWorks could use big cloud IaaS services to create products that would generate business. Give some precise examples and explain how the start-up would be able to scale client businesses.
3. Describe how TechWorks could use big cloud PaaS services to create products that would generate business. Give some precise examples and explain how the start-up would be able to scale businesses.
4. Assume you teach carpentry to students attending a vocational/technical school. How might you incorporate XR applications into your teaching tools? List at least three different ways you will use XR apps to help your students learn carpentry skills.
5. You are designing robots using biomimetic robotics. The robots will be used to assist individuals with mobility issues as they perform gardening tasks, such as watering and weeding flowers. Be creative and discuss two animals whose habits you will use as guidance as you develop the principles of design for this robot gardener.
6. To develop fully functioning supersocieties, we need a better understanding of what drives human reasoning and planning. Provide at least three suggestions for how we can gain this understanding and how we can apply this understanding to artificial intelligence.

Labs

1. Implement this tutorial on hybrid architecture design (https://openstax.org/r/76ArchitDesign) to deploy a hybrid app that spans both Azure and Azure Stack Hub and uses a single on-premises data source.
2. Build the "GanGogh" application to create art using GANs following the instructions (https://openstax.org/r/76GanGogh). Note that the code is heavily inspired and built off of the improved Wasserstein GAN training code (https://openstax.org/r/76Wasserstein).
   You may also adapt the application to create other images (or other types of data files) based on your own training data.
3. Explore these quantum computing tutorials (https://openstax.org/r/76Quantum) learn about the uses and applications for quantum computing.



14

Cyber Resources Qualities and Cyber Computing Governance

Figure 14.1 As technology has advanced, the digital platforms that evolved make it challenging to optimize user experience while protecting society and the planet. (credit: modification of "140218-N-XX999-002" by Shawn Miller/U.S. Army Engineer Research and Development Center, Public Domain)

Chapter Outline

14.1 Cyber Resources Management Frameworks
14.2 Cybersecurity Deep Dive
14.3 Governing the Use of Cyber Resources

Introduction

The broadscale adoption of cyber resources faces challenges in ensuring underlying solutions' qualities (e.g., security, performance, reliability) and responsible use (e.g., focus on sustainability, ethics, and professionalism) by current and upcoming generations. TechWorks is a fictional startup company fully committed to leveraging innovative technologies as part of its repeatable business model and as a business growth facilitator. The company faces the various challenges discussed in this chapter. In particular, TechWorks' ability to successfully define, measure, and enforce cyber resource quality requirements and cyber computing governance mechanisms has a profound effect on the company's operational excellence and provides a competitive edge for the innovative solutions that TechWorks develops.

To demonstrate the aforementioned challenges and how to address them, this chapter first describes the approaches used to define and measure the quality of cyber resources. It then focuses on the evolving aspects of cybersecurity assurance, which protects IT solutions against undesirable use. Lastly, this chapter delves into global governance mechanisms (e.g., responsible computing) that regulate the use of cyber resources to protect our current and future generations and the planet.

14.1 Cyber Resources Management Frameworks

Learning Objectives

By the end of this section, you will be able to:

- Describe cyber resources quality requirements
- Analyze various frameworks developed over time to manage cyber resources
- Identify specific cyber resources qualities such as cybersecurity
- Explain the growing challenges that are faced by innovative web solutions
- Analyze cloud cyber resource implementation strategies
- Describe the metaverse ecosystem and its cyber resource challenges

The "cyber" qualifier typically applies to anything related to computers or information technology. Cyber resources include cyber platforms, solutions, processes, policies, and procedures or guidelines required. Cyber resources store, process, and manage data and other assets electronically and make them available via networks. Cyber resources must be evaluated in a world heavily inundated with computers to ensure they conform to best practices and meet certain quality requirements. The qualities expected from cyber resources include security, safety, performance, usability, reliability, and autonomy. These qualities apply to the various architectural components of cyber resources, including business, application, data (and the pyramid of knowledge layers above data, including information, and knowledge), and infrastructure components. Cyber resources' qualities are typically paired with one another and enhanced to optimize user experience. For example, technology solutions that leverage front-end interaction must be usable and ensure safety in various forms (e.g., data protection and user safety).

While cybersecurity is meant to ensure the quality of an organization's cyber resources, it also calls for a specific cyber resource that encompasses the policies, procedures, technology, and other tools, including people, that organizations rely on to protect their computer systems and technological environments from digital threats. Regarding cybersecurity, organizations typically implement an information security policy (ISP) that outlines security practices for employees and systems. Since organizations' assets are not always information-centric, the ISP policy applies to all organizational architectural components, including its business, applications, data (and associated pyramid of knowledge layers), and infrastructure. For example, the ISP policy details how the various infrastructure components are evaluated and outlines how the organization's information is secured. An ISP policy typically includes safety procedures and best performance practices, but it does not handhold workers through their jobs by telling employees how to maintain safety and perform within their role.

To help manage the development of cyber resources architectures that abide by specific requirements, organizations typically choose a specific framework, or a **Technical Reference Model (TRM)**, that details the technologies and standards that should be used to develop systems and deliver services. Analyzing frameworks, which can be complete or incomplete, helps determine which TRM applies to an organization's needs. However, most TRMs and related frameworks briefly outline procedures or system structures/capabilities. It then becomes the responsibility of the workers to determine how to best adapt the framework to their company's needs. In adopting these TRMs and frameworks, computer scientists need to take a proactive approach to enhance the overall quality of cyber resources, including cybersecurity.

In 2012, a reference model and associated framework referred to as The Open Group Architechture Framework (TOGAF) TRM was created to help users develop architectures based on their specific requirements. It champions a layered and componentized architecture that can be leveraged to develop specialized domain and industry architectures for infrastructure and business applications. It also provides a set of qualities that these specialized architectures and the applications that use them must fulfill.

LINK TO LEARNING

Read this TOGAF Series Guide (https://openstax.org/r/76togaf) on the TOGAF Reference Model from The Open Group for more information about TOGAF, including a diagram that shows the Technical Reference Model in detail. The TOGAF Reference Model shows the service categories that may apply to an information system. Not all architectures will include all of these services. TOGAF users should select services based on their specific requirements to build an architecture.

This framework was created and designed to be customized for each organization, resulting in a more secure design framework. The TRM analyzes business information systems, including application software, platforms, and communication infrastructures. The TRM assumes that all software and applications are running on the Internet. The original intention of the model was to create a taxonomy of vocabulary and standards that could be used to evaluate business and infrastructure applications. The different qualities within the TRM were meant to break an organization's infrastructure into manageable segments. These segments could then be evaluated for cyber resource quality and implementation needs. This model tried to establish easy **interoperability**, allowing two or more computers or processes to work together for infrastructure applications while providing the versatility to analyze business applications. This TRM is vastly applicable to modern systems, but newer TRMs have been created to handle cloud and other infrastructure areas.

While optimizing the quality of cloud-based solutions, smart ecosystems, and supersociety solutions is proving to be quite difficult, these solutions are also opening the door to current and future challenges in computing including sustainability, ethics, and professionalism. This has led to the creation of the Responsible.Computing() movement as an IBM Academy of Technology (AoT) initiative in early 2020. Subsequently, in May 2022, the Object Management Group (OMG) created the Responsible.Computing() consortium with IBM and Dell as founding members. This led to the creation of the Responsible Computing Framework (Figure 14.2 and Figure 14.3).

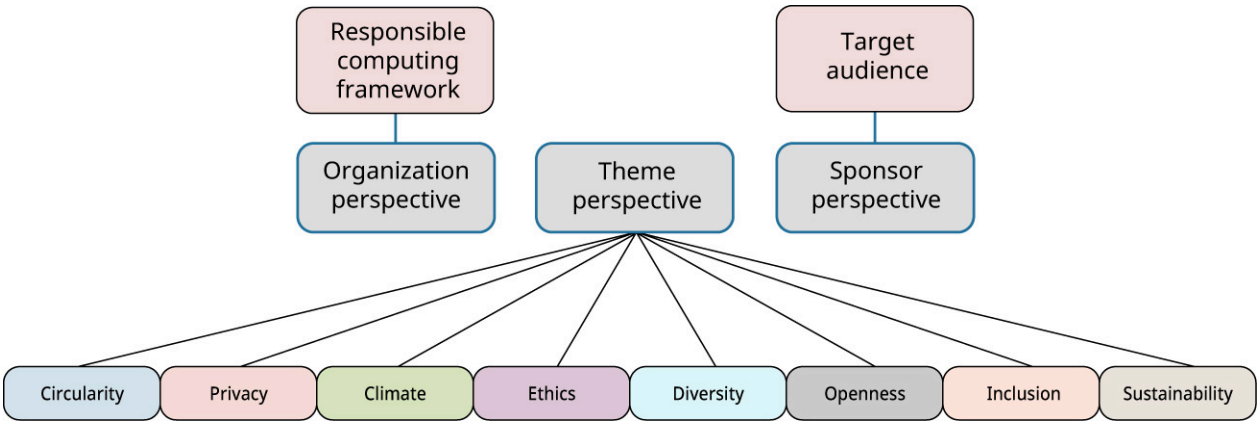


Figure 14.2 The various perspectives of the responsible computing approach include organization, theme, and sponsor. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

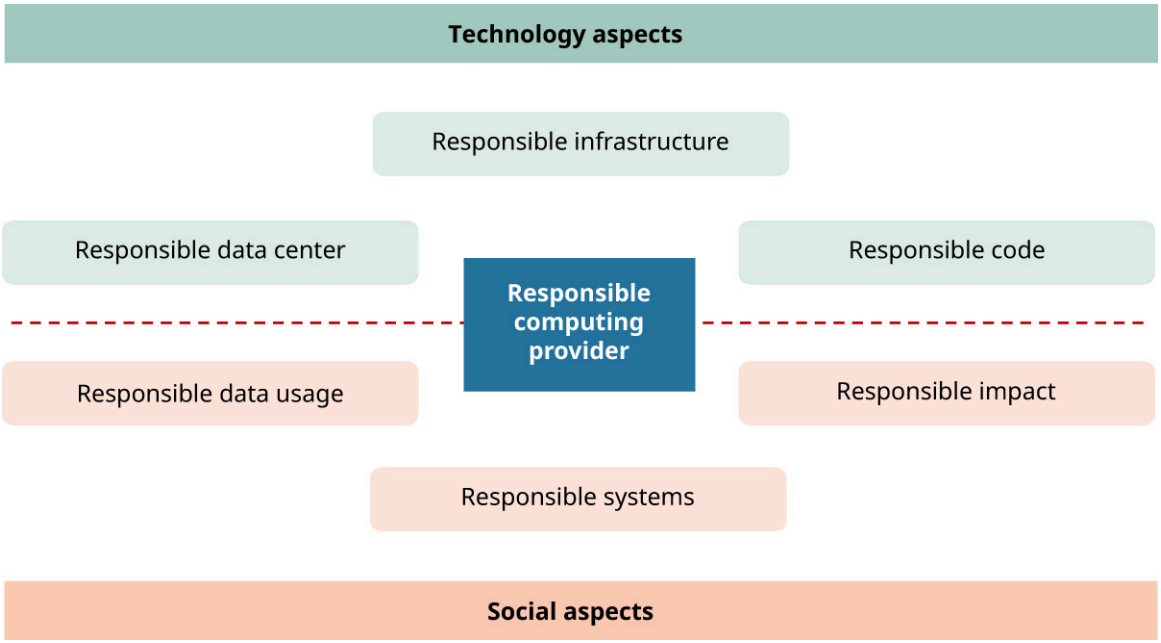


Figure 14.3 The Responsible Computing Framework helps organizations regulate the use of cyber resources to protect our current and future generations and the planet. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Responsible computing is a comprehensive approach tackling present and future computing challenges. It emphasizes both systematic design and soft skills crucial in the industry. Developers are urged to anticipate potential harm from their work, emphasizing secure coding and adversarial systems development.

These frameworks and styles target various enterprise business platforms. Traditional architectural styles like OMA and SOA are generic models that can be applied to the TOGAF. The OMA focuses on creating and assembling components that can be used for information systems, including interfaces. At the same time, the SOA relates to the assembly of services that can be applied to the TOGAF TRM. Understanding the difference between OMA and SOA is crucial, especially for cyber-quality resources. Newer platforms require updated models and the adoption of older models with modifications to fit system needs. This realization becomes more critical as cyber ecosystems continuously change.

These architectural styles offer vision but are not core concepts of cyber resources. The OMA reference model (OMA-RM) (Figure 14.4) was developed in the 1990s to detail communications with object request brokers and different services. It visualized interfaces and services before the need to control and protect web interfaces.

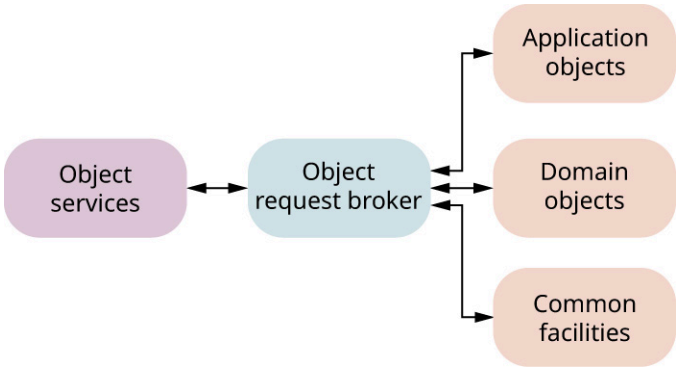


Figure 14.4 This figure shows the OMA reference model developed in the 1990s to explain communications and interactions with object request brokers and different services. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Different architectural styles cater to various project managerial styles, such as the waterfall model and the Agile development style (refer to Chapter 9 Software Engineering). Each framework has its strengths and weaknesses, influencing the implementation and evaluation of cyber resource qualities, which significantly

affects the information security policy. IT specialists must consider all cyber qualities, not just security. Best practices for quality and performance determine data storage methods, which can be automated to enhance cyber quality resources.

Cyber Resources Qualities

Cyber resources address various cyber platforms and their functional and non-functional requirements, including software and hardware solutions for information systems. Well-designed platforms balance the requirements while analyzing the effect on the environment in which the systems will function. In systems engineering, quality attributes are non-functional requirements, also called architecture characteristics, used to evaluate the performance of a system. As development practices evolve, different terms and nomenclature are created. For example, "**ilities**" refer to the "abilities" of architectural properties. They can include system architecture trade-offs, which may not be visible to the user, but that ensure the architecture will perform as needed.

Developers and enterprise architects are responsible for incorporating "ilities" into architecture. When developing information systems and quality management plans, the developer must analyze and create functional and non-functional requirements that are only measurable once a solution platform is built (refer to Chapter 9 Software Engineering and Chapter 10 Enterprise and Solution Architectures Management). This guides the creation of large-scale horizontal platforms such as the web/mobile and cloud platforms, ensuring acceptable quality and security.

Defining Cyber Resources Qualities

Cyber resources qualities are developed and measured within software models. The ISO/IEC 25010 standard, detailed in ISO/IEC 25002:2024,[1] details different software quality models and "ilities." This standard aims to provide guidelines for creating high-quality software and systems. A suite of ISO standards focuses on the quality of software and systems development, helping to identify quality characteristics that guide the development of functional and non-functional requirements.

Additionally, frameworks like OMA RM, OMAGuide (OMG), and TOGAF TRM guide these requirements by creating a taxonomy of service qualities. This expands the implementation of the different "ilities" a system can sustain. Functional specifications can fall into categories (Table 14.1).

Categories	Definitions
interoperability	The ability for two or more computer devices or processes to work together
composability	The ability to incorporate services within applications
scalability	The ability to enhance or retract system requirements for the number of users involved in the system
evolvability	The ability to adapt the system to new standards and practices
extensibility	The ability to modify the system to include new requirements or remove old requirements that are no longer needed
tailorability	The ability to customize the system for the needs of the users or industry

Table 14.1 Categories of "Ilities"

1 International Organization for Standardization and the International Electrotechnical Commission. *ISO/IEC 25002:2024.* https://www.iso.org/standard/78175.html

Categories	Definitions
security	The ability to ensure safe use of the system
reliability	The ability of the system to perform as needed (i.e., without failure) and to specification
adaptability	The ability to change or modify the current system to meet the needs of a different industry requirement
survivability	The ability to survive an attack or disruption of service within a system
affordability	The ability to create a system that is cost-efficient, not only monetarily but also with resource usage
maintainability	The ability for the system to be upkept
understandability	The ability of the system to be used (also referred to as the learning curve)
performance	The ability to work within measurable standards and requirements
quality of service	The ability to provide a solution to a problem or task in a way that most of the necessary user requirements are met
real-time	The ability to operate the system within normal operational time
nomadicity	The ability to work in a self-contained environment or the ability to move the system from location to location, when system location is a requirement

Table 14.1 Categories of "Ilities"

Defining system requirements into "ilities" makes it the developer's/architect's responsibility to ensure system security. This may seem daunting, but any system can be broken down into functional and non-functional requirements. A TRM supports and sustains the "ilities" while ensuring that the information security policy can be enforced. To do this, the TRM focuses on four service qualities: availability, assurance, usability, and adaptability (Table 14.2).

Qualities	Components of the Taxonomy
Availability	Locatability, manageability, performance, reliability, recoverability, serviceability
Assurance	Credibility, integrity, security
Usability	Ease-of-operation, international operations
Adaptability	Extensibility, interoperability, portability, scalability

Table 14.2 Service Qualities of the TRM

These quality areas and the categories of "ilities" provide an avenue and direction for cyber resources.

Measuring Cyber Resources Qualities

TOGAF recommends combining quantitative and qualitative methodologies to measure a system and provides several quality assessment techniques. The systems architect should create assessment plans to ensure all functional and non-functional requirements are met and tested, aligning with the framework. This methodology checks to see if the requirements are met and evaluates how well the requirements meet their objective. Other models, like the CIS Critical Controls,[2] can be used to test compliance with policies and requirements.

An important approach to testing resource qualities is to log tests performed and continually analyze system changes. This aids in assessing maintainability, survivability, and performance. The TOGAF Architecture Development Model (ADM)[3], centers the requirements around the development process.

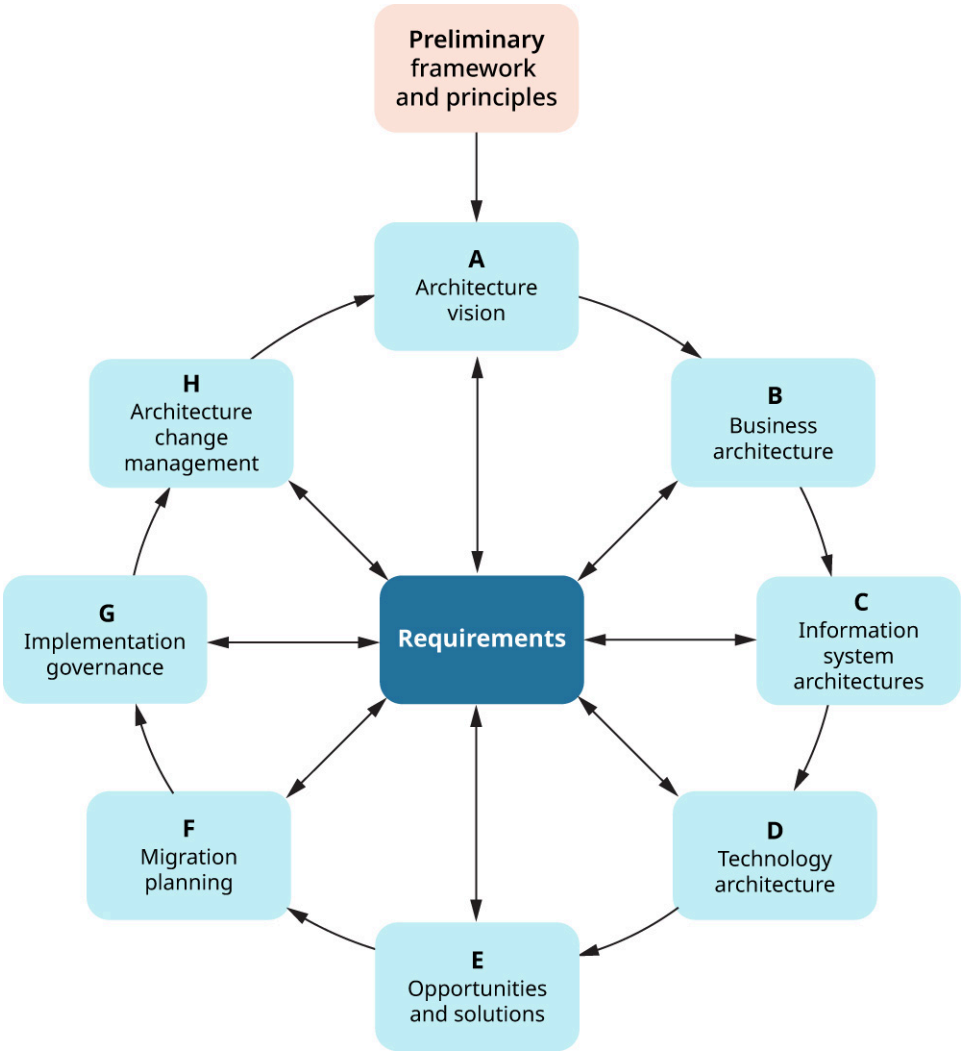


Figure 14.5 The TOGAF ADM is a standard to assist enterprises in identifying the steps of different activities and the inputs required to develop an architecture. (credit: modification of "TOGAF ADM" by Stephen Marley/NASA/SCI, Public Domain)

To assess the quality of a platform/solution architecture, TOGAF recommends using a combination of quantitative and qualitative methods. Quantitative methods measure performance against quality attribute scenarios, while qualitative methods review the architecture against best practices, principles, standards, and guidelines. TOGAF provides several tools and techniques for quality assessment, including the following:

2 Center for Internet Security, "CIS critical security controls," v8.1, 2024. https://www.cisecurity.org/controls
3 Refer to the TOGAF ADM in the TOGAF Standard at https://openstax.org/r/TOGAF

- architecture compliance review to check alignment with enterprise architecture standards and policies
- architecture maturity assessment to measure maturity level based on a predefined model or framework
- architecture capability assessment to measure an organization's capability to deliver and manage an architecture based on a predefined model or framework
- architecture trade-off analysis to compare and balance trade-offs between different quality attributes and design alternatives

To improve the quality of a platform/solution architecture, TOGAF recommends a continuous improvement cycle: plan, do, check, and act. In the plan step, define and agree on quality objectives, criteria, and metrics. In the do step, execute and document the quality activities, tasks, and responsibilities. The quality outcomes, results, and feedback are collected and analyzed in the check step. The quality issues, gaps, and opportunities are addressed and resolved in the act step.

To ensure the acceptance and adoption of a platform/solution architecture, TOGAF emphasizes quality communication. TOGAF provides various tools and techniques, such as architecture views, viewpoints, models, and documentation. Architecture views represent a subset of architecture, which focuses on a specific quality attribute or concern. Architecture viewpoints specify the conventions and rules for creating and using an architecture view, while architecture models are formal or informal descriptions of an aspect of the architecture. Lastly, architecture documentation collects artifacts that capture and convey information about architecture and decisions.

Cyber resource measurement is a layered process. Remember that according to IBM's responsible development methodologies, quality assurance should be done during all development steps, not just at the end. Following these steps results in an incremental system with proper checks and controls for security purposes.

Web and Mobile Solutions Era

Web, mobile, and IoT solutions are used at a global scale today. We have built development models, created a TRM, and educated computer scientists to prioritize cyber resource qualities. We have set the stage for web and mobile platforms by emphasizing these qualities in development. The TRM's focus on usability, ease of operation, assurance, security, and adaptability has built a framework for responsible content development. Utilizing web-centric server-side coding, we have created more secure platforms that efficiently communicate through APIs.

Scalability, extensibility, and flexibility of web platforms pose significant challenges for cyber quality. In the past, quality focused on ensuring that cyber solutions functioned as intended, but now it is also focused on security. Securing a web platform involves more than just the web front end; it also requires securing the server architecture, the physical locations of the servers, the cloud solutions, and interfaces. Many web platforms use third-party software plug-ins that rely on the proprietary software's security and quality. Poorly written code is easily exploitable, especially with the widely accessible tools to do so freely accessible on the World Wide Web (WWW), the Internet information system that connects public websites and users.

Web platforms have continuously changed in direction and architecture design. The architectural principles of the Internet stated in the Request for Comments (RFC) 1958 set forth by the Internet Engineering Task Force (IETF) in June 1996.[4] Knowing that the Internet had to be able to change was a design principle. Anticipating that IP addresses would need to adapt or be replaced at some point because the number of devices would exceed the allotted number was a guiding principle that forced the Internet to be scalable and extensible. Quality of service and assurance/security concerns drive the commercial Internet. Initially, the Internet's architecture was unregulated, and there was no mandate for IP address allocation or domain name services (DNS) from the Internet Architecture Board (IAB). Public Internet is now ubiquitous in most economic areas of the world. The increase in mobile platforms and the continuous need to be connected pushes the boundaries of cyber quality beyond the original architectural goals of the Internet. Performance and usability demands drive technology and the Internet to adapt and create new mediums easily adapted into scalable infrastructure.

Open Web Platform Quality Challenges

The Open Web Platform (OWP) initiative was designed to offer royalty-free (open) technologies for web platforms.[5] Its goal was to provide a foundation of services and capabilities for developing web/mobile applications, as illustrated in Figure 14.6. Using the OWP, everyone can implement a web or mobile solution without requiring approvals or license fees and maximizing usability and accessibility.

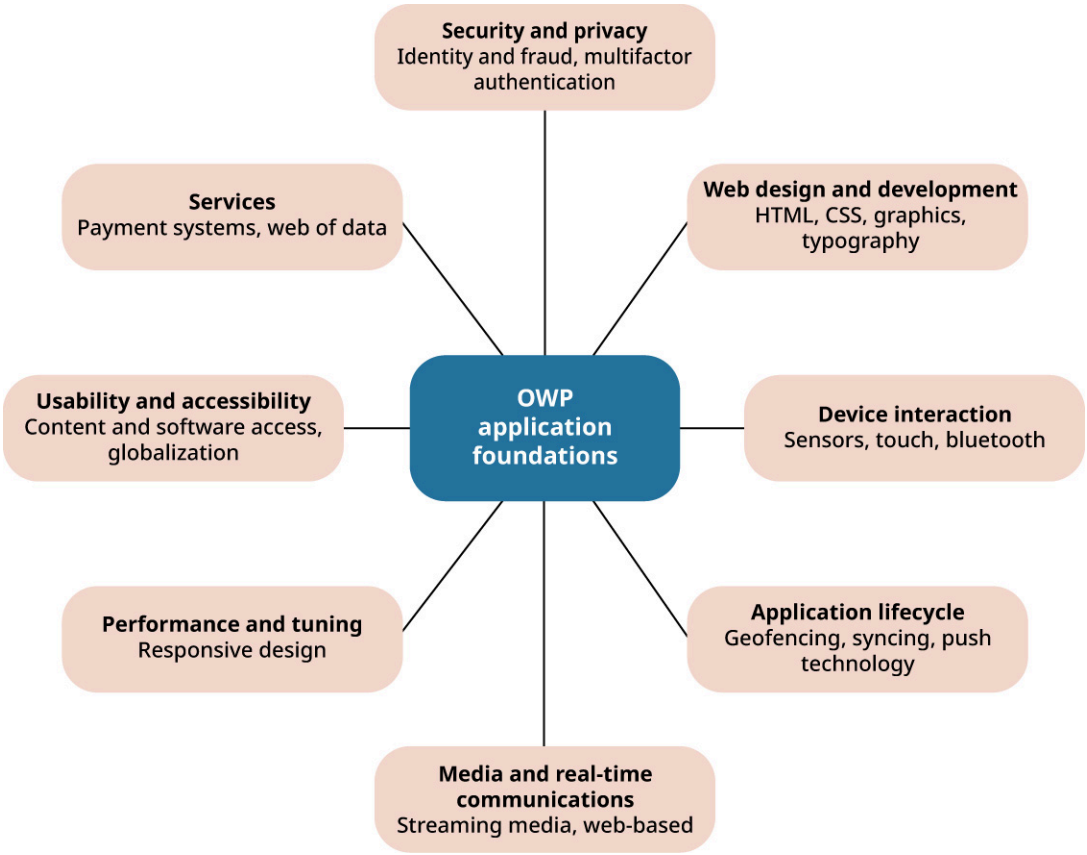


Figure 14.6 The foundational characteristics of OWP solutions allow anyone to implement a web or mobile solution without licensing. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The cyber qualities of these technologies are continuously evaluated. People want free web applications that are easy to use, have good performance, are secure (always an afterthought), are compatible with their current technology, and, most of all, remain free and open to the public. Security plays a significant role in the adoption, and ease of use is often a struggle. The most straightforward security measure, restricting access to

4 See Network Working Group, "Architectural principles of the Internet," June 1996. https://datatracker.ietf.org/doc/html/rfc1958
5 See W3C, "Open web platform," August 27, 2020. https://www.w3.org/wiki/Open_Web_Platform.

the system, is not feasible if companies want to interact with the world. Several companies, like Meta, use a limited functional interface called a **walled garden approach**, which limits openness and prevents users from accessing a platform. This approach goes against the spirit of OWP. Open web platforms like WordPress and Drupal provide user-friendly web design through drag-and-drop web page creation frameworks. However, updates to these technologies can break other areas or disrupt compatibility with other open web software packages. To mitigate this, people and companies are adopting the walled garden approach to web applications, limiting controls and plug-ins, and the developer can incorporate their own HTML security measures.

The wall garden approach goes against the principles of the OWP. The concept of "openness" wants technology to be free and interoperable. Giving individuals a subset of technologies and forcing them only to use those technologies diminishes the power and expansibility of the Internet.

Modern Web/Mobile Solutions Quality Challenges

The World Wide Web (WWW) demands 24/7 access as users expect constant availability. Companies like Amazon have set the bar for quality of service and how fast a product purchased on the Web can be delivered to a residency. Smartphones and the IoT have provided new access models to services and goods. What challenges does this pose?

- Account management: Users have numerous online accounts, requiring unique, strong passwords for each. Research by Colorlib in May 2023 found that the average person has 100 accounts needing passwords, up from 70–80 in 2022.[6] This growth highlights the challenge of managing diverse and increasing technology.
- Workforce shortage: There is a shortage of qualified professionals. Higher education and companies like Microsoft, CompTIA, and CISCO provide certifications to prepare workers, but students without industry experience struggle to find suitable jobs, and entry-level positions are limited.
- Threat landscape: The rapid growth of web platforms and technology increases security challenges. The use of cloud technology and IoT devices introduces new vulnerabilities. Companies like Microsoft invest heavily in training programs and certifications, particularly for securing Azure web platforms.

Modern Walled Garden Platforms Quality Challenges

The main challenge for Walled Garden Platforms is interoperability. For instance, WordPress offers more than 59,000 plug-ins. While these plug-ins allow for customization, only cosmetic changes are typically possible. This allows security to be maintained in a walled garden approach. Plug-ins are tested for security issues before being added to a library. However, using plug-ins in ways they were not intended can create vulnerabilities. Using plug-ins in untested ways goes against IBM's responsible coding practices. Then questions must be asked, such as who is responsible for the misuse? Who is responsible for updating the plug-ins for newfound security issues? And lastly, who is responsible for removing outdated plug-ins, whose functionality no longer works with the current technology?

Using frameworks to assess the proper development of websites and content should be a requirement for web programmers. Insecure content puts users at risk. Security frameworks, plans, and walled garden approaches can enhance web platform security. However, this requires skilled developers and available resources.

Cloud-Centric Solutions Era

Cloud platforms rely on applications built for web and mobile platforms, similar to the Open Systems Interconnection (OSI) networking model. In this analogy, cloud-centric solutions are layer 3, web programming, applications and the WWW are layer 2, and the Internet is layer 1. Each layer has its security challenges. Cloud-centric solutions involve many different systems, software, and communication methods. Applying the TRM to this model highlights availability and assurance as major concerns with adaptability being

6 Colorlib, "Password statistics (how many passwords does an average person have?)," June 9, 2024. https://colorlib.com/wp/password-statistics/

crucial with interoperability, scalability, portability, and extensibility. Usability is the biggest challenge. If users cannot use the product, then implementation will fail. Hardware issues present additional challenges beyond user interactions in the cloud-centric model. The OMG Cloud Working Group (CWG) addresses myths associated with cloud-centric Solutions, focusing on scope and applicability, costs, security, availability, and management. The CWG advises maintaining a calm approach to solving cloud issues and warns against "overselling" cloud solutions because of the difficulties in implementation needs.

Hardware Virtualization Quality Challenges

Many cloud-centric implementations use hardware or software virtualization. Hardware virtualization challenges face compatibility issues and constant updates, often resulting in performance loss when implementing a virtualized approach for cost savings. Vendors like Amazon, Google, and Microsoft offer hardware virtualization options that typically improve efficiency, flexibility, scalability, and security. However, a big concern with virtualized cloud solutions is that the availability of resources comes with extra processes. Users needing to access their data and information must navigate the complex interfaces that provide walled approaches to web design. This limits functionality, and since many organizations decide to host their cloud solutions themselves, it leads to new challenges with regard to cost, assurance, data privacy, and security.

When using virtualization solutions with the TRM, availability, and manageability are key concerns for users. Poor performance and outdated hardware can lead to liability for an organization. Redundant systems are necessary for maintenance and upgrades because there are times when a system is vulnerable and not available. This is where virtualized solutions can help. It is easy to spin up a third instance of a cloud-centric virtualization and perform the needed hardware upgrades in the first backup system and then the production system. While the backup system is down for hardware upgrades, the virtualized backup covers any necessary security issues. Once the servers are upgraded, an entire image is copied from the main server onto the backup server. Then, the backup server becomes the primary server with a virtualized backup solution, and the original main server can then be upgraded to become the new backup server.

During this hardware update, the implementation of the TRM becomes important. Information must be kept available throughout the whole process. If the faulty hardware is left in the systems, then errors or crashes may occur. Then, known vulnerabilities are identified in hardware configurations, and the process of hardware upgrades must be implemented.

Virtualized solutions require constant monitoring. Network modeling virtual local area networks must always be monitored for load balancing. A **virtual local area network (VLAN)** is a tool to connect devices and nodes from various LANs. Storage calculations and estimations must be constantly performed to ensure upgrades keep ahead of any issues. Manageability and serviceability are constant challenges, including both hardware and human resources. Training workers in new software is costly and time-consuming, resulting in minimal opportunities to learn on the clock and possible security lapses. Security procedures are often ignored for the sake of time. As a result, many companies seek cloud-centric systems to avoid these issues.

Container Management Platforms Quality Challenges

A **container** is a lightweight package that bundles together applications to form a solution to specific problems. When applying the TRM to containers, usability and ease of operation are major challenges for web/mobile and cloud platforms. Containers simplify development, deployment, and management across complex environments, offering a quick solution for loading features without concerns about the operating system. With a container, the developer is limited to the operating system for which the package was developed. While this seems like a challenge, it provides security and adaptability. Containers are built for specific application purposes. They are often referred to as enterprise solution support containers. Kubernetes includes support for load balancing, rollout of patches, configuration management, and storage orchestration. Alternatives include Docker Swarm, Apache Mesos, and cloud solutions such as Azure containers.

Effective container management benefits from the TRM in several ways. The usability and ease of operation

and the quality of web/mobile platforms include several approaches that help the user and developer manage and build the system.

- Ease of setup: The management system provides a drag-and-drop system for scheduling, storage maintenance, and system monitoring.
- Enhanced administration: It simplifies IT management, allowing administrators to focus on other tasks.
- Automation: It automates any number of processes and does automatic load balancing.
- Continuous health checks: Reporting and logging are essential for container management and allow an admin to do their job efficiently.
- Change management: It keeps a detailed change log of the packages. This allows for troubleshooting issues arising from upgrades of older packages.

Using the TRM to maintain alignment of the container management to the "ilities" is a large concern. Every aspect of the TRM can be applied to the container management platforms. It is recommended that when the TRM is used to evaluate the management systems, the developer thinks about incorporating an outside resource like the CIS Critical Security Controls to help determine the level of compliance needed for the given control. Overall, container management can greatly assist an organization looking to speed up its development time and interoperability between systems.

Cloud Big Data Analytics PaaS Quality Challenges

In our development pyramid, where cloud-centric platforms are at the top (layer 3), cloud and big data analytics form an overarching umbrella. The process of analyzing that data to find correlations and trends that can be used for decision-making is called big data analytics. Incorporating the Platform as a Service (PaaS) aspect for big data analytics provides a cloud-based development environment that can solve many development problems and provides methods to analyze and enhance system performance.

A big challenge with data analytics from a TRM perspective is time. It takes time to properly analyze data collected to make informed business decisions. Big data only helps information management when data is converted into information on time. While cloud providers do their best to offer 24/7 customer service, issues arise when third parties and containers are introduced. The data may not be immediately available from the third-party vendors, or they may not want to share their data. Having clear agreements with third parties about what data will be shared and what data will not be available is important.

A second challenge with big data solutions is collecting the wrong data. For instance, firewall log files might record time stamps of external IP address access attempts, but if the firewall doesn't account for IPv4 to IPv6 conversions, some IP address data could be lost. If the firewall only handles IPv4 and the attacker uses IPv6, the log files may be incorrectly formatted, resulting in incorrect or incomplete data analysis.

A third challenge for data analysis and PaaS is bandwidth. Proper data analysis requires large amounts of computing power and hardware, slowing network communications during data transmission. Storage is another issue, as analyzing multiple days of data requires additional storage locations, often leading to cloud storage and virtualized solutions. Companies like Google generate revenue by doing big data analysis and selling their findings to companies for marketing strategies. Managing over 1.8 billion Google accounts with a "free" 10GB cloud storage drive necessitates virtualization and an extensive security plan.

A service-level agreement (SLA) must be established when customers enter into agreements with PaaS platforms. The SLA outlines the services that the platform provides. Companies can estimate growth and need for their organization using SLAs. However, SLAs are not covered in the TRM, and since they are not actual "ilities," they may fail to include actionable items. This means that the SLAs are "what if" actions. For example, if there is a data breach, this will happen, making the SLA reactive rather than proactive. The TRM aims to prevent issues before they arise. SLAs are criticized for lacking assurance and privacy protections, as data breaches can put user information at risk. This highlights the importance of trust, which, while not directly mentioned in the TRM, is addressed in many different "ilities" in implied ways.

Trust in any service-providing system has a narrow margin for error. A trust relationship is established when consumers entrust their information to a company. To bolster this trust, companies should have information security policies detailing data security practices. However, exact methods should not be disclosed to prevent giving hackers helpful information. The trust relationship aims to make the consumer feel secure, and therefore increase their ease-of-use with the system. This trust can be a valuable metric for measuring quality based on customer satisfaction.

Cloud IoT PaaS Quality Challenges

Earlier, we compared technology to a one-mile race, with IoT representing the last ten steps. Every technological advancement has led to the IoT, as consumers want more power and faster access to information. They are dissatisfied when companies fail to provide information quickly and efficiently. Cloud IoT, as a PaaS, provides more challenges than this small section of the book can cover. To start, IoT's largest challenge is scalability. The rapid development of wearable devices, smart devices, smart cars, appliances, and artificial intelligence suggests no end to this growth. Consequently, IoT scalability is increasingly challenging to manage.

If scalability is out of control, then it is typically the case that security is out of control. Most TRM "ilities" encounter challenges with the IoT. Having a quality security plan for every possible third-party IoT device is not feasible and it is sufficient to analyze the performance of the underlying Bluetooth and Wi-Fi technologies. Developing better and stronger ways to secure communications between devices would be an excellent first step. There will always be an issue with hackers capturing data transmitted wirelessly.

The next challenge for IoT as a PaaS is storage. Where is all the captured data stored? Apple devices are stored in their iCloud, but other companies face security issues with stored data. Consider the data that is left behind on a device that is no longer in use. Best practices for disposing of computer hard drives and other storage devices don't always apply to IoT devices and smartphones. We created regulations like the General Data Protection Regulation (GDPR) and the California Consumer Privacy Act (CCPA) to address concerns arising from IoT storage situations.

Interoperability is a major challenge with IoT devices, given the vast types of devices and communication methods. Securing biometric data is a constant concern, and hacking into home monitoring systems with IoT devices poses significant safety risks. Since most IoT devices rely on Wi-Fi, improving interoperability efforts and security in wireless communications is necessary. Cloud-centric services are another concern, as all IoT devices are part of cloud technology. Developing interfaces for IoT devices is also challenging due to the need to accommodate cultural and language differences.

Cloud Robotics PaaS Quality Challenges

Cloud robotics PaaS merges two challenging areas: cloud computing and robots. Robotics has faced development challenges, especially in natural language processing, to help robots perform actions and communicate with users. This is often seen as a futuristic world where robots handle daily operations, reducing the need for human labor. However, the high cost of robotics remains a barrier. AI continues to evolve, driving industries to use it to enhance production and services. We as humans must repurpose our skillsets to "train" the robots to do our work.

Cloud robotics is the next step in the evolution of standard robotics. Traditionally, robots perform pre-loaded algorithms, making decisions based on predetermined outcomes. However, with cloud robotics platforms, robots use big data analysis to make more informed decisions. Leveraging the power of cloud data and rapid processing speeds, they can approach "thinking" to analyze a problem or a task. This is not to be confused with automation. Traditional automation involves robots replacing human jobs or tasks. Automation is a programmed response with limited inputs and decisions. Cloud robotics PaaS, in contrast, allows for more extensive inputs and a broader range of decisions. This enhances the robots' decision-making capabilities through advanced algorithms.

The TRM has small applicability to robotics, with key considerations being security and connectivity. Robotic cameras and other devices need to transmit their data for processing, making connectivity to cloud platforms essential. Reliable high-speed data transmission is critical to cloud robotics, raising concerns about network outages. Robots might cease functioning without incoming data, and latency or corrupted data packets can impair operation.

Cloud robotics shares many challenges with simple networking, but the issue is that robotics may have a more important role than a computing platform. If the robots are not getting the needed inputs, they may be unable to carry out the functional tasks. For example, suppose a robotic drone traveling over a mountain range loses communication while searching for lost hikers over a mountain range. In that case, it might crash without its ability to transmit and receive data. The drone cannot receive any flying commands and would plummet to the ground. Safety protocols are essential, such as hovering in place until communication is reestablished or the battery depletes. While these connectivity issues are not new, the context has shifted. Previously, this scenario would have involved a helicopter or airplane doing the search, but it is more costly to have a plane fly to search for lost hikers than a drone. Connectivity remains a persistent challenge for any system relying on remote communications.

Industry 4.0 Metaverse Smart Ecosystems Era

The fourth industrial revolution, also referred to as Industry 4.0, is characterized by increasing automation and the employment of smart machines and smart factories; insight obtained from data helps produce goods more efficiently and productively across the value chain. The term “metaverse,” which was coined in Neal Stephenson’s 1992 novel “Snow Crash,” has been used by Meta, the company that developed Facebook, to refer to a platform that applies augmented and virtual reality in a social media environment. However, the metaverse concept extends beyond social media and refers to the next generation of the Web that will provide an immersive augmented/virtual reality (AR/VR) interface and will leverage new technologies including AI/ML, IoT, blockchain, 3-D models, and others. The metaverse will go beyond Web 3.0 that already includes blockchain 2.0 capabilities on a decentralized Internet.

The metaverse is a hybrid cloud environment aiming to uphold the OWP principles. Software developed for the metaverse must ensure full interoperability and interactivity with all other platform software. These requirements significantly constrain the technology's ease of use and performance requirements.

Incorporating virtual and augmented reality into the metaverse typically involves using an AR/VR headset to access a library of software programs, each offering a highly realistic user experience. To be part of the metaverse, software must adhere to these rules[7]:

1. There is only one metaverse, and all people should have access to it.
2. The metaverse exists beyond everyone’s control and must be accessible at all times.
3. The metaverse doesn’t care about hardware.

We must understand how metaverse solutions relate to the hybrid cloud environment and leverage the foundations set forth by OWP. New challenges for each one of the qualities arise as we go up in layers. For example, cloud security requires users to trust their cloud provider. The ecosystem resulting from the software developed to interact with the metaverse provides challenges for all cyber-quality services.

From a security perspective, users must know about privacy and data protection when using AR. Systems collect large amounts of data, and users need to understand what is being collected and where it is stored. For instance, self-driving vehicles use AR technology to navigate and store driving patterns and routes. If hacked, this data could reveal when a user is not home. Additionally, many self-driving cars store garage door information to automatically open doors when the vehicle is nearby, posing further security risks.

The ecosystem should provide a plug-and-play mentality for all metaverse-associated technology. Without this,

7 S. Subrahmanyam, “What is metaverse and how is it changing AR/VR world?” November 28, 2022. https://readwrite.com/what-is-metaverse-and-how-is-it-changing-ar-vr-world/

ease of use, performance, and reliability may suffer. Increasingly, people are integrating technology into their lifestyles, especially since the COVID-19 pandemic, with more online shopping and remote working. Virtual meeting rooms and advanced technology are replacing traditional office interactions. A work from home environment that is fully immersed in technology could be a way to provide life-like meetings where 3-D images and holograms of people sit at conference desks instead of flat-screen monitors. The technology performance, automation, and ease of use are at an all-time high. For this environment to become commonplace, trust in technology must increase. Companies must prioritize incorporating all cyber qualities to ensure high performance, automation, and ease of use.

A large issue is creating content that is both interoperable and secure. The demand for VR worlds and platforms drives rapid production, often leading to unchecked vulnerabilities due to insufficient time and resources. This is particularly problematic in VR, where large multiuser dimensions are common. Data exchanged between entities has to happen seamlessly, but the whole system is at risk if one system is compromised. Since there is only one metaverse, a security vulnerability in any software on the platform puts the entire system at risk.

A new risk that has emerged is addiction. Users may prefer the AR version of their environments, leading to a sensory deprivation effect in the real world that drives them toward AR and VR. This is particularly problematic for adolescents. Since the onset of the pandemic, increased AR/VR use has made younger users more susceptible to addictive behavior with these systems.[8]

Smart Ecosystems 3-D Modeling Platform Quality Challenges

Ecosystems that incorporate 3-D modeling are faced with many different challenges. To apply the TOGAF reference model to the problem, all of the qualities of concern are as follows:

- Graphics and images: Cloud computing has provided a storage medium for graphics and images, but the quality of the images based on their resolution is challenging. For AR/VR platforms, the images need to be of high resolution, which leads to more storage and lower transmission rates.
- Data management: The pure amount of data to create 3-D modeling is immense. The data collected for AR/VR to be successful requires more storage and memory within machines. The fastest Internet connections are needed for data transmissions, and archival procedures must be implemented for the data system's lifetime.
- Data interchange: As with data management, data interchange struggles with the transmission speeds of large 3-D models and the physical hardware found within the different AR/VR systems platforms. Since no complicated platforms are required, the data interchange must be universal.
- User interface: This is paired with graphics and images, but there is an aspect that is a different challenge. The ease of use is a challenge with 3-D modeling. The user interface must be seamless for the AR/VR to work properly. The user must be able to interact with objects appropriately rendered, and the user interface must be life-like.
- International operations: This area is a serious challenge with 3-D modeling, and it comes in the form of symbolism and color coding. In different countries and different cultures, symbols have different meanings. With 3-D modeling, where every country in the world is interacting with the metaverse, all software must be interoperable. Each country involved in the metaverse will have different data storage and security standards.
- Location and directory: Working with data management, location data, and directory information is always challenging. If two software development companies collaborate and create 3-D models that hold user data or geo-record, then storing that information is challenging for the system.
- Transaction processing: 3-D modeling requires a large amount of object interaction. When objects are developed for 3-D modeling, they are typically put into a setting or a landscape. Most objects interact with

8 A. H. Najmi, W. S. Alhalafawy, & M. Z. T. Zaki, "Developing a sustainable environment based on augmented reality to educate adolescents about the dangers of electronic gaming addiction," *Sustainability*, 15 no. 4, pp. 3185. January 19, 2023. doi: 10.3390/su15043185. [Online].

other objects, which is done through transaction processing.

- Security: The risks and challenges associated with security are some of the most difficult to identify and detect. Many companies are creating 3-D models for other entities to use. These models are not always verified, and penetration testing is not always performed on each 3-D model. If multiple 3-D models are incorporated into different companies' systems, then the security concerns escalate. Security seems to be an afterthought based on performance and system requirements.
- Software engineering: This area of the TOGAF deals with skills in different 3-D modeling software. Many HTML or scripting languages cannot do proper 3-D rendering; therefore, the software engineer needs special skills to develop the models correctly and with proper aesthetics for the platforms and worlds they will be used within.
- System and network management: The challenges are related to communications and interoperability. Since there is no requirement on what hardware or technology can be used to interface with the metaverse, network management is crucial. The communication needed to transfer 3-D models between AR/VR nodes is important, and the system resources will be challenged to perform these transfers properly. The metaverse is not housed on one computer and the bandwidth needed to transfer the large amounts of data needed for 3-D modeling will always be challenging.

The TOGAF and the TRM provide a good framework for analyzing most system challenges. Analyzing 3-D modeling is similar to analyzing any resources needed for the metaverse. For instance, besides 3-D modeling, there is a large language barrier in the metaverse, and each of the qualities of the TOGAF can also be used to analyze such barriers.

Smart Ecosystems AR/VR Platform Quality Challenges

Any ecosystem that houses AR/VR technologies will always struggle with ease-of-use issues. The TRM stresses that accessibility to any platform must be made easy. New technology introduced through the IoT presents challenges for AR/VR platform issues. The headsets needed for most AR/VR technology cannot be worn to walk around. They are usually meant to be worn safely away from obstacles the individual might encounter or trip over. This virtual platform may be easy to manipulate in the device, but the hardware is bulky and often clumsy. Wearing AR/VR gloves to interact with computer systems is a nice feature when the individual can use them fully.

Using AR/VR technologies to assist individuals who have disabilities is a major reason to invent the technology and push the boundaries of the technology.

Smart Ecosystems IoT Platform Quality Challenges

IoT platform challenges are equal to that of cloud technologies. The main challenges are bandwidth, storage, and infrastructure. There is a need for bandwidth that is more significant than 5G technology for all communications. The use of fiber optic communication greatly increases the rate of data transfer, but many times, the network is left to backhaul the network. Backhauling the network directs traffic to a longer out-of-the-way route to perform load balancing on the network, help prioritize important data, and send the data to the most direct route. Using backhauling for data that are not as important as primary communication is a reasonable workaround for having poor network bandwidth.

The infrastructure challenges are constantly being evaluated. The need to move all devices to the IPv6 protocol in the TCP/IP network layer away from IPv4 provides its challenges. The move to different IP settings is small compared to Internet provider (ISP) issues. Many ISP providers share mediums, but many others install their networks. These independent networks do not communicate; therefore, all information must go to large data centers, which can then be sent from there. This forces central offices and organizations to be formed to handle data warehousing. This introduces the challenge of central locations to IoT. There needs to be enough data centers close enough to each other to have data transmitted quickly and without any transmission issues. Network and mobile device security always needs to be addressed. The attack vectors created by the IoT

increase with every new device created.

The biggest issue with data storage is the physical need required for the data storage. Data warehousing and data mining have become their distinct area of the industry. The sheer amount of data collected becomes unimaginable. The average person in North America has over thirteen devices that have telecommunication needs.[9] It is reported that each household in the United States has over eight IoT devices.[10] There is a constant increase in the number of IoT devices in households. This means that there is a direct corollary with the number or size of the data center needed to provide proper services to these devices.

A serious challenge that needs to be addressed immediately is the standards and requirements for IoT platforms. ISO/IEC standard 31041 addresses the IoT and reference architecture. This standard helps smooth over a seamless connection with the intent of being safer. The framework that this standard is a good starting point, but the challenge is the implementation of the standard. It is not a requirement for companies to use. ISO/IEC 27400 family addresses IoT security and data privacy guidelines, but nothing is forcing a company to use or address these standards.

Smart Ecosystem Blockchain Platform Quality Challenges

Blockchain platforms suffer from availability/scalability challenges directly associated with the TRM model. All blockchain platforms can only process so many transactions per second. If the data the blockchain received are greater than the computational capabilities, then the computational ability of the platform slows down and gives the impression of being offline. Blockchains tend to be more secure than traditional computer systems because most systems are distributed. Many are based in the cloud. A distributed and cloud-based platform will always be directly affected by availability and scalability issues. Most blockchains have strong cryptographic principles, strengthening the TRMs assurance and integrity issues.

The TRMs ease-of-use plays an important role in the challenges with blockchain. Many owners of blockchains become landlords to data mining infrastructure, facing challenges to cryptocurrencies and other dilemmas on how the information will be disseminated. The ease-of-use challenge is that while crypto-currencies are relatively easy to use, their values fluctuate greatly, and people do not trust transactions based on fictional money. The different blockchain software packages do not always communicate well with each other, making transaction processing difficult at times. A challenge introduced by cryptocurrencies is the issue of auditing and transaction analysis. Individuals in charge of the blockchain can control what data are collected and what they will do with it. While this does not directly play into the ease of use, you must adhere to the owners' rules, and they can make it as easy or difficult as they wish for you to use their software.

Blockchain platforms also face challenges in terms of adaptability and interoperability. Although the World Wide Web seems to have been around for an eternity, blockchain concepts pre-date it. Cryptographical secured chains were discussed in the early 1990s by Stuart Haber and W. Scott Stornetta—the idea is that cryptographic hashes could be broken up into "blocks" to be transmitted. Choosing the size of the block and communicating these blocks has always been a challenge. The argument of what form of cryptography, or the process of using codes to protect data and other information, is the best and what the best ways of transmitting the information remain challenging to all blockchains. These different modes of transmission and the different cryptography used challenge adaptability and interoperability.

Smart Ecosystems AI/ML Platform Quality Challenges

artificial intelligence (AI) and machine learning (ML) provide a variety of platform and quality challenges. These two types of technology suffer from the same difficulties as the cloud big data analytics platforms discussed earlier, with the addition of TRM quality standpoints. The challenges of availability, assurance, and usability are still present, but this issue is how to ensure quality from an AI/ML perspective. Generative AI can interact with

9 Statista, "Average number of devices and connections per person worldwide in 2018 and 2023," January 19, 2023. https://www.statista.com/statistics/1190270/number-of-devices-and-connections-per-person-worldwide/
10 C. Weinschenk, "Report: People underestimate number of IoT devices in their homes." April 3, 2023. https://www.telecompetitor.com/report-people-underestimate-number-of-iot-devices-in-their-homes/

a customer base, but that customer base usually wants to talk to a "real" person. When used in chatbots and other front-end interfacing devices, users can typically get their answers quickly, but they often lack depth or leave out important facts or features. The generative AI can only produce responses and information based on its calculating algorithms. The Turing model of AI is based on algorithms, while the Lovelace model of AI is based on spontaneous creation of material. We are not in a world where AI/ML devices can "think" and create new content. The generative AI, for instance ChatGPT, uses a predictive text model to form its responses. The quality of responses from such devices is left in question because if the device cannot determine the correct answers, it often makes up the material.

The use of cloud devices based on AI/ML provides the challenge of communication and speed. The availability of data that is accessible and of the highest quality of analysis is always a risk. The need for secure communications and smart backups is also critical. This is a definite concern for compliance issues. The CIS critical controls can be used to evaluate procedures and test compliance levels against company policies. The problem is creating policies that handle all possible combinations of AI/ML and their applications within a company.

In higher education, colleges create ML models based on information collected in student datasets. They use these models to perform targeted marketing in an effort to increase the number of college applications. The challenge is that many of the datasets that are used to train ML models are culturally biased and do not consider all possible learners; they usually have a targeted demographic. Therefore, the new student enrollment plan could be flawed because of a population of students that is not included or represented.

INDUSTRY SPOTLIGHT

Challenges of Health Care Information Systems

According to LinkedIn, health care information systems struggle to provide accurate and complete data. Many fields within patient records are left blank, or the administrative assistant cannot read the patient's handwriting. If you have done any research, you will learn the phrase, "data is dirty." When you ask participants to fill in surveys or forms, you have no idea what kind of data they will be receiving. Therefore, patients' incomplete fields or spelling mistakes all lead to dirty data. With the advancements of AI and ML, data records can be analyzed for completeness, and fields can be evaluated for proper responses.

The use of AI in medical fields is growing rapidly. Tools like ChatGPT have had high success rates in properly diagnosing conditions already verified by doctors.[11] Studies have shown that generative AI tools can be used by doctors to help with diagnostic evaluations of patients. These tools are not 100% accurate, and patients should be wary about only using computer-related devices for diagnosis. You should consult a physician if you feel something is wrong. If tools like ChatGPT can decrease medical professionals' research time, they can find a quicker place to start treatment and be able to treat more patients. Medical professionals should proceed with caution because the studies show there is about a 10% chance of wrongful diagnosis.

Smart Ecosystems 3-D/4-D Platform Quality Challenges

When discussing 3-D/4-D platform systems, there are two areas in which challenges can be analyzed. The first is 3-D/4-D printing, and the second is 3-D/4-D immersive gaming. 3-D/4-D printing is a new exciting concept that allows the materials used in 3-D printing to transition or transmute into other shapes or substances. The concept of 4-D printing is an add-on to 3-D printing with special features. The challenges with this are the materials needed to print in a 3-D environment properly and the transitioning period for the print. There are

11 P. T. Paharia., "ChatGPT: A diagnostic sidekick for doctors? Caution advised for non-professionals." *News Medical Life Sciences*, April 27, 2023. https://www.news-medical.net/news/20230427/ChatGPT-A-diagnostic-sidekick-for-doctors-Caution-advised-for-non-professionals.aspx

typically special environmental conditions that will trigger the transmutation. The challenge with 3-D printing is properly designing the model. Sometimes, these models need to be printed with supports and other features, which is a waste of filament. If that filament is meant for 4-D printing, then that expense must be accounted for. The success rate of 3-D printing is not 100%. Many conditions will cause the print to fail. One of the biggest issues is the filament absorbing moisture from the air. If the 4-D model is based on moisture or water, the transition might be triggered during printing, which would not be the intended outcome. Many 3-D models, if not designed properly, are frail and have many weak points. This is not always desirable if the 4-D transition is going to depend on the strength of the material.

3-D/4-D immersive gaming has all the same concerns as the AR/VR platforms discussed previously in the chapter. The 4-D aspect of the AR is a condition that triggers an alternative sensory response. It goes beyond just the visual aspects of the oculus, but it may incorporate sound or touch. This is common on immersive 4-D rides found at amusement parks. Providing an experience beyond the 3-D interaction has challenges based on environment and materials. Most 4-D immersive experiences use air or mists of water to generate their effects, but some go as far as to incorporate pheromones.

Metaverse Smart Ecosystems Platform Quality Challenges

The metaverse is a big World Wide Web (WWW) addition. The immersive AR/VR culture the platform intends to create is nothing short of a science fiction movie. The WWW and the Internet have had decades to develop and majorly influence modern life. The challenges that the metaverse presents are the issues of adaptability and interoperability and the usability and ease-of-use areas of the TRM. The metaverse prides itself on being platform-independent. It is meant to be used on all IoT devices and all computer platforms. It is reasonable to assume that interoperability will significantly influence how smoothly the metaverse functions and how stable the environment is. The ability to travel seamlessly between virtual worlds and transition from one system to the next will only be as good as the resources introduced into the system. The amount of memory each computer has and cloud technologies will be challenging. The platform will want seamless integration of 3-D modeling, and the adaptability of this has been previously discussed in the chapter.

A second major concern is that the metaverse is not a traditional medium. It is different than current movies or films produced for large screens. These screens can be as small as a watch face or as large as a city skyscraper. All images in the metaverse are digital. There will be no print media from it. Individuals with disabilities will have to adapt to the new culture, and the level of detail within the models may not meet expectations. The platform will be expensive, and in a world where we struggle to ensure everyone has access to the Internet and WWW, making sure everyone has access to the metaverse is a further stretch. Individuals will hesitate to provide all the information needed to have a metaverse account, and therefore, a lot of unauthentic information will be provided. Alias will be used, and sometimes wrong data will be provided just to gain access.

The rate at which the software is being created for metaverse is a definite area of concern in the TRM. Software engineering skills in web technologies, networking, quality assurance, security, and overall software usability need to be increased. Developing these skills is not the biggest challenge. With the advent of the Internet, the modern personal computer, and all of the IoT devices, where does the innovation come into play with the metaverse? It will be years before the current world and experiences in it are fully brought into the metaverse, but we need people who are going to look beyond the metaverse to see where it is going. If people get immersed in the metaverse, looking past it for newer technology won't be easy.

LINK TO LEARNING

Read more about the metaverse (https://openstax.org/r/76metaverse) and how to get started in the immersive culture. This website details how technology will be used to access the metaverse and what steps individuals need to take to move forward with the meta experience.

Industry 5.0 Supersociety Solutions Era

Industry 5.0 is a new and emerging phase of industrialization that sees humans working alongside technology and AI-powered robots. It also relates to a transformation of industries from production-based to value-based, focusing on social and environmental benefits (i.e., Society 5.0) as well as profit-making. Industry 5.0 aims to enhance workplace processes, increase efficiency, and improve resilience and sustainability. Industry 5.0 relies on various supersociety solutions that typically operate in a hybrid cloud environment. Therefore, they rely on the application foundations set forth by the OWP for web/mobile applications as well as services that are made available by multi-cloud platforms and smart ecosystem solutions. The concept of a **supersociety** is a technologically rich environment. The individuals who live in these societies choose to immerse themselves in technology. The household would often be completely reliant on the IoT and cloud environment software. As we prepare for super-societies, think about the science fiction movies we have seen where a computer runs and maintains the entire household. While we are not there yet, the members of super-societies work toward this concept. The infusion of cloud-based software applications and the idea of being fully connected to the Internet plays a significant role in utilizing or being a member of a supersociety.

Supersociety Autonomous Systems Platform Quality Challenges

The challenges we face with an **autonomous system**, which is a system that can operate with limited human control, are ever-changing. The theory is that if every car in the world was autonomous, there would be no more accidents. This is a myth because, as we know, computers are prone to failure. Individuals who wish to hack autonomous vehicles will do so to possibly cause harm. Many people cannot manage the sense of not being in control of a vehicle while it is driving, even when they are behind the wheel. Humans, by nature, prefer to be in control of their environments.

Autonomous vehicles rely on sensory data to function. This data provides perception, decision-making, and execution of real-time data analysis. The vehicles rely on cloud mapping services for navigation and other cloud data for weather and environmental issues. If any of these channels are interrupted, the vehicle must function based on the last known data. This information may be insufficient, or erroneous data may be used for decision-making. This will change the *perception* of the vehicle and may lead to the system's inability to make safe decisions.

The vehicles are expensive to manufacture. Autonomous vehicles can also refer to drones and smart homes. Any device that could work independently of humans is at risk for false data. With smart homes, there is also the challenge of power outages. While these platforms are in constant development, and many companies are contributing to the enhancement and evolution of these devices, as a society, we need to analyze how we will integrate them into our lives. Theoretically, as the autonomous machinery increases, it should seamlessly fit within the metaverse. The TRMs interoperability and adaptability will be challenged, and data mining and decision-making through more developed AI mediums will continue to challenge the functionality of immersive technology.

The great hope for autonomous vehicles, and in reality it is most true, is that autonomous machinery will be faster than human interactive computing and will provide humans the opportunities to repurpose themselves so they do not have to do the tasks these autonomous entities can provide. The issue is that we need more computer scientists studying AI/ML and a greater devotion to nanotechnology. All companies must enhance their quality assurance protocols, and the world cannot be the testbed for software. The ISO 22737:2021 Intelligent Transport Systems and ISO 39003:2023 Road Traffic Safety must continue to evolve to handle the newest technology. Standards cannot be an afterthought for autonomous platforms.

Supersociety Advanced Robotics Platform Quality Challenges

As previously discussed in the chapter, there are two bases of AI in the world: Turing and Lovelace forms of AI. Robotic platforms cannot exist without implementing one of the two types of AI. Since robotics has not been proven to create thoughts spontaneously, the Lovelace form of AI does not seem to be implantable. Therefore,

we can assume that AI based on algorithmic analysis will be the current resolution for robotics with machines working toward the Lovelace form of AI. If we consider humanoid robotics, much can be done with predictive AI and the advancements in neurosciences.

The Human Brain Project seeks to find disorders in human consciousness. Although this project is currently on hold because of its ambitious goals, we can still learn from the foundations they established. Henry Markram, creator of the Human Brain Project, spent many years trying to reverse-engineer the brain to understand brain disorders like autism. In a 2009 TED talk, Markram said that he would be able to figure out the brain and create simulations in the form of holograms that will be able to think and talk with you. His foundation did not meet its goals, but it did advance brain model research in previously analyzed areas. This will help with the creation of robotic brains that can think. The challenge is that the neurological community cannot agree on one concise development course. This has led to the delay and descension of scientists leaving the project.

The TRMs quality assurance and software engineering are challenged every step of the way with robotics. An ethical dilemma also comes with all robotics: Do we need a robot that can function the same way as a human? If that is the case, when they are developed, will humans become obsolete? Science fiction movies have warned about robots taking over the world. If the quality assurance and software engineering skills increase to a point where a robot is created that can truly think, the world will experience a new set of challenges.

Supersociety Nanotechnology Platform Quality Challenges

There are many challenges to nanotechnology (NT) platforms and their introduction into society. Nanotechnologies are small (1 billionth of a meter) devices introduced into technology to help the device function in a modified way, hopefully for the betterment of the technology. The process required to develop such technologies is costly, and many different chemicals are involved. What to do with these chemicals once the manufacturing is done is a constant challenge for the NT companies.

The NT contradicts the TRMs thought process about extensibility. NT is created to perform a specific task; it often cannot adapt to new conditions or environments. The NT has a series of functional requirements that it is meant to carry out. Once the NT is introduced into the system, it is difficult to remove the technology.

NT developments are under the supervision of universities and private companies. Since there is competition to advance these technologies, there is not a lot of information sharing between the entities involved. The duplication of effort with the advancement of technologies and the fact that many employees are required to sign a non-disclosure agreement when working for these companies. The industry does not see as much of a turnaround of employees moving from one company to the next because most technology is proprietary.

There are other ethical challenges with NT that the world does not have answers for. NT can be used to create biological weapons and other powerful weapons. Although these weapons can be developed, the question is, should they be? Or why do they need to be? The fear of atomic warfare has shaped the current military forces. Still, the thought of cybersecurity and NT advancing the attack surfaces is scary and challenging to society. The use of NT to enhance human abilities is also in question. Should NT be used to improve the human brain or muscle system? Can we use NT to enhance beasts of burden to work harder and longer during the day? This would create a divide between wealthy people who could afford the NT implants and those who could not. The technology divide between classes would grow even more. The NT's most significant challenge is the simplest of questions: Is it needed?

Supersociety Super-Compute Platform Quality Challenges

Super-societies cannot exist without advanced technology. This means that data warehousing and communications must be at the performance scale's extreme top end. Supercomputers, often called clusters or parallel computers, are made to share resources and interconnect processing CPUs. The CPUs are grouped in nodes where the communication mediums between the nodes are of high quality and have the fastest speeds. The power and speed of a supercomputer is based on the number of nodes the computer possesses.

Researchers have tried to enhance the supercomputers through NT and mimic the human nervous system. The neuromorphic systems try to emulate the nervous system and the way the brain works to deliver information. In this area, IBM has done extensive research, creating the IBM TrueNorth chip and the Intel Loihi 2 chip. These will eventually replace IBM's Eagle Quantum computer, one of today's top supercomputers.

The TRM challenges introduced through the supercomputer are availability and quality. Most families do not have supercomputers in their homes and do not have access to them. Therefore, unless you are a research scientist working for these organizations, you will probably never have the opportunity to interface with this type of computer. The second challenge of quality is a difficult challenge to measure. Quality with supercomputers comes in different forms. The first is based on speed and efficiency. If the computer is doing what it needs to do and the outcome is accurate, it is usually okay with the users. The second challenge is correlated with speed, which is information accuracy. Many supercomputers produce data at an alarming rate, and it is difficult to check that data in real-time before it can be used in other calculations. Implementing algorithms needs to be unit-tested, and quality assurance is a must. Quality has many meanings to different people, and a computer that can perform tasks as fast as a supercomputer can also be used for malicious purposes. Since the cost of these computers is at the extremely high end of the industry, they are mostly found in government agencies and defense agencies. Using a supercomputer to perform hacking techniques could either ensure success because of the raw computer power, or it could be used to supply the ultimate denial of service attacks.

Super-computers are ideal for analyzing big data. Using these computers to perform calculations on data sets that are generally not manageable by regular computers could advance technology and industries in ways that were not previously thought about. The U.S. government, through NASA, has been analyzing the data captured from telescopes to evaluate different areas of the universe. This is scary to some people because they feel the government could use the computers to analyze privacy issues and to spy on people. It will take a visionary in the computer industry to truly utilize the processing power of these computers. As visionaries, we need people to think about capabilities beyond how computers are currently being used and consider how these devices could be used in the future. This is a challenge because most of the industry is extending or adapting the current technology to perform new tasks, not thinking outside the box on what could be possible with the speeds of modern supercomputers.

Supersociety Autonomous Super-Systems Platform Quality Challenges

The challenges of autonomous super-systems platforms combine all the previous difficulties and significant complications. Since we have not seen a truly autonomous platform that combines robotics and supercomputers, we are left to think about *what-ifs*. What can we expect from a system that can think for itself, regulate itself, fix any issues that might arise, and do it faster than humanly comprehendible? Our only hope as a society is the concept that Dell introduced: the Responsible Computing Framework. The ethical implications are endless if we have autonomous super-systems. However, if these systems are built with the proper series of checks and balances and the technology used to develop the systems is constantly tested for flaws, our only concern is the human factor of implementation. At some point in time, humans will create autonomous super-systems. We can only hope that they are of the highest ethical standards and that introducing these systems is socially acceptable. Suppose individuals adhere to the Responsible Computing Framework and always consider an information security policy when implementing the systems. In that case, we can only hope that the systems will do their needed jobs and do them with the highest ethical standards.

CONCEPTS IN PRACTICE

Industry Certificates Combined with Degree Programs

Professional skills are always needed within the industry, and industry certifications seem to be in demand. Cloud technology companies seem to be thriving, but the engineers who will support them do not appear

to be growing at the rate of demand. Several companies have openings in roles related to cloud technology. Let's look at what Microsoft offers for different certificates pertaining to the Azure Cloud Services.

Microsoft offers a variety of certifications for particular areas when working with its Azure platform. The first recommended certificate is the Azure Fundamentals. This certificate focuses on the learner's skills in cloud concepts, including the benefits of cloud computing and IaaS, PaaS, and SaaS. After this certification, there are a variety of options. The learner can take the Microsoft Azure Data Fundamentals or the Azure Monitoring, Troubleshooting, and Optimizing. There are certificates for Storage Solutions, Subscriptions and Resources, Third Party Connectivity, and Virtual Machines.

Microsoft is not the only company offering skill-based certificates for its technologies. Companies like Google and IBM offer options, and external entities offer programming, software engineering, and security certifications. These certificates are exceptional add-ons to a college education. They help you focus your skillset for the job at hand. The education you receive from a college or university should provide foundational knowledge to build upon.

14.2 Cybersecurity Deep Dive

Learning Objectives

By the end of this section, you will be able to:

- Understand the meaning of cybersecurity
- Learn how to secure information and communication
- Learn how cybersecurity can be used to protect software solutions
- Learn how cybersecurity can be used to protect Internet mobile/web applications
- Understand how cybersecurity can be used to protect cloud-centric solutions
- Relate to cybersecurity as it applies to Industry 4.0 metaverse smart ecosystems
- Relate to cybersecurity as it applies to Industry 5.0 supersociety solutions

As discussed in the previous section, broadscale adoption of cyber resources brings challenges as it seeks to ensure TRM qualities such as security, performance, and reliability. This section focuses on enforcing **cybersecurity assurance**, which is the confidence that every effort is made to protect IT solutions against undesirable use. Cybersecurity assurance is an in-depth topic, and due to lack of space, this section will provide only a brief overview.

What Is Cybersecurity?

The field of **cybersecurity** includes the policies, procedures, technology, and other tools, including people on which organizations rely to protect their computer systems and information systems environments from digital threats. Cybersecurity focuses on five categories of security: network, application, critical infrastructure, IoT, and cloud. To assess an organization's cyber risks and develop cybersecurity assurance, you should ask the following questions:

- What is the threat model?
- Who are the attackers, and what are their capabilities, motivations, and access?
- What are the risks, vulnerabilities, and likeliness of breach per risk assessment?
- What are the technical and nontechnical countermeasures, and how much will the countermeasures cost, including both direct and indirect costs? (Nontechnical countermeasures include laws, policies, procedures, training, auditing, and incentives. Indirect costs can be reputation, or future business.)
- What assets do you seek to protect? (This question relates to security policies that address confidentiality, integrity, service availability, privacy, and authenticity.)
- Who and what do you trust to help maintain cybersecurity?

Cybersecurity threats include behaviors such as

- breaching privacy by revealing confidential information such as corporate secrets, private data, or personally identifiable information (PII);
- damaging integrity/authenticity by destroying records, altering data, or installing unwanted software (e.g., spambot, spyware); and
- denying access to a service through activities such as crashing a website for political reasons, causing a denial-of-service attack, or allowing only certain individuals to have access to services.

In 2001, the **Open Web Application Security Project (OWASP)** was launched with the purpose of securing web applications. This model was only concerned with actionable controls and possible risks. The controls were focused on securing the risks involved with the development and deployment of the applications (Table 14.3).

Rank	Risk
1	Broken access control
2	Cryptographic failures
3	Injections
4	Insecure design
5	Security misconfigurations
6	Vulnerable and outdated components
7	Identification and authentication flaws
8	Software and data integrity failures
9	Security logging and monitoring flaws
10	Server-side request forgery

Table 14.3 Open Web Application Security Project Top 10 (Source: Indusface. "OWASP Top 10 Vulnerabilities in 2021: How to Mitigate Them? February 24, 2022. https://www.indusface.com/learning/what-are-the-owasp-top-10-risks-2021/)

LINK TO LEARNING

To achieve security, we must eliminate defects and design flaws in systems and make them harder for hackers to exploit. This includes developing a foundation for deeply understanding the networked information systems we use and build. We also must be aware that no system is completely secure. To learn more, read this article about the security mindset (https://openstax.org/r/76security) from the *Journal of Cybersecurity*.

Why Is Cybersecurity Important?

In 2023, the average cost of a data breach was $4.45 million globally, which, in just three years, was a 15% increase over 2020.[12] These costs cover expenses incurred to discover and respond to breaches, lost revenue from downtime, and long-term damages to the business reputation and brand. Cybersecurity Ventures recently shared its top ten cybersecurity predictions and statistics for 2024, unveiling the alarming fact that global cybercrime financial damage will likely reach $10.5 trillion by 2025. Based on this total, if cybersecurity is regarded as a country, it would rank third, trailing only the United States and China as the world's largest economy.[13] Personally identifiable information (PII) is generally the target of cybercriminals. They collect information, such as names, addresses, identification numbers, and credit card information, and sell them in the dark web or other underground marketplaces. This results in several consequences for these organizations, including regulatory fines, legal action, and the loss of customer trust.

Cybersecurity is increasingly important as the cost of breaches increases. However, a comprehensive cybersecurity strategy based on best practices and using machine learning, advanced analytics, and artificial intelligence (AI) can effectively combat cyber threats and reduce the impacts of breaches.

CONCEPTS IN PRACTICE

Think like a Hacker

To implement effective cybersecurity policies and procedures, cybersecurity experts need to understand hackers, including how they think and how they are likely to target an organization. What does it mean to think like a hacker?

Skilled hackers tend to be inquisitive, with an in-depth understanding of technology and its capabilities. They stay abreast of technological advances, including cybersecurity measures. They tend to be creative thinkers with excellent analytical skills, which makes them capable of finding innovative ways to circumvent cybersecurity features and hack into a system. They also tend to understand human nature, including our weaknesses and vulnerabilities.

To think like a hacker, you must develop these skills and learn to use them with a hacker mindset, constantly reviewing your system and recognizing how it can be hacked. This includes prioritizing security and remaining aware that hackers constantly look for weaknesses that enable them to exploit a system. As part of this, learn about ethical hacking, which includes penetration testing, and use ethical hacking practices to ensure your system is up-to-date and ready to withstand cyberattacks.

Domains of Cybersecurity and Associated Cryptography Techniques

Cybersecurity includes various domains that comprise overall cybersecurity. Common cybersecurity domains include the following:

- Protection of computer systems and networks that ensure security at a national scale, economic health, and public safety, called **infrastructure security**. The National Institute of Standards and Technology (NIST) cybersecurity framework is meant to help organizations in this area. Additional guidance is provided by the U.S. Department of Homeland Security (DHS).
- Protection of wired and wireless (Wi-Fi) computer networks from intrusions, called **network security**. The designs for operating systems, virtual machines, and monitors must include protections against unauthorized use.
- Protection of applications on premise and in the cloud, with security integrated during the design phase

12 IBM. 2023. "Cost of a Data Breach Report 2023. https://www.ibm.com/reports/data-breach.
13 Morgan, Steve. 2024. "Top 10 Cybersecurity Predictions and Statistics for 2024." *Cybercrime Magazine.* https://cybersecurityventures.com/top-5-cybersecurity-facts-figures-predictions-and-statistics-for-2021-to-2025/

for data handling and user authentication, called **application security**. The process of authentication confirms the identity and authorization of people and devices that use a system. Many resources support these efforts, including books and online materials on security engineering, “robust” programming, and secure programming for specific operating systems.

- Protection of the data, digital files, and other information a system maintains, called **information security**. For example, database systems must protect against SQL injection by ensuring that queries issued to the database do not somehow contain malignant code that could compromise the security of the database and its infrastructure. Information security requires data protection measures, such as the General Data Protection Regulation (GDPR), that secure sensitive data from unauthorized access, exposure, or theft.

These domains function cooperatively to create a cybersecurity risk management plan (Figure 14.7).

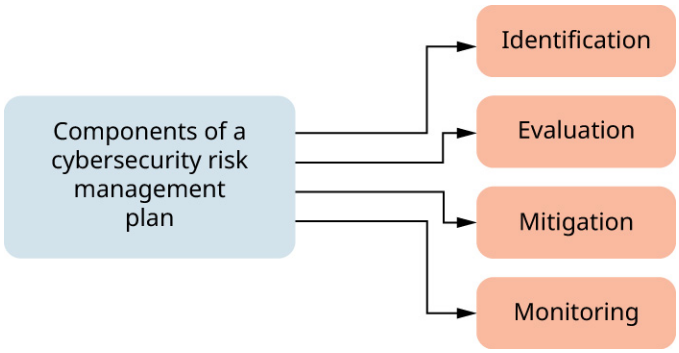


Figure 14.7 Cybersecurity should be based on a comprehensive risk management plan that identifies and evaluates cybersecurity risks, designs and implements practices to mitigate those risks, and monitors the system and identifies areas that need to be updated. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

An important pillar of cybersecurity assurance is **nonrepudiation**, which provides proof of data’s origin, authenticity, and integrity. Nonrepudiation is achieved through cryptography using, for example, digital signatures, which are used in online transactions to ensure that a party cannot later deny placing an order or denying its signature’s authenticity. In addition to digital signatures, nonrepudiation is also used in digital contracts and emails. Email nonrepudiation involves methods such as email tracking to assure senders that messages are delivered and also provides recipients with proof of the sender’s identity. This ensures neither party can deny that a message was sent, received, and processed.

Beyond basic securing of information systems, cybersecurity requires creating and governance of processes that protect organizations and individuals against costly breaches. End-user education is a critical part of this process. Organizations must promote security awareness to enhance endpoint security. For example, training users to delete suspicious email attachments and avoid using unknown USB devices can help.

Disaster recovery and business continuity planning are also essential to minimize disruption to key operations. This requires tools and procedures to respond to cybersecurity incidents, natural disasters, and power outages. Data storage is another critical area. Protection measures that promote data resilience with safeguards include encryption and immutable and isolated data copies that can quickly be restored to recover data and minimize the impact of a cyberattack.

Evolvability is an important cyber quality, but the evolution of platforms on which information systems may be deployed creates a need for new security measures. For example, mobile solutions security requires specific protection measures to protect applications, containers, and mobile mail. Furthermore, when using the cloud, organizations must ensure confidential computing by encrypting data at rest (i.e., while data are stored in the cloud), during transfer, and during processing. This ensures that processes meet business requirements and regulatory compliance standards and supports customer privacy.

Cybersecurity Misconceptions

Because of the constant evolution of technology, there are also several misconceptions surrounding cybersecurity risks.

1. Threats only come from outside the organization. Instead, breaches can involve people within the organization working maliciously with external hackers or organized groups.
2. Risks are known and can be predicted. In reality, the attack surface is constantly expanding with new vulnerabilities, and human error by negligent employees or contractors can also increase the risk of data breaches.
3. Attack vectors are limited. Cybercriminals constantly discover new attack vectors through various environments.
4. The industry is safe. Every industry faces cybersecurity risks, and adversaries exploit communication networks across the governments and private sector. Ransomware attacks are expanding to nonprofits and local governments, and threats are increasing in other areas.[14]

Supply Chain and Security Issues

After the COVID-19 pandemic, companies still struggle to get necessary goods and services. Many products are still in ports and warehouses waiting to be delivered because there are not enough truck drivers in the workforce. In the wake of the chip manufacturing plant shortage, the automobile industry started to recover in 2023. For many months, car dealership lots were empty of new cars. There is a constant strain on workers to get goods and materials to the customers quicker. Amazon has set a precedence for the delivery rate of goods that the rest of the world is struggling to keep up with.

The world has seen increased cyber threats because more businesses are being conducted online. If you go to any sports stadium or concert event, most of them involve cashless transactions. All transactions must be done with credit or debit cards. This means there are many more chances for your information to be leaked because of the number of companies with which you are interfacing. Granted, it is your choice to use these businesses, but remember, security is only as strong as its weakest link. As our culture demands we go to cashless transactions, the number of available attack surfaces for hackers increases. If we implement the concepts of supersocieties and immerse ourselves in this technology, then if a security vulnerability compromises us, it may do irreparable harm to our lifestyle.

Common Cyber Threats

Cyber threats are constantly evolving, and there are various common cyber threats of which programmers should be aware. Malicious software variants such as viruses, worms, Trojans, spyware, and botnets that allow for unauthorized access or can damage computers is called **malware**. Modern malware attacks often bypass traditional detection methods, such as antivirus scans for malicious files, by being "fileless." Types of malware include the following:

- Viruses contain code that propagates or replicates across systems by arranging to have itself eventually executed, creating additional, new instances of itself; it generally infects by altering stored documents or code (e.g., boot sector or memory-resident code), typically with the help of a user.
- Worms contain code that replicates itself across a network, infecting connected computers without user intervention.
- Rootkits modify the operating system (e.g., modify system exploration utilities, replace the target OS with a virtual machine monitor that can attack systems) to hide their existence.
- A backdoor is a concealed feature or command within a program that enables users to execute actions they normally would not be permitted to do; sometimes called a trapdoor (e.g., Easter egg in DVDs and software).
- A Trojan horse is software that seems to serve a beneficial purpose but is designed to execute covert

14 Kizzee, Ken. 2024. "Cyber Attack Statistics to Know," *Parachute*. https://parachute.cloud/cyber-attack-statistics-data-and-trends/.

malicious activities. An example is spyware, which can be installed by seemingly legitimate programs and then provides remote access to the computer for activities such as keylogging or sending back documents.
- Botnets involve a network of compromised machines (bots) under (unified) control of an attacker (botmaster); once the botmaster has control, the attacker has access to the devices and their system, enabling the attacker to steal data, execute scams, and perform other malicious tasks.

The scenarios that enable malware to run include

- a vulnerable client, such as a browser, connecting to a remote system that delivers an attack;
- exploitation of a network-accessible service with buffer overflow vulnerability,
- malicious code introduced into a system component during manufacturing, through a compromised software provider, or via a man-in-the-middle (MitM) attack;
- using the autorun functionality, especially through the insertion of a USB device;
- deceiving a user using social engineering into running or installing malicious software; and
- an attacker with local access directly downloading or running malicious software, potentially using a "local root" exploit for elevated privileges.

Some of the dangers that can occur as a result of malware include generating a pop-up message with a brag, exhort, or extort; trashing files; damaging hardware; launching external activity; stealing information; and keylogging via screen, audio, or camera capture or via file encryption, such as ransomware. Malware that encrypts data and demands a ransom to unlock or prevent data exposure is called **ransomware**. An **insider threat** is posed by current or former employees, partners, or contractors who misuse their access. It can also include vulnerabilities intentionally created by programmers, such as malware. The form of social engineering that tricks users into providing personal information through fake emails or text messages posing as legitimate companies is called **phishing**.

Additional threats included a **distributed denial-of-service (DDoS) attack**, which overloads a server with traffic to crash a server, website, or network. Multiple coordinated systems often overwhelm enterprise networks by attacking devices such as modems, printers, and routers that use the Simple Network Management Protocol (SNMP). An **advanced persistent threat (APT)** is used by intruders to infiltrate systems to spy on business activities and steal sensitive data while remaining undetected and leaving the networks and systems intact. A **man-in-the-middle** attack is used by cybercriminals to eavesdrop on and intercept communications between two parties in order to steal data (most often on unsecured Wi-Fi).

GLOBAL ISSUES IN TECHNOLOGY

Technology Skills Gap

A big challenge in technology is the growing issue of the skills gap. There is a mismatch between what is needed in the industry and what students learn in colleges and universities and industry certificates. Many companies are advertising for software engineers with three to five years' experience and not entry-level positions. The demand on companies forces them to look for workers who can hit the ground running with little supervision. The need for entry-level jobs is increasing, but the pay scale is not what people can afford to take after graduating from college. The amount of debt students are accruing to earn their degrees is increasing, and students are forced to take jobs they know they won't like, only to get the experience to jump to a better-paying job. The skill gap is increasing in the areas of cybersecurity and data analytics.

The cybersecurity skills gap is framed from the perspective of individuals in the industry repurposing themselves. People who have been in the computer industry are retooling themselves to be part of the cybersecurity demands. If you are an IT professional with years of experience and have a certificate or degree in cybersecurity, it is hard to start over at the lower end of the job pool. These individuals want

management or higher-end jobs, but they have only entry-level skills. This creates a divide because these individuals have the experience to be successful in the industry and don't need as much assistance as a pure entry-level employee.

Key Cybersecurity Technologies and Best Practices

Key cybersecurity technology and associated best practices typically fall under three categories: identity and access management, a data security platform, and security information and event management.

The technology used to manage users' roles and access privileges is **identity and access management (IAM)**. Approaches used to allow access to systems include single sign-on (SSO), multifactor authentication, privileged user accounts, and user life cycle management. SSO keeps users from entering their credentials multiple times within the same session. Multifactor authentication leverages multiple access credentials provided via different devices. Privileged user accounts are only accessible to specific users. User life cycle management handles user credentials from initial registration to retirement. Cybersecurity professionals use IAM tools to investigate and respond to security issues remotely and contain breach damages.

A **data security platform** is meant to automate the proactive protection of information via monitoring and detecting data vulnerabilities and risks across multiple environments, including hybrid and multicloud platforms. The protection provided by a data security platform simplifies compliance with data privacy regulations and supports backups and encryption to keep data safe.

The practice of **security information and event management (SIEM)** focuses proactively on the automated detection and remediation of suspicious user activities based on the analysis of security events. SIEM solutions typically analyze user behavior and use artificial intelligence (AI) techniques to detect and remediate suspicious activities according to the organization's risk management guidelines. SIEM tools may be integrated with security orchestration, automation, and response (SOAR) platforms designed to fully automate the management of cybersecurity incidents without human intervention.

Some organizations also choose to use a zero-trust strategy, which assumes the system is compromised and sets up controls to continuously validate every user, device, and connection in the system for authenticity and purpose.

LINK TO LEARNING

Cybersecurity is big business, and many companies in the technology industry have developed cybersecurity solutions and services that they sell to other organizations. In particular, IBM provides a variety of such solutions and services (https://openstax.org/r/76ibm) to cover all aspects of cybersecurity, including AI and cloud security.

Securing Information and Communication

Cryptography is an essential tool for securing information systems, which use codes to protect data and other information. With cryptography, information is encrypted and accessible only by those authorized to decrypt and use the information.

Cryptography can help ensure properties such as confidentiality (i.e., secrecy, privacy), integrity (i.e., tamper resilience), authenticity, availability, and nonrepudiability (or deniability).

As shown in Figure 14.8, the components of cryptography include the following:

- plaintext: refers to the data that need protection

- encryption: handled by an algorithm, and creates ciphertext and an encryption key for plaintext
- ciphertext: uses an encryption key to scramble plaintext
- decryption: also handled by an algorithm, and uses a decryption key to transform the ciphertext back into plaintext
- encryption key: value known to the sender of the data; by inputting the encryption key into the encryption algorithm, the sender converts the plaintext into ciphertext
- decryption key: value known to the receiver of the data; by inputting the decryption key into the decryption algorithm, the receiver converts the ciphertext back into plaintext

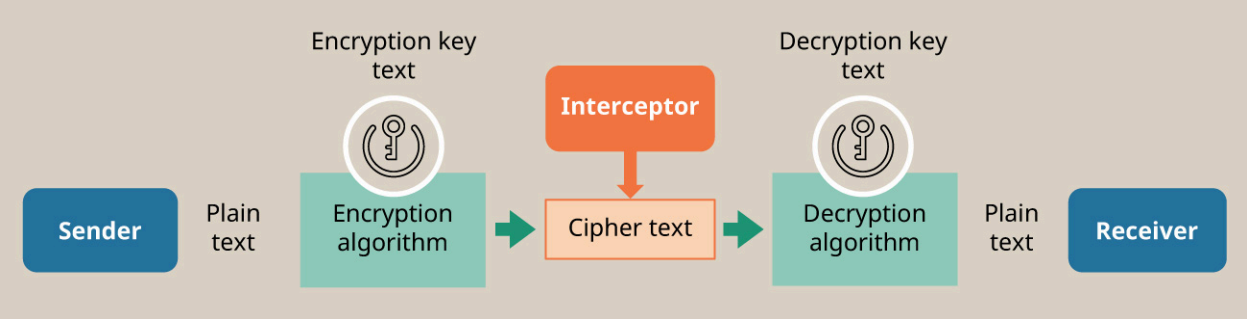


Figure 14.8 Cryptography is a process that enables senders and receivers to secure digital data by using encryption and decryption algorithms. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit key icon: modification of "Accessible" by Phạm Thanh Lộc/Wikimedia Commons, CC BY 4.0)

Cryptography can be used to create different types of ciphers, including the following:

- Substitution: Ciphertext is created by substituting plaintext characters, bits, or character blocks with alternate characters, bits, or character blocks. The substitution can be monoalphabetic, meaning that if the letter D is enciphered as P in one part of the ciphertext, D will be used for P throughout the message. The substitution can also be polyalphabetic, meaning that while D may be enciphered as P in one part of the message, in another part, D may be encoded as a different letter.
- Transposition: Instead of substituting the letters and characters, they are rearranged using a specific algorithm, such as writing a message vertically to produce a ciphertext that is read horizontally. Transposition ciphers are also known as permutation ciphers.
- Polygraphic: Substitution is performed on two or more blocks of letters simultaneously.

Cryptography can also be performed using asymmetric encryption (also known as public-key cryptography), which uses private and public keys. With public-key cryptography, the sender and receiver have a preshared secret key to handle encryption and decryption. In public-key cryptography, the sender uses the receiver's public key to encrypt the message and then send that ciphertext to the receiver. Only the receiver has the private key that is needed to decrypt the message.

Authentication and Passwords

Authentication confirms the identity and authorization of people and devices that use a system. It is a vital two-step process to help ensure that only authorized users have access to a system. Authentication first requires identification followed by verification to establish and confirm a user's unique credentials and ensure the user is authorized.

A **password** is a secret string of characters used to gain entry into a system. It is a critical part of authentication. To function as a cybersecurity tool, passwords must be secure, which can be challenging. Attackers can steal passwords by guessing, installing a hardware or software keylogger, finding written passwords, obtaining them via social engineering/phishing, intercepting the password over the network, or stealing them from a service or third party.

To address these cyber threats, the DHS's Cybersecurity and Infrastructure Security Agency offers three tips for passwords:[15]

- Create long passwords with at least 16 characters.
- Make passwords random by following NIST's recommended password rules.
- Ensure that passwords are unique by using a different password for every access point in a system.

Access Control

A vital part of cybersecurity is **access control**, which regulates the people and devices that can use a computer system's resources. The three most common access control designs include the following:

- Mandatory access control (MAC) is a strict system where security decisions are fully controlled by a central policy administrator, making it impossible for users to set permissions irrespective of ownership.
- Discretionary access control (DAC) is a system where users can set permissions for their files, including granting access rights to other users on the same system. DAC is the most common access control design approach in commercial operating systems. While generally less secure than mandatory control, DAC is easier to implement and more flexible.
- Role-based access control (RBAC) is a system in which access depends on an individual or group's role in an organization and the access they need to meet job requirements. Typically, roles with more responsibility and authority have greater access to the system.

The typical attacks launched against cryptography generally involve the following:

- brute force (e.g., try all possible private keys)
- mathematical attacks (e.g., factoring)
- timing attacks (e.g., based on knowledge of the time it takes to decrypt)
- hardware-based fault attack (e.g., take advantage of faulty hardware to generate digital signatures)
- chosen ciphertext attack (e.g., gather information by obtaining the decryptions of chosen ciphertexts and attempt to recover from this information the secret key used for decryption)
- architectural changes (e.g., use knowledge of vulnerabilities)

Anonymity and Privacy

Protecting anonymity and privacy is an important aspect of cybersecurity. Being able to interact on the Internet, even publicly, while concealing your identity, is considered **anonymity**. Anonymity is not the same as secrecy/confidentiality. As discussed previously, confidentiality is about message contents (i.e., what was said). Anonymity is about identities (i.e., who said it and to whom) and must be preserved to ensure certain civil liberties such as autonomy, free association, free speech, and freedom from censorship and surveillance.

There is a wide spectrum of "nimity," including

- linkable anonymity (e.g., loyalty cards, prepaid mobile phone),
- (e.g., pen names on blogs),
- unlinkable anonymity (e.g., paying in cash), and
- verinymity (e.g., credit card numbers, driver licenses, addresses).

By remaining anonymous, users make it difficult for hackers to steal their personal data (e.g., passwords, credit card information). It also allows users to preserve their civil liberties (e.g., free speech and association) on social media. Anonymity can also be important for people concerned about their safety and do not want to create safety risks because of their online activities.

Keeping your actions online, such as messages intended only for certain individuals, concealed from the public is called **privacy**. Privacy is regarded as a basic human right. While not explicit in the U.S. Constitution, privacy rights are implied by the personal protections offered in the First, Third, Fourth, Fifth, and Ninth Amendments in the Bill of Rights. As Figure 14.9 shows, privacy is related to anonymity but is a separate concept. Both anonymity and privacy are important to promote cybersecurity.

15 Cybersecurity & Infrastructure Security Agency. No Date. "Use Strong Passwords." https://www.cisa.gov/secure-our-world/use-strong-passwords.

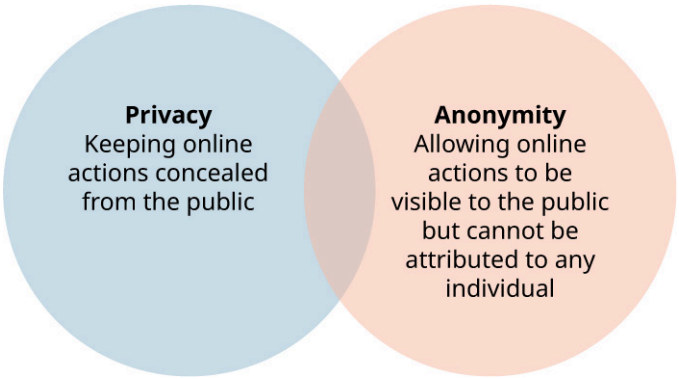


Figure 14.9 Privacy and anonymity are related concepts that are important to promote cybersecurity. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Internet anonymity is difficult, if not impossible, to achieve for average users, but it seems easy for unethical people. Anonymity is a state-of-the-art technique that relies on proxy intermediaries that relay Internet traffic through trusted third parties. Generally, the process requires setting up an encrypted virtual private network (VPN) to the third party's site, and all your traffic goes through it.

To understand how unethical people use proxies, assume a scenario depicted in Figure 14.10 where Alice wants to message M to Bob. Bob does not know that the message M is from Alice, and Eve ("Eve" is short for *eavesdropper*) cannot determine that Alice is communicating with Bob. HMA accepts messages encrypted for it, then extracts the corresponding destination addresses and forwards the message accordingly.

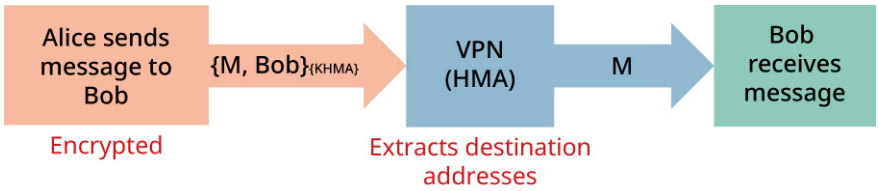


Figure 14.10 When Alice sends a message to Bob, HMA accepts encrypted messages, then extracts the corresponding destination addresses and forwards the messages accordingly. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Anonymity is meant to counter the type of surveillance that is mandated as part of the Patriot Act (section 215, and national security letters or NSLs) and the FISA Amendment Act.

An older Google transparency report (Table 14.4) shows the number of NSLs that FBI agents issued without a judge's approval to obtain personal information.

Reporting Period	National Security Letters	Users/Accounts
January to June 2016	0–499	500–999
July to December 2015	1–499	500–999
January to June 2015	0–499	500–999
July to December 2014	0–499	500–999
January to June 2014	500–999	500–999
July to December 2013	500–999	1,000–1,499

Table 14.4 Google Transparency Report Data from Google.

Reporting Period	National Security Letters	Users/Accounts
January to June 2013	0–499	500–999
July to December 2012	0–499	500–999
January to June 2012	500–999	1,000–1,499
July to December 2011	0–499	500–999
January to June 2011	0–499	500–999
July to December 2010	0–499	1,000–1,499
January to June 2010	500–999	1,500–1,999
July to December 2009	0–499	500–999
January to June 2009	0–499	500–999

Table 14.4 Google Transparency Report Data from Google.

NSLs cover everything except the contents of your communications (i.e., if, when, how much, who). These data are included in the exception because such metadata often provides privileged information that is essentially, according to the FTC, a "proxy for content." In fact, the U.S. National Security Agency (NSA) collection of bulk call data was ruled illegal in 2015.

Encryption tools such as Pretty Good Privacy (PGP), which was introduced earlier, can be used. GnuPG is a free software recreation of PGP created by Phil Zimmerman (1991). These tools allow users to encrypt emails to hide their content by creating a hash of the email's content and using digital signatures to sign the hash. Both the message text and the digital signature attached to the message are protected using hybrid encryption. Digital signatures require public-key cryptography (which has separate public and private keys). Before sending a message, the message is signed with the signature key, and both the message and its signature are encrypted with the recipient's public encryption key. Once the message is received, it is decrypted with the private key to extract it and its signature. The sender's public verification key is then used to check the signature's authenticity.

Fingerprints must be used to secure this process further. Because Bob's public key can be obtained from his website, it needs to be verified via out-of-band communication of fingerprints. A fingerprint is a cryptographic hash of a key. To support this approach, you need key servers that store public keys to look up keys by name/email address and verify them with fingerprints. If you do not know Bob personally, you can rely on the Web of Trust (WoT) or "friend of a friend" mechanism (i.e., Bob introduces Alice to Caro by signing Alice's key).

There are drawbacks to (just) using encryption because Bob's private key may be compromised. In that case, the specifics of the keys may become known, and past messages may be decrypted and read. Because the sender's signature is available as part of the messages sent, it also becomes possible to prove the sender's identity, which defeats the security scheme. This attack exposes many incriminating records, including the key material that decrypts data sent over the public Internet and signatures with proofs of who said what.

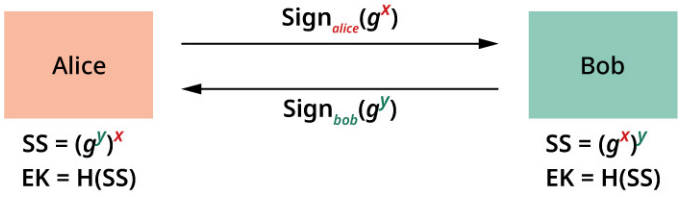
There is nothing better than "off-the-record" conversations where Alice and Bob talk in a room, and no one else can hear them. In that case, no one else knows what they say unless Alice or Bob tells them. Furthermore, no

one can prove what was said, not Alice or Bob. Based on this, desirable communication properties are as follows:

- Deniability makes it plausible to deny having sent a message.
- Forward secrecy allows past messages to be safe even if key material if compromised.
- Mimic off-the-record conversations to facilitate deniable authentication. Make it possible to be confident of who you are talking to but unable to prove to a third party what was said.

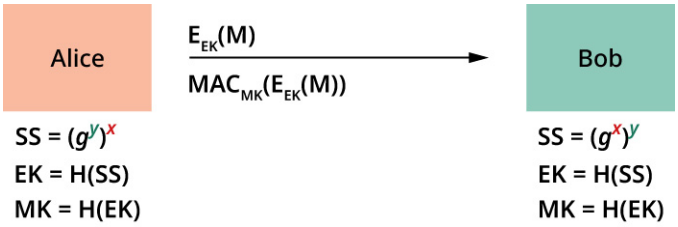
One technique is to use off-the-record (OTR) messaging.

1. Use authenticated Diffie-Hellman (DH) protocol to establish a (short-lived) session key:
   Diffie-Hellman is a security algorithm with only one (symmetric) private key that is shared by both participants (e.g., Alice and Bob). Alice and Bob agree on values for prime number *p* and a generator number *g* (or base), where 1< g < p and g is any number agreed upon by both parties that is a generator of p. The number g is a generator of p because, when raised to positive whole-number powers less than p, it never produces the same result for any two such whole numbers. For ease of computation, g is usually chosen small, and the order of g should be prime and approximately p/2.
   Alice and Bob pick private values x and y respectively and they generate a key and exchange it publicly. So Alice selects x and generates public key a = gx modulo p and Bob selects y and generates public key b = gy modulo p. For example, if p = 23 and g = 9, the private keys for Alice and Bob are respectively 4 and 3, and the secret keys for Alice and Bob are both 9.
   During the DH key (signed) exchange, the only pieces of information that are exposed to the public (and susceptible to interception by malicious actors) are x, y, p, and g. None of which are sufficient to recover Alice and Bob's private keys. It is also not enough information to recover the shared (symmetric) secret (SS) cryptographic key. SS can then be used by Alice and Bob to send encrypted messages to each other safely, which is done using a secret-key encryption algorithm, using a hash of SS as the encryption key EK, to transmit ciphertext.
   The strength of the scheme comes from the fact that (gy)x modulo p = (gx)y modulo p is a one-way function that takes an extremely long time to compute/invert using any known algorithm (just from the knowledge of p, g, gx modulo p, and gy modulo p.



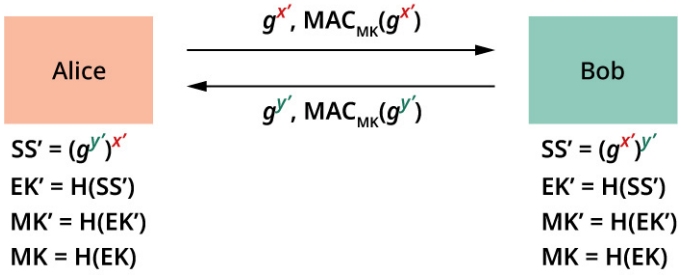
(attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

2. Then Alice uses secret-key encryption on message M and sends it across as EEK(M) … and authenticates the message using a message authentication code MACMK(EEK(M), where MK is computed as a hash of the EK, H(EK):



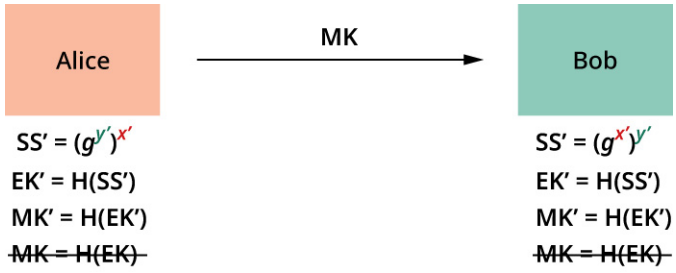
(attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

3. Re-keying is performed using the DH protocol to exchange new private values x′ and y′:



(attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

4. Publishing the old MK:



(attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Note that OTR is more applicable to interactive communication than email. It provides message authentication, confidentiality, deniability, and forward secrecy. However, there are no practical examples of "deniability." OTR is built into Adium and Pidgin; in that case, some defaults apply. In particular, logging is enabled by default, and etiquette dictates you should disable this, as do past instances where people's activities were discovered using those logs (e.g., Chelsea Manning, who leaked classified information, was discovered after the Army reviewed access logs). It is very different from Google Hangout's OTR feature, which does not allow the conversation to be logged.

An interesting anonymity solution is the protocol behind the Signal app (iPhone, Android), which uses the double ratchet algorithm set forth by Trevor Perrin and Moxie Marlinspike. It provides forward secrecy (i.e., today's messages are secret, even if the key is compromised tomorrow), future secrecy (i.e., tomorrow's messages are secret, even if the key is compromised today), deniability (no permanent/transferable evidence of what was said), and usability (i.e., tolerates out-of-order message delivery).

Another interesting idea for anonymity is achieved via plausibly deniable storage. In this case, the goal is to encrypt data stored on your hard drive. Someone can be compelled to decrypt it. The idea is to have a "decoy" volume with benign information (e.g., VeraCrypt).

Note that it may be worthwhile to differentiate sender from receiver anonymity. An interesting example of a protocol that achieves sender anonymity is described by David Chaum as the "dining cryptographers problem."[16] In this case, three cryptographers dining together are told that payment will be made anonymously by either one of them or the agency that employs them. As they respect each other's right to make an anonymous payment but wonder if their agency will be paying, they carry out the protocol described to achieve sender anonymity.

A naive solution to achieve anonymity for browsing is to use VPNs. Organizations providing these may receive court orders asking for information relating to an account, and they will cooperate with law enforcement if they receive a court order. A better approach is to use Tor by downloading the Tor browser bundle (https://openstax.org/r/76TorProj) or becoming a volunteer in the Tor network. Tor is built on a modified version of Firefox and is a low-latency anonymous communication system that hides metadata (i.e., who is communicating). As noted earlier, you may get in trouble when an encrypted message you are sending is

16 Chaum, David. "The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability. *Journal of* Cryptology, vol. 1, 65–75, 1998. https://chaum.com/wp-content/uploads/2022/01/chaum-dc.pdf

intercepted, and the included metadata are exposed. To avoid this, Tor completely hides the existence of communication (e.g., web connections). Tor operates at the transport layer and makes it possible to establish Transmission Control Protocol (TCP) connections without revealing your IP address. The Tor network relies on many nodes (i.e., onion routers) operated by volunteers and located worldwide. The Tor approach becomes useful if Alice wants to connect to a web server without revealing her IP address. Simply speaking, onion routing (Figure 14.11) generalizes to an arbitrary number of intermediaries (“mixes” or “mix-nets”). Alice ultimately wants to talk to Bob, with the help of HMA, Dan, and Charlie, and as long as any of the mixes is honest, no one can link Alice with Bob.

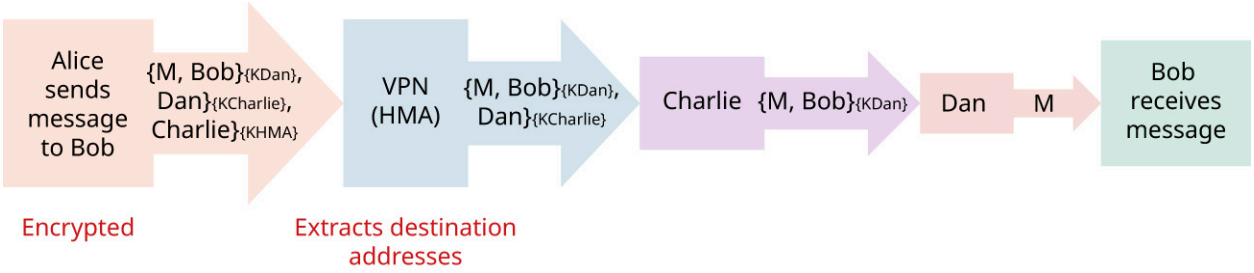


Figure 14.11 Users can achieve anonymity by using Tor. This is what the industrial-strength Tor anonymity service uses. It also provides bidirectional communication. The key concept here is that no one really knows both you and the destination. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Tor end-to-end paths are called Circuits, and Tor almost always uses 3-hop circuits (i.e., k = 3). Tor balances anonymity (i.e., k not too small so as to be traceable) and latency (i.e., k not too large). The last node in a Tor circuit is called an "exit” node. To the outside world, the Tor exit node is initiating connections to destinations. Every peer in the Tor network gets to decide whether to be an exit node or to just relay between other Tor nodes. Tor exit nodes also determine what IP addresses/websites to exit. Tor clients learn about other Tor peers by downloading a list of them.[17] The list provides information for each one of the Tor routers, such as IP address and hostname, the country where it resides, the uptime, the average throughput, websites it is willing to be an exit node for, and the node’s public key.

Tor’s attack model is more “relaxed” than the models used by typical mix-nets. In particular, Tor does not assume a global, passive attacker. It does assume that a limited subset of the Tor nodes are malicious and that there may be some level of eavesdropping on small portions of the links but not a global view of all traffic. This relaxed attack model, which assumes a less powerful adversary, makes it possible for Tor to achieve better performance. That said, there are a few aspects that Tor does not cover as compared to typical mix-nets: Tor does not batch or delay packets. If only one client were to communicate over Tor, there would be no anonymity. The philosophy behind the Tor relaxed attack model is to assume that if the performance is reasonable enough, users will be more likely to adopt it. The more users adopt it, the more "cover traffic" there will be, making it harder for an attacker to map packets to any one sender. Figure 14.12 summarizes how Tor works at a high level.

17 A list of active Tor peers (i.e., Tor routers) is available at https://torstatus.rueckgr.at.

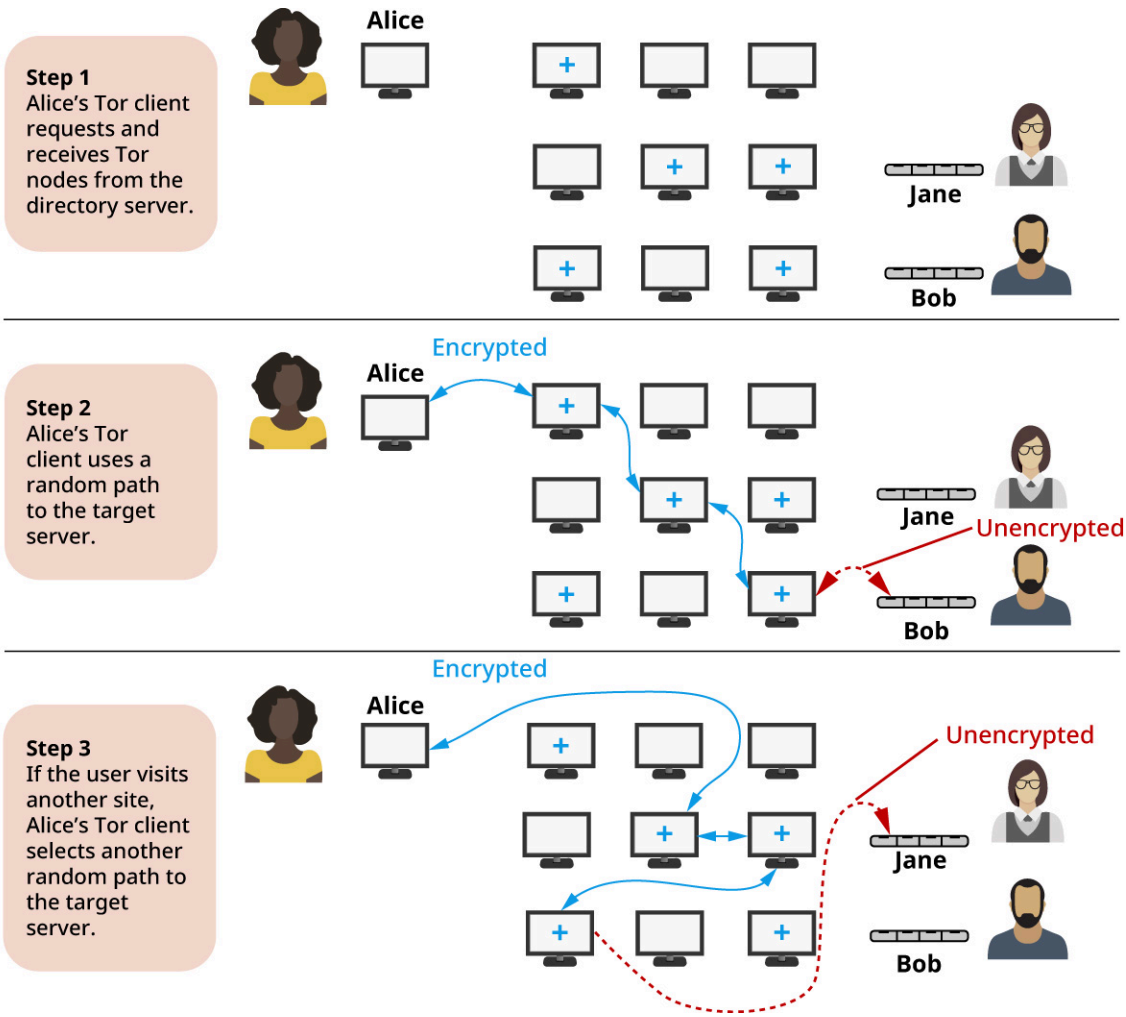


Figure 14.12 Tor is a multistep process that enables users to achieve anonymity online. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit key icon: modification of "Accessible" by Phạm Thanh Lộc/Wikimedia Commons, CC BY 4.0; credit: reproduced with permission from Dave Levin)

Tor implements the following trust protocol:

- Entry node knows that Alice is using Tor, and also knows the identity of the middle node but does not know the identity of the destination.
- Exit node knows that some Tor user is connecting to the destination but does not know which user.
- Destination knows a Tor user is connecting to it via the exit node.

It should be noted that Tor does not provide encryption (e.g., use HTTPS) between the exit node and the destination). We discussed earlier how senders can hide their identities. Similar to that, Tor is also a means of allowing destinations to hide their identities. An example was The Silk Road, an eBay-like online store where users could purchase illicit and illegal goods and pay for them using Bitcoins. Running such a website requires a certain degree of anonymity. Therefore it was run as what is known as a "Tor hidden service." Hidden services have since been renamed to onion services.[18] Interestingly, onion services can achieve receiver anonymity using techniques that achieve sender anonymity (Figure 14.13).[19]

18 To read more about how onion services work, visit https://community.torproject.org/onion-services/overview/, and to read more about how to set them up, visit https://community.torproject.org/onion-services/setup/.
19 For more information on Tor, how it is used, and where it is being censored, check out the Tor metrics site at https://metrics.torproject.org/.

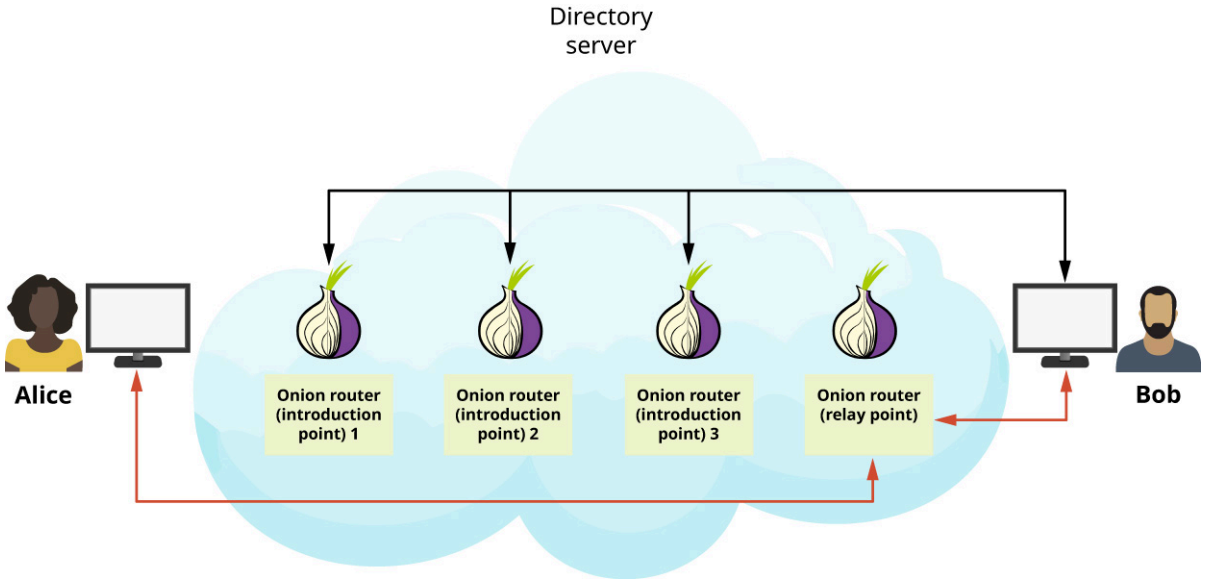


Figure 14.13 Tor hidden services, also known as onion services, achieve receiver-anonymity using techniques that achieve sender-anonymity. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Note that there are various known onion routing attacks and other issues as follows:

- Attacks:
  - Rubber-hose cryptanalysis of mix operators; as a defense, use mix servers in different countries
  - Adversary operates all of the mixes; as a defense, has lots of mix servers.
  - Adversary observes when Alice sends and when Bob receives and links the two together.
  - Side channel attack exploits timing information; as a defense, pad messages or introduce significant delays. (Tor does the former, but note that it is not enough for defense.)
- Issues include
  - impaired performance (i.e., messages may bounce around a lot) and
  - traffic leakage (suppose all of your HTTP/HTTPS traffic goes through Tor, but the rest of your traffic does not).

Concerning the traffic leakage problem, Tor's solution is to inspect the logs of their DNS server to see who looked up sensitive.com just before your connection to their web server arrived. The hard, general problem is that anonymity is often at risk when an adversary can correlate separate sources of information.

To summarize, Tor hides metadata (i.e., the "what," or message content) via TLS/PGP/OTR/Signal, and also hides the "who" via the onion routing protocol. It also provides a messaging system called Pond (Figure 14.14), which hides the "when" and "how much" parts as illustrated. Note that Pond is not an email; rather, it is a forward, secure, asynchronous messaging system. Pond seeks to protect the possibility of leaking traffic info against all but a global passive adversary (i.e., forward secure, no spam allowed, and messages expire automatically after a week).

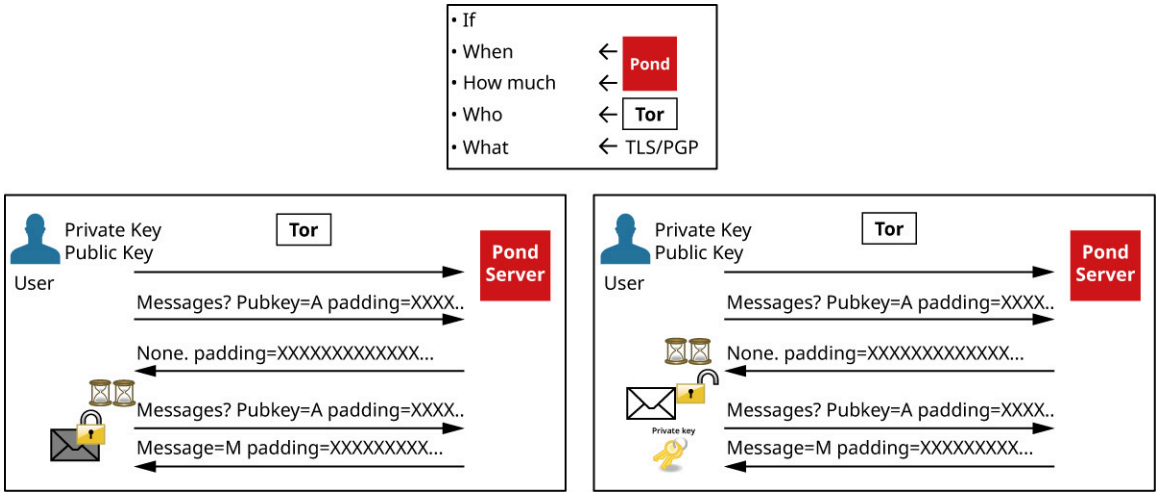


Figure 14.14 Pond is a forward, secure, asynchronous messaging system that seeks to protect against leaking traffic information. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

Zero-Knowledge Proofs

A **zero-knowledge proof (ZKP)** is a cryptographic system that functions as a useful tool to protect privacy. With ZKPs, users rely on cryptographic algorithms to verify information without accessing the supporting data. Digital signatures based on PKI (i.e., RSA-algorithm or ECC) are examples of ZKPs. In these cases, the person holding the private key can convince any public key holder that they know the private key without revealing it.

Properties of ZKPs include:

- Completeness: A statement is true if an honest prover can convince an honest verifier.
- Soundness: If the prover is dishonest, he cannot fool the verifier.
- Zero-knowledge: No private information is revealed to the verifier.

To understand ZKPs, it is important to know about the PCP theorem, which states that every decision problem in the NP complexity class has probabilistically checkable proofs of constant query complexity and logarithmic randomness complexity.

For an error probability of 2-k, 3k bit positions must be verified. (For example, the Soundness error probability of 2-40 ≈ 10-12 with 120-bit verification.)

Computational integrity proofs are probabilistic proof systems based on the PCP theorem, allowing a prover to convince a verifier of the correctness of an arbitrary computation with an exponentially faster efficiency than naive checking of the computation. Computational integrity proofs exhibit the following properties:

- Completeness: A statement is true if an honest prover can convince an honest verifier.
- Soundness: If the prover is dishonest, they can't fool the verifier.
- Succinctness: There is exponentially faster verification than naive checking of the computation.
- Zero-knowledge (bonus): No information is revealed to the verifier.

ZKPs rely on a transcript of the original computation expanded into a proof using an error-correcting code (e.g., a Reed-Solomon code, polynomial commitment), which spreads any errors within the proof. A low number of (e.g., three) of stochastic queries by the verifier is sufficient to prove the correctness of the computation with high probability (Figure 14.15).

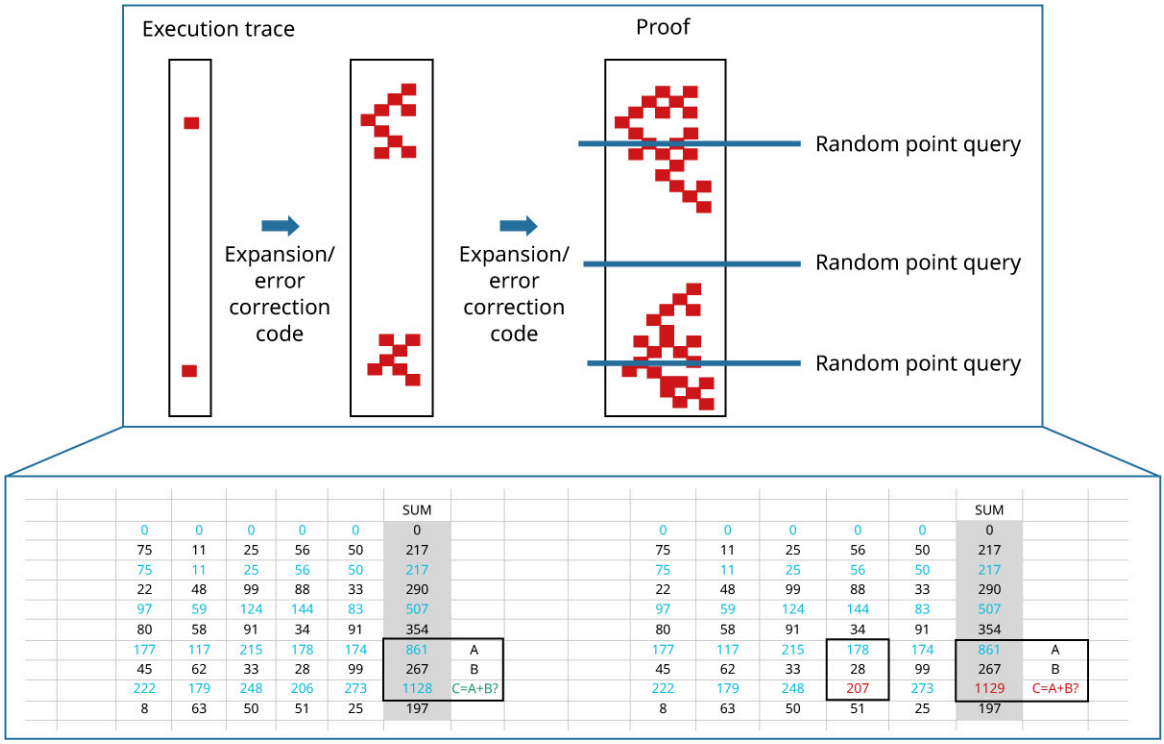


Figure 14.15 ZKPs can be used to prove the correctness of computations. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

The zk-SNARK acronym stands for zero-knowledge succinct non-interactive argument of knowledge. A zk-SNARK is cryptographic proof that allows one party to prove it possesses certain information without revealing it. The proof is made possible using a secret key created before the transaction occurs. zk-SNARK is used as part of the cryptocurrency Zcash protocol. In a noninteractive proof, the interactive dialog between the prover and verified is replaced by randomness. Query locations are predefined by randomness, which the prover cannot influence.

zk-SNARKs cannot be applied directly to computational problems. Problems need to be first converted into the right "form." The form is called a quadratic arithmetic program (QAP), and transforming the code of a function into a QAP is highly nontrivial. It requires turning the computation into an algebraic circuit and transforming it into a rank-1constraint system (R1CS) that can then be converted into a QAP. In addition to the process for converting the code of a function into a QAP, another process may run alongside in such a way that if you have an input to the code, you can create a corresponding solution (sometimes called "witness" to the QAP). Once this is done, another fairly intricate process must be followed to create the actual "zero-knowledge proof" for the witness and a separate process for verifying proof that someone else passes along to you. The full machinery between zk-SNARKs is illustrated in Figure 14.16.

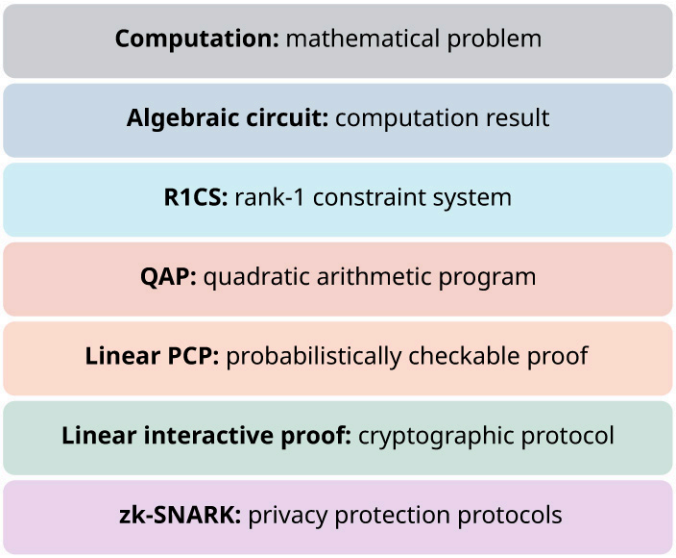


Figure 14.16 zk-SNARK's full machinery includes computation, algebraic circuit, R1CS, QAP, linear PCP, and linear interactive proof. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

zk-SNARKS are not transparent and require a so-called trusted setup to secure the randomness of the parameters determining the selection of queries in the verification procedure. The random parameters are, for example, generated using a hash function. However, the knowledge of the original values fed into the hash function ("toxic waste") must be kept secret. Otherwise, retrieving the random parameters and manipulating the selection of verification queries would be possible. A trusted setup is a multi-party computation (MPC; "ceremony"), which recursively generates random parameters. If at least one party is honest and forgets its input values, the randomness of the trusted setup is safe. Universal trusted setups can be generated once and can be reused for other applications as well; nonuniversal ones are tied to the specific circuit. The difficult part is to generate a zk-SNARK with reasonable resources for real-world applications.

Zero-knowledge scalable transparent argument of knowledge (zk-STARKs) is a type of ZKP where one party can prove to another that a given statement is true without revealing any other information other than the fact that the statement is true. Attributes of the zk-STARK concept are as follows:

- zero-knowledge (refers to privacy preservation)
- scalability (indicates that verification time is substantially less than the time taken for naive computations)
- transparency (reflects the lack of a trusted setup requirement)
- argument and knowledge (related to the security and robustness of the cryptographic scheme).

Zero-knowledge STARKs work by leveraging leaner cryptography, specifically collision-resistant hash functions, to validate the truth of a statement without sharing the details behind it. Unlike zk-SNARKs (Zero-knowledge succinct non-interactive argument of knowledge), which rely on an initial trusted setup and are theoretically vulnerable to quantum computer attacks, zk-STARKs eliminate these issues. That said, it is important to note that this leaner approach results in a significant disadvantage. Specifically, zk-STARKs generate proofs that are typically 10 to 100 times larger than those created by zk-SNARKs, thus making them more expensive and potentially less practical for certain applications.

The trade-offs between different properties of different noninteractive and transparent proof systems amount to a difference in verification time (between 2 ms to 250 ms), prover time (1 s to 100 s), and proof size (between 200 B to 250 kB).

The cryptographic primitives used by ZKPs include the following cryptographic primitives:

- Collision-resistant hash function (quantum secure): STARK, Fractal, Aurora
- Elliptic curve cryptography: Bulletproofs, Halo

- Knowledge of exponent/pairing groups: Groth16, Sonic, Marlin, PLONK
- Groups of unknown order: Supersonic
- Lattice-based cryptography (quantum secure): under development

There are myriads of recent applications of ZKPs to blockchain technology for privacy and scalability improvements. A related topic is verifiable delay functions (VDFs), which emerged in June 2018. A verifiable delay function (VDF) is a function f : X → Y that takes a prescribed minimum time to compute (even on a parallel computer). However, once computed, anyone can quickly verify the output. They can prevent fraud or frontrunning on exchanges, online auctions, games, or prediction markets. Another related topic is multi-party computations (MPCs), which are methods for parties to jointly compute a function over their inputs while keeping those inputs private. MPCs are different from traditional cryptographic tasks that use cryptography to ensure the security and integrity of communication or storage and assume that the adversary is outside the system of participants. In MPC, cryptography protects participants' privacy from each other. An example of an application is that of a trusted setup ceremony, which is a secure MPC.

An important related topic is fully homomorphic encryption (FHE), the "holy grail of cryptography." FHE allows arbitrary mathematical operations on encrypted data (i.e., for every *f*: $y = f(x)$ -> Encrypted $y' = f(x')$).

Open Problems in Cryptography

The ongoing attacks and the need to defend crypto schemes require staying on top of best practices. Ideally, developers should write code that can be changed easily. Also, remember not to develop your own cryptographic mechanisms. Go through peer review and apply Kerckhoff's principle. Do not even implement the underlying crypto, and do not misuse existing crypto.

Information about a particular implementation could leak (e.g., power consumption, electromagnetic radiation, timing, errors). Attacks based on this are referred to as "side-channel attacks." As an example, simple power analysis (SPA) may be used to interpret power traces during a cryptographic operation. Simple power analysis (Figure 14.17) can reveal the sequence of instructions that have been executed.

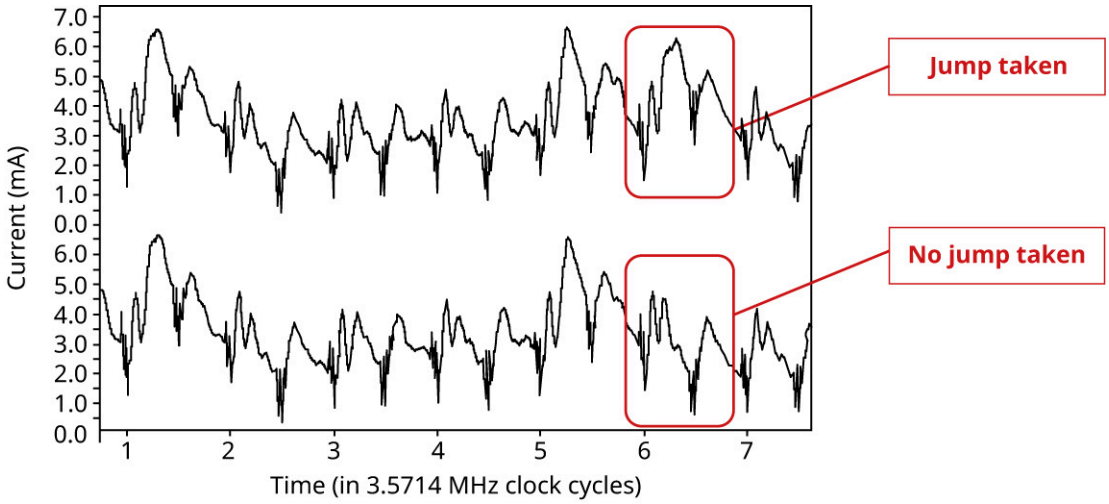


Figure 14.17 Simple power analysis can reveal specific instructions such as those shown here. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

Assuming the program execution path depends on the inputs (i.e., key/data), SPA can be used to reveal the keys. Different from SPA, which visually inspects a single run, differential power analysis (DPA) can operate interactively and reactively across multiple samples. Using this approach, DPA can produce new plain text messages that can be passed as inputs repeatedly.

In order to counter these types of attacks, it is necessary to hide information by making sure that the execution paths do not depend heavily on the inputs. This may require dropping optimizations that depend on

specific bit values in keys. In the past, Chinese remainder theorem (CRT) optimizations allowed remote timed attacks on Secure Socket Layer (SSL) servers. In general, cryptosystems should be designed to resist information leaks.

A different type of side-channel attack happens when keys are safely stored in memory, and attackers do not have access to a machine. In that case, if the attacker can access the physical machine and reboot it into an OS they control, it becomes possible for the attacker to look at the memory contents. While memory loses its state without content, it does so much more slowly at very cold temperatures. Therefore, an attacker could cool down the memory, shut down the machine, move the memory to a different machine, and boot it into a different OS. All that is left to do then is to scan the memory image for keys, which is difficult but feasible, assuming the keys have a format that is easy to detect. A couple of techniques can be used to counter these types of attacks. One solution is to encrypt all the memory, which requires additional CPU power. Another solution, which is used on Xbox, requires a trusted platform module (TPM) to store hardware keys, making installing them very difficult. Some TPM self-destruct when tampered with or keep keys in memory for a limited time (e.g., the keys are removed from memory when going to sleep mode).

There are new mechanisms to permit new types of interactions. A style of interaction that has been getting a lot of attention is the following:

- Alice has proprietary data.
- Bob has proprietary code (or computational resources).
- The goal is for Bob to run his code on Alice's data without learning her input or the output.

There are problems introduced earlier that still require usable solutions:

- Secure multiparty computation: For example, Alice and Bob both have data and want to know the output of a function over their private data without having to reveal their data to each other (e.g., "which of us has more money" without having to reveal exactly how much either has).[20] Communication overhead and vulnerability to attacks from colluding parties are the main challenges. While there are techniques to solve these problems, they usually come with higher computational costs.
- Fully homomorphic encryption: Homomorphic encryption (HE) can perform computations on encrypted data without first decrypting it with a secret key. It then encrypts the computation results, and only the owner or the private key can decrypt them.[21] Partial HE systems have been around since the 1970s, and a fully HE scheme that makes it possible to apply mathematical operations to encrypted data was first developed by Craig Gentry in 2009. However, fully homomorphic encryption in its current form is impractically slow.

Traditional Software Solutions Security

To protect systems, software solutions architects and software developers must consider security as a property of the systems they build. The way software safeguards system resources, including data, to provide access to only authorized users is through **software security**. Security should be part of the software design process to take a proactive approach to cyber threats and risks. This includes prioritizing security in software requirements, programming, testing, implementation, and maintenance. Practices that should be part of software security include threat modeling and vulnerability management.

Generally, to implement security in the software development process, software architects and developers should follow these steps:

- Make software security a priority and focus of the development process.
- Identify security risks that should be addressed with software.

20 For a recent assessment of the state of the practice of this technology, see https://research.aimultiple.com/secure-multi-party-computation/
21 For a recent assessment of the state of the practice of this technology, see https://research.aimultiple.com/homomorphic-encryption/

- Identify vulnerabilities.
- Use the appropriate standards, best practices, and frameworks to guide the development process.
- Review and analyze the code extensively with an emphasis on cybersecurity.
- Implement penetration testing, which includes the following steps:
  - Planning process to gather information about the system and define testing goals.
  - Scan the system with tools to learn how the system responds to threats.
  - Execute attacks and make every effort to gain access to the system, revealing its weaknesses.
  - If accessed is gain, make every effort to maintain that access without detection.
  - Analyze test results and make system changes before testing again.
  - Repeat this process over and over to identify system weaknesses.

Software Security

Software solutions architects and developers must consider security as a property of the systems they build. Many attacks begin by exploiting a vulnerability. In this case, a vulnerability is a software defect that yields an undesired behavior (i.e., the code does not behave correctly). Software defects arise due to flaws in the design or bugs in the implementation. Unfortunately, software can't be completely bug-free, and fixing every known bug may be too expensive. In general, the focus is to fix what is likely to affect normal users and not focus on bugs that normal users never see or avoid. Because attackers are not normal users, they look for bugs and flaws and try to exploit them. Therefore, to achieve software security, it is necessary to eliminate bugs and design flaws and/or make them harder to exploit. Doing so requires thinking like attackers and developing a foundation for deeply understanding the systems built and used.

Most (interesting) software takes inputs from various sources, and any of these inputs may be malicious, such as the following:

- direct user interaction (e.g., user interfaces with software via a command line interface or opens a document)
- third-party libraries that are linked to the software
- future code updates

Securing software in this context should result in correct operation despite malicious inputs. In order to study how to secure software, we will focus on what should be done to secure software written using the C programming language and investigate program control flow hijacking via buffer overflows, code injection, and other memory safety vulnerabilities. This is motivated by the fact that the C language is consistently used widely, and many mission-critical systems are written in C (e.g., most operating systems kernels such as Linux, high-performance servers such as Microsoft SQL server, many embedded systems such as Mars rover). Furthermore, the same techniques apply more broadly.

Buffer Overflow Attacks and Defenses

Many buffer overflow attacks were perpetrated over the years against software programs written in C. Buffer overflows are prevalent and constitute a significant percentage of all vulnerabilities.[22] In 1988 for example, Robert Morris sent a special string via a buffer overflow attack to the fingered daemon on a computer running the VAX operating system and caused it to execute code that created a worm copy that propagated over the network and affected Sun machines running the BSD operating systems. This resulted in $100M worth of damages as well as probation and community services. Robert Morris subsequently became a professor at MIT. In 2001, the Code Red worm leveraged a buffer overflow error in the MS-IIS server. As a result, 300,000 machines were infected within 14 hours. In 2003, the SQL Slammer worm leveraged a similar buffer overflow error in the MS-SQL server. As a result, 75,000 machines were infected within ten minutes. In 2008–2009, the Conficker worm exploited a buffer overflow in Windows RPC, which infected more than ten million machines. In 2009–2010, Stuxnet exploited several buffer overflows in the Windows print spooler service, LNK shortcut display, task scheduler, and the same RPC buffer overflow as Conficker, which led to legitimate cyber warfare. Between 2010 and 2012, Flame exploited the same print spooler and LNK buffer overflows as Stuxnet, which resulted in a cyber-espionage virus. On January 8, 2014, a 23-year-old discovered an X11 server security stack buffer overflow vulnerability (i.e., scanf used when loading early 1990s BDF bitmap fonts) from 1991. The GHOST glibc vulnerability was introduced in 2000 but was only discovered many years later. One last example is the Syslog logging infrastructure daemon bug in macOS and iOS, which relied on the fact that running programs would issue log messages and Syslog would handle storing and disseminating them. The problem was that Syslog used a buffer to propagate these messages, which was not large enough and would sometimes write beyond the end of the buffer.

Based on this, understanding how C programs buffer overflow attacks work and how to defend against them is critical and requires knowledge (refer to Chapter 4 Linguistic Realization of Algorithms: Low-Level Programming Languages and Chapter 5 Hardware Realizations of Algorithms: Computer Systems Design) of the C software compiler, the operating system on which the program is run, and the computer system architecture on which the operating system runs—in other words, a whole-systems view. As a refresher, the stack layout on a 32-bit (Intel IA32) computer when calling a sample C function (Figure 14.18) is shown. Note that in this case, there are two 4-B values between the arguments and the local variables.

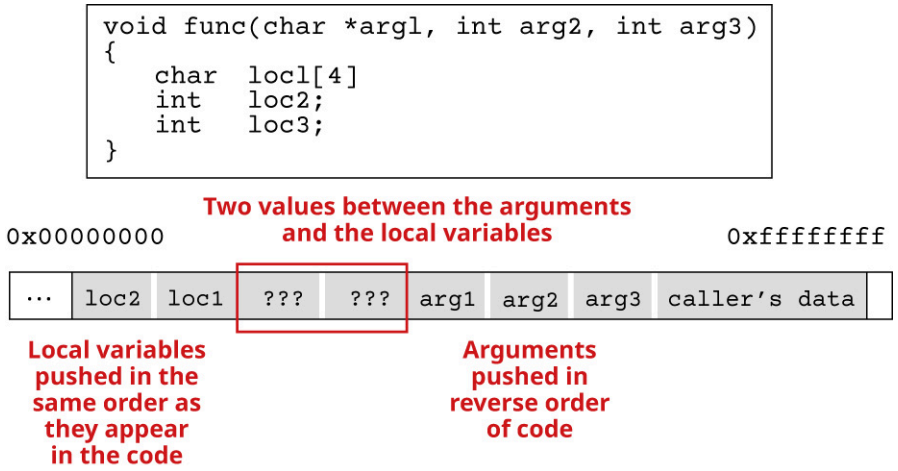


Figure 14.18 Local function variables are pushed on the stack in the order they appear in the code, while function arguments are pushed in the reverse order or their appearance in the code. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

The function func can access variable loc2, using the stack frame pointer %ebp (Figure 14.19). Note that the same would apply on a 64-bit computer by using a 64-bit memory layout and changing the names of the

22 Search for "buffer overflow" in the national vulnerability database at https://nvd.nist.gov/vuln/search?adv_search=true&cves=on&cwe_id=CWE-119 and look for MITRE's top-25 most dangerous software errors for 2011 as an example at https://cwe.mitre.org/top25/

registers accordingly (e.g., %rbp vs. %ebp).

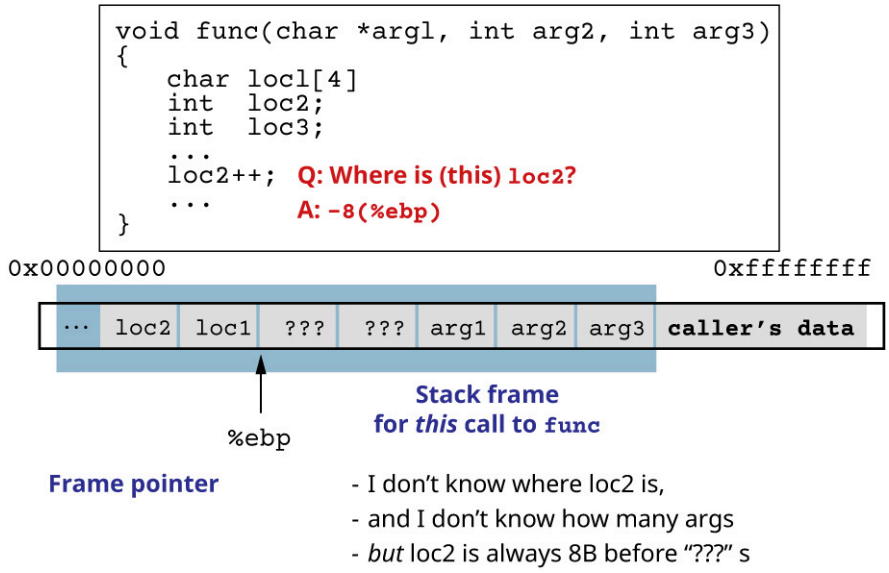


Figure 14.19 The location of the loc2 variable in the stack frame is always 8 B before the address contained in %ebp. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

Figure 14.20 illustrates how to properly return from a call to the function `func`.

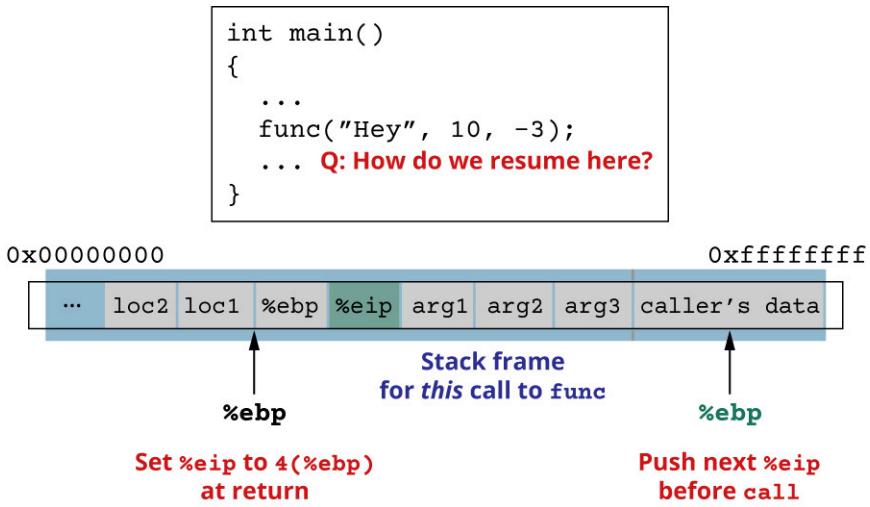


Figure 14.20 This figure shows how to properly return from a call to the function `func`. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

In summary, the steps to follow in order to call and return from a function are as follows:

Calling function:

1. Push arguments onto the stack (in reverse).
2. Push return address onto the stack (i.e., the address of the instruction that needs to be run once control returns to the calling program: %eip + something).
3. Jump to the function's address.
   Called function:
4. Push old frame pointer onto the stack: %ebp.
5. Set frame pointer %ebp to where the end of the stack is right at this time: %esp.
6. Push local variables onto the stack; access them as offsets from %ebp.
   Returning function:
7. Reset previous stack frame: %ebp = (%ebp) /* copy it off first */.

8. Jump back to return address: %eip = 4(%ebp) /* use the copy */.

Let us investigate a buffer overflow example based on understanding the stack memory layout when calling and returning from functions. Buffers are commonly used in C to store sets of values of a given data type. For example, strings are buffers of characters in C. A buffer overflow occurs when more values are put into the buffer than it can hold. Let us consider this buffer overflow example (Figure 14.21) and right before the "strcpy(buffer, arg1);" statement is executed in the function `func`.

```
void func(char *vall)
{
    char buffer[4];
    strcpy(buffer, val1);
    ...
}


int main()
{
    char *mytxt = "Authme!";
    func(mytxt);
    ...
}
```



Figure 14.21 The strcpy statement executed in the function `func` overflows the size of the buffer specified. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

Once the "strcpy(buffer, arg1);" statement is executed in the function func (Figure 14.22), it overwrites the stack memory location were %ebp was stored, which causes a segmentation violation when the function returns.

```
void func(char *val1)
{
    char buffer[4];
    strcpy(buffer, val1);
    :
}

int main()
{
    char *mytxt = "AuthMe!";
    func(mytxt);
    .

}
```

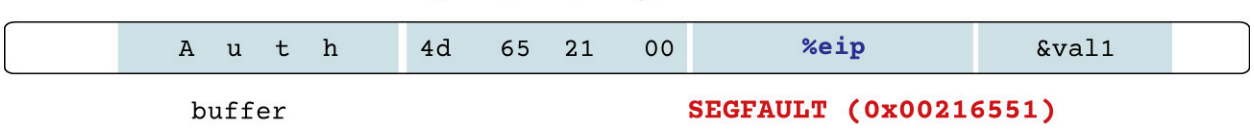


Figure 14.22 The strcpy statement executed in the function `func` sets %ebp to the wrong address upon return due to the buffer overflow. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from

Dave Levin)

Using the GNU debugger (i.e., gdb) is useful to debug programs that run into buffer overflows. It enables users to show information about the current frame and registers, examine bytes of memory starting at a given address, set a breakpoint at a given function address, and step through a call to it.

A safe version of the function `func` will never cause a buffer overflow. It will simply limit the number of characters read from the command line and could be easily adapted to replace the function `func`:

```
Void nooverflow() {
  char buflimit[100];
  fgets (buflimit,
  sizeof(buflimit));
}
```

Note that strcpy lets you write as many characters as you want until it reads and end-of-string character (i.e., null character "\0" in a C string); therefore the problem could get worse than just overwriting %ebp. Figure 14.23 illustrates a different type of buffer overflow that would occur if the function `func` were to execute the code provided. In that case, the input writes from low to high addresses.

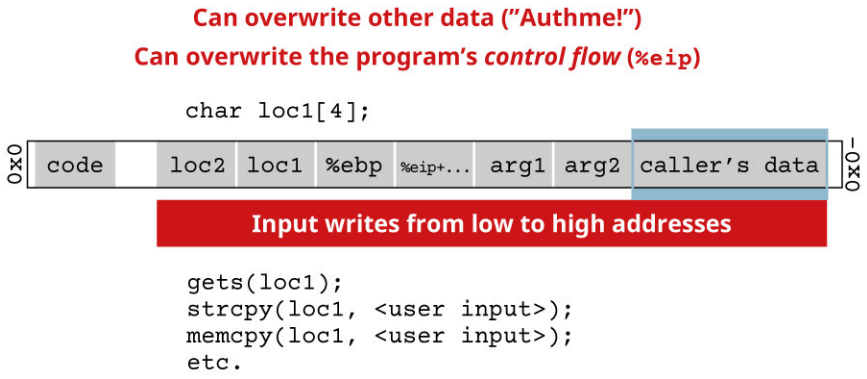


Figure 14.23 The execution of the code shown is such that the input writes from low to high addresses. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

Code Injection Attacks and Defenses

The example of buffer overflow shown earlier uses a string provided by the program itself but, in general, inputs could come from different sources (e.g., text input, network packets, environment variables, file inputs). Therefore, the existence of a buffer overflow bug in a program can lead to a code injection attack (Figure 14.24).

```
void func(char *val1)
{
    char buffer[4];
    sprintf(buffer, val1);
    ...
}
```

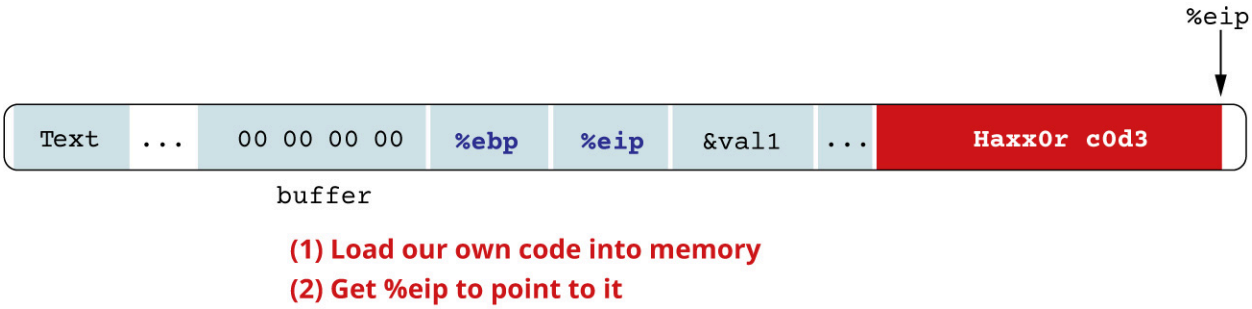


Figure 14.24 A code injection attack can be staged by loading code into memory and point %eip to it. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

Pulling off this type of attack requires overcoming a few challenges, which we will discuss in detail. In general, making it really hard to overcome the various challenges is key to defending programs against code injection attacks. The challenges are as follows:

1. Loading the code into memory:
   The code that is loaded into memory must be machine code that is ready to run. It should not contain any all-zero bytes. Alternatively, some library functions (e.g., sprintf, gets, scanf) will stop copying. The loader cannot be used because the code is injected. Finally, the code injected cannot use the stack because it is designed to smash it.
   The best type of code for this is full-purpose shellcode that can be launched as a shell (Figure 14.25). There are many examples of such code, and there is a lot of competition to write the smallest amount of code. Also, a way to ensure that the injected code will work most effectively is to attempt privilege escalation and go from guest (or nonuser) to root.

```
#include <stdio.h>
int main()
{
char *mytext(2);
mytext[0] = “/bin/sh”;
mytext[1] = NULL;
execve(”/bin/sh”, mytext,
NULL);
return 0;
}
```

```
xorl %eax, %eax
pushl %eax
pushl $0x68732f2f
pushl $0x6e69622f
movl %esp,%ebx
pushl %eax
...
```

Assembler

```
“\x31\xc0”
“\x50”
“\x68””//sh”
“\x68””/bin”
“\x89\xe3”
“\x50”
...
```

Machine code

Figure 14.25 The best type of code for code injection is full-purpose shellcode that can be launched as a shell. (attribution:

Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

Concerning privileged escalation, the idea is to exploit knowledge of permissions on the targeted operating system. In the case of Linux, files have read/write/execute permissions owner, group, and others. Permissions are defined for userid and groupid, and the root userid is p. The command passwd may be used as part of an attack by making it possible for any user to execute that command rather than just its owner (i.e., root). The idea is to have a root-owned process run setuid(0) or seteuid(0) in order to get root permissions. While root owns "passwd," users can run it, and getuid() will return the userid of the person who ran it. Executingseteuid(0) next will set the effective userid to root, which is allowed because root is the process owner.

2. Getting injected code to run:
   Because it is only possible to write forward into a memory buffer, the running code must already be used to jump to the injected code. The typical approach is to hijack the saved %eip and change it to point to the address of the injected code (Figure 14.26).

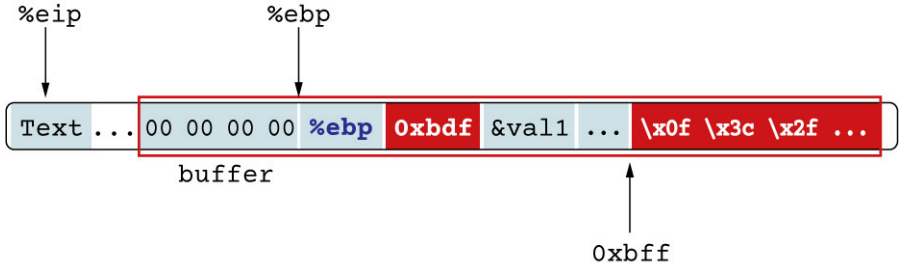


Figure 14.26 Code injection is performed by hijacking the saved %eip and changing it to point to the address of the injected code. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

However, getting to know the address of the save %eip is a challenge. Furthermore if the %eip is wrong and points to data, the CPU will panic when it attempts to execute an invalid instruction.

3. Finding the return address:
   Because the code cannot be accessed, there is no way to know where the buffer starts based on the saved %ebp. One possibility is to try a lot of different values, and the worst-case situation for a 64-bit memory space involves computing 2^{64} possible answers. If address space layout randomization (ASLR) is disabled, which you cannot count on today, the stack always starts from the same fixed address and then grows. Still, it does not usually grow very deeply unless the code is heavily recursive.
   Another approach consists of using nop sleds. Because nop is a single-byte instruction, it makes it possible to move to the next instruction, thereby improving chances to hit the address of %eip (Figure 14.27).

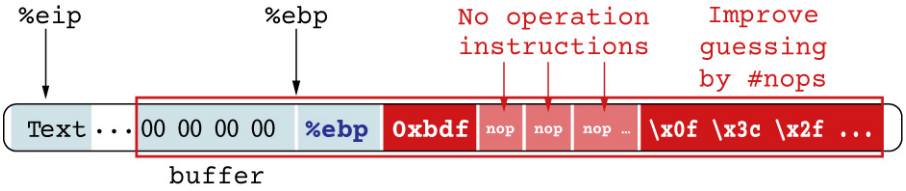


Figure 14.27 Introducing single-byte nop instructions improves the chances to hit the address of %eip. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

Finally, putting it all together, the recipe for code injection is to achieve that shown in Figure 14.28.

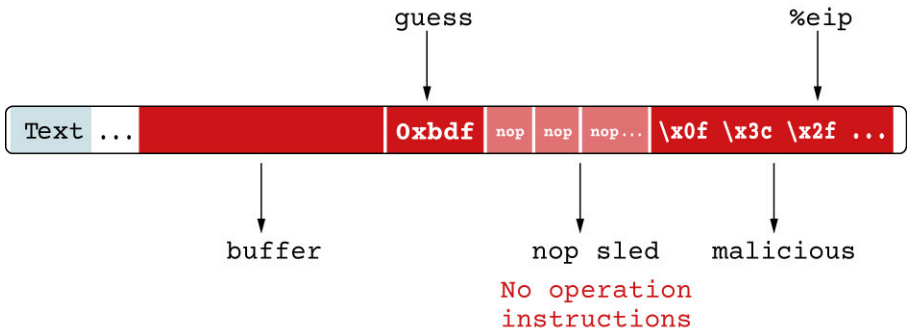


Figure 14.28 The recipe for code injection is to guess the stack return address and use nop sleds to improve the chances to hit the address of %eip. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

A typical way to protect a program against code injection is to prevent data execution by marking memory pages as nonexecutable or make it impossible to put code into the memory by detecting overflows with canaries. Using a canary amounts to placing a known string of characters in memory at the end of the buffer and aborts the program execution if the expected value at that location is changed (Figure 14.29).

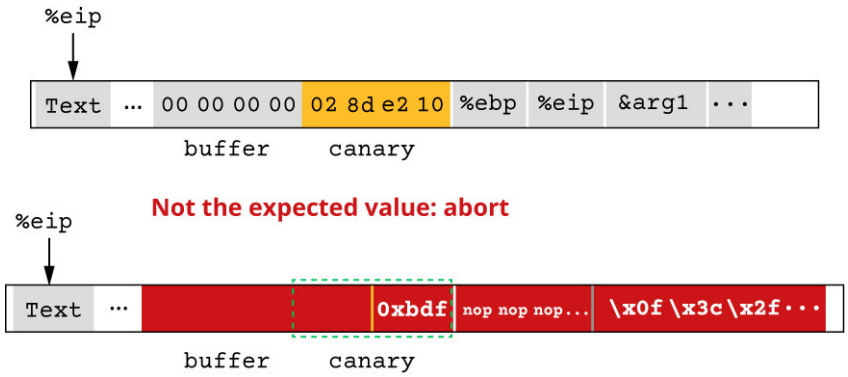


Figure 14.29 Placing a canary at the end of the buffer makes it possible to detect whether the content at the end of the buffer has been changed. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

There are a few possibilities for canary values as indicated:

1. Terminator canaries (e.g., CR, LF, NULL, -1) leverage that scanf and other similar functions do not allow these values.
2. Random canaries write a new random value when each process starts and saves the real value somewhere in memory; it is necessary to write-protect the stored value in that case.
3. Random XOR canaries work the same way as random canaries but store “canary XOR <some control info> instead.

Integer Overflow Attacks and Defenses

Programmers have a tendency to think about integers as mathematical integers. It is, therefore, easy for them to write C code that causes integer overflows (e.g., assigning larger types to smaller types, arithmetic overflow). For example, the following multiplication using integers represented by the two complement notations causes an arithmetic overflow that yields a negative result: 15000000*500 = -1089934592. Knowing this, attackers may simply control the value of an integer and cause software to behave unexpectedly. Defending against integer overflow requires using appropriate types (e.g., using sizet in the C language).

Format String Vulnerability and Defenses

A format function is a special kind of ANSI C function used as a conversion function to represent primitive C data types as human-readable strings. Format functions are used in most C programs to output information, print error messages, and process strings. A format string vulnerability occurs when an attacker can provide

the format string to an ANSI C format function in part or as a whole. If the attacker can do so, the behavior of the format function is changed, and the attacker may get control over the target application.

For example, by calling the following function using a command line parameter:

```
int func (char *user) {
    printf (user);
}
```

an attacker can get control over the entire ASCII string of the printf function (i.e., the part that contains text and format parameters). To avoid this problem, the function should be written as follows:

```
int func (char *user) {
    printf ("%s", user);
}
```

This kind of vulnerability is more dangerous than the common buffer overflow vulnerability.[23]

Heap Control Data Vulnerability

The heap is managed by the malloc() function, which requests pages of memory from the operating system, manages free chunks, and allocates memory for programs. Attackers can use the malloc function to overwrite heap metadata and abuse it to exploit buffer overflows by injecting control data in malloc space.[24]

Code Reuse Attacks and Return-Oriented Programming (ROP)

We have discussed earlier ways to prevent an attacker from executing any injected code using canaries. While attackers may attempt to bypass stack canaries last time, they may also simply focus on bypassing data execution prevention (DEP) measures by executing existing code such as the program code itself, dynamic libraries, or libc. In particular, libc contains valuable functions such as system (runs a shell command) or protect (changes the memory protection on a region of code). Rather than returning to shellcode, an attacker may decide to return to a standard library function like system and cause a system to crash to exit. Another alternative is to return to protect, inject code, and make it executable. An attacker may also chain two functions together. In fact, attackers may not need to limit themselves to functions and cleanup code. An alternative is to encode arbitrary computation, including conditionals and loops, by returning to sequences of code ending in `ret`. This last approach is referred to as return-oriented programming (ROP)[25] and relies on the following steps:

1. Disassemble code (i.e., library or program).
2. Identify useful code sequences (e.g., code sequences usually ending in `ret`).
3. Assemble useful sequences into reusable gadgets.
4. Assemble gadgets into desired shellcode.

Time of Check/Time of Use Problem

Figure 14.30 illustrates the time-of-check to time-of-use (TOCTOU) problem.

23 See https://julianor.tripod.com/bc/formatstring-1.2.pdf for more details.
24 See https://web.archive.org/web/20220911001330/http://www.phrack.org/issues/57/8.html for more details on this vulnerability.
25 See https://hovav.net/ucsd/dist/geometry.pdf for more details on ROP.

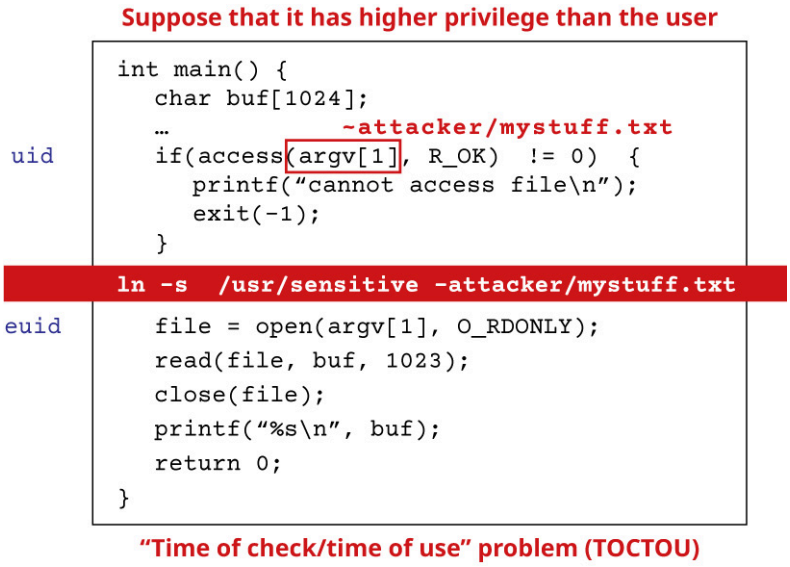


Figure 14.30 Access is intended to check whether the real user who executed the setuid program would normally be allowed to read the file, so access checks the real uid instead of the effective userid euid. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

The code should be modified as in Figure 14.31 to avoid the TOCTOU problem.

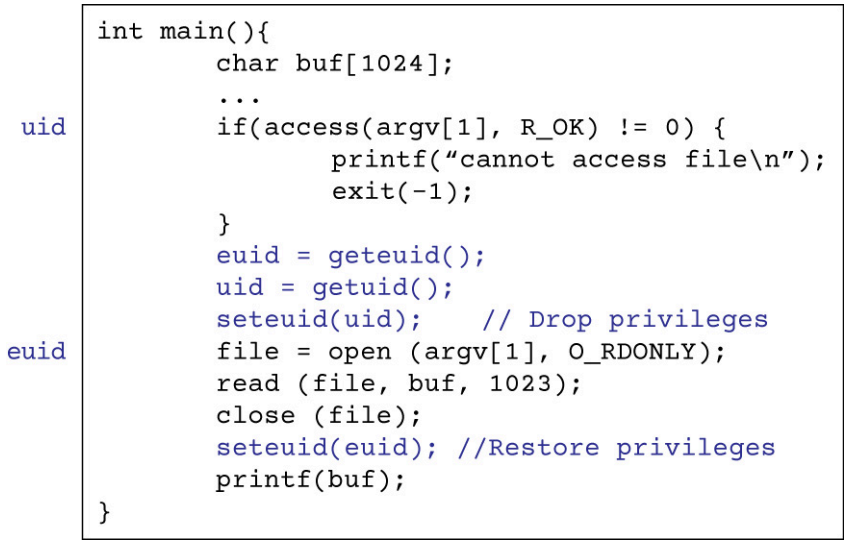


Figure 14.31 The code is modified to switch the user (uid) that executed the setuid command to the effective user (euid) to ensure that proper permissions are used when accessing the document. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

Playing Cat and Mouse to Secure Software

The following illustrates how securing software is like playing cat and mouse:

- Defense: Make stack or heap nonexecutable to prevent code injection.
- Attack response: Return to libc.
- Defense: Hide the address of the desired libc code or return the address using ASLR.
- Attack response: Perform a brute force search (for 32- and 64-bit systems) or information leak (i.e., format string vulnerability)
- Defense: Avoid using libc code entirely and use code in the program text instead.
- Attack response: Construct needed functionality using return-oriented programming (ROP).

Common Cyber Threat Defenses

Common cyber threats were introduced earlier along with various examples of malware. Protection against malware involves the use of an intrusion detection system (IDS) as well as an intrusion prevention system (IPS). An IDS may be host- or network-based (i.e., HIDS or NIDS). In this case, detection happens after the attack (i.e. the memory is already corrupted due to a buffer overflow attack). A preventive measure must stop the attack before it reaches the system (i.e., the shield does packet filtering). Some tools support both IDS and IPS (e.g., Snort).

Malware Detection Methods

In general, some types of malware may rely on some delay based on a trigger to run (e.g., time bomb, logic bomb), and they may include a backdoor to serve as ransom. Other types of malware piggyback on other pieces of code. For example, viruses run when users initiate a task (e.g., run a program, open an attachment, boot the machine). Worms run while another program is running and do not require user intervention. Therefore, a virus propagates by ensuring it is eventually executed, assuming user intervention. Once executed, the virus creates a new separate instance of itself and typically infects by altering stored code. A worm self-propagates by making sure it is immediately executed without user intervention. Once executed, the worm creates a new separate instance of itself, and it typically infects by altering the running code. There is a fine line between viruses and worms; some malware uses both types.

Detecting self-propagating malware (e.g., viruses or worms) is challenging. While antivirus software attempts to detect viruses, virus writers strive to evade human response and avoid detection for as long as possible. In the case of worms, the virus writer wants to spread the worm and hit many machines as quickly as possible to outpace the human response. Viruses have been around since the 1970s. They are opportunistic and eventually run as a result of a user action. Two orthogonal aspects define a virus: the way it propagates and what it does (i.e., the “payload”). A general infection strategy consists of altering existing code to incorporate the virus, share it, and expect users to (unwittingly) re-share it. Viruses infect other programs by taking over their entry point so the virus is run when executing these programs. They infect documents, boot sectors, or run as memory resident code. They increase their chances of running by attaching malicious code to a program a user is likely to run (e.g., email attachments). Once viruses run, they also look for an opportunity to infect other systems (e.g., proactive creation of emails).

An obvious method for detecting viruses is to use signature-based detection, which consists of looking for bytes corresponding to injected virus code and protecting other systems by installing a recognizer for a known virus within them. This approach requires fast scanning algorithms and has resulted in creating a multi-billion-dollar antivirus market. Adding recognized signatures to that market enables marketing and leads to competition. To combat this detection method, virus writers give viruses harder signatures to match by creating polymorphic viruses. In this case, the virus generates a semantically different version of the code every time it propagates. While the higher-level semantics of the virus remain the same, the actual execution code differs (e.g., machine code instructions are different, different algorithms are used to achieve the same purpose, the code makes use of different registers, or different constants are used). This can be accomplished by including a code rewriter with a virus or adding some complex code that never runs to evade detection attempts. Instead of appending the program to the virus, virus writers surround the program with virus code, or overwrite uncommonly used parts of the program in order to confuse virus scanners. They also change the virus code so scanners cannot pin down a signature. Code changes can be mechanized so that the code looks different every time it is injected. To do so, they use public key encryption (and the fact that it is nondeterministic) to generate different virus code each time they encrypt the virus. At the same time, decryption always produces the same virus code (Figure 14.32). Virus writers also iteratively obfuscate the code (i.e., encrypt + jmp + ...) using different encryption algorithms until the obfuscated code is fully undetectable.

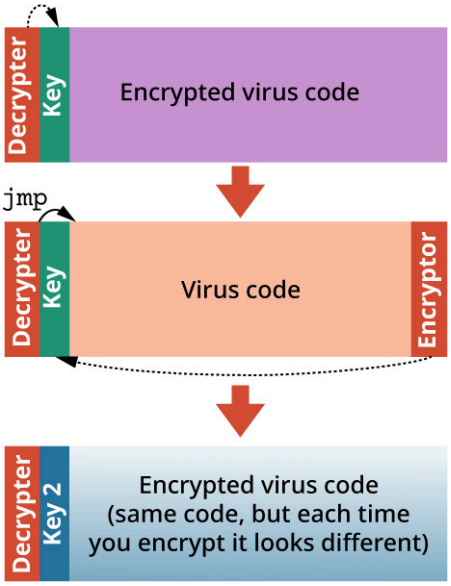


Figure 14.32 Virus writers generate a different virus code each time they encrypt the virus, while decryption always produces the same virus code. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

Scanning is insufficient to detect metamorphic viruses, and proper detection requires analysis of the code execution behavior. Two general approaches can be applied to facilitate detection, and both need to be conducted in a safe environment (e.g., gdb or a virtual machine). One of the approaches used by antivirus companies focuses on analyzing a new virus to locate its behavioral signature. Another approach focuses on suspicious code analysis to see if it matches the signature. In general, attackers have the upper hand because antivirus systems share signatures, which provides insight that attackers may use to react. Attackers change viruses slowly to make it hard to create a matching behavioral signature or they can start acting differently to avoid detection. In order to detect polymorphic viruses, antivirus writers can record narrow signatures used by virus writers to catch the associated decrypters. Because these signatures are often very small, this approach can result in many false positives. To counter this approach, attackers may spread small decrypter code around and use a jmp instruction to get to the virus code. Another approach to detecting polymorphic viruses is executing or statically analyzing the suspicious code to see if it decrypts. The issue with this last approach is that it is hard to differentiate an encrypted virus from a valid common “packers” program that does something similar (e.g., decompression). It also depends on how long the code can be executed without any side effects. Virus writers can combat these approaches by changing the decrypter. For example, oligomorphic viruses change from one of a fixed set of decrypters, and true polymorphic viruses can generate an endless number of decrypters (e.g., brute force key break). While this approach leads to inefficiencies, it makes it extremely difficult for antivirus software to detect viruses.

Today, malware detection is a technological arms race between detection and avoidance. Initially, only a few very clever people were capable of creating viruses. Viruses are now commoditized, and anyone can launch one. The creation of viruses remains hard. Still, it is no longer an academic interest focus but is rather driven by economic pursuits (e.g., zero-day markets) and cyberwarfare.

Infection Cleanup

Cleaning up after an infection highly depends on the extent of the damage. It may be necessary to restore and/or repair files; numerous antivirus companies provide this type of service. In some cases, when a virus runs with root privileges, it may be necessary to rebuild the entire system. In this case, recompiling the system may not be sufficient. The malware may have infected the compiler and created a backdoor, such that recompiling will simply reintroduce the malware into the compiler. In that case, it may be necessary to resort to original media and data backups.

Software Solutions Assurance Methodologies

As discussed earlier, software security can be compromised as a result of memory safety attacks that include the following:

- buffer overflows, which may be used to read/write data on stack/heap or to inject code (ultimately via a root shell);
- format string errors, used to read/write stack data;
- integer overflow errors, used to change programs' control flow; and
- TOCTOU problems, used to raise privileges.

Various methodologies and associated approaches that may be used as part of the software development life cycle to prevent these attacks are described in the following sections.

Defensive Programming

If you think of defensive driving as an analogy, it is about avoiding dependence on anyone but yourself. Minimizing trust makes it possible to better react to unexpected events (e.g., avoid a crash, or worse). Defensive programming pretty much works in the same way. Each software module is responsible for checking the validity of all inputs it receives and throwing exceptions or exiting rather than running malicious code and/or trusting inputs, even when they come from callers you know.

Defensive programming requires code reviews. While real or imagined, these reviews force programmers to organize their code and focus on code correctness to address issues that could raise flags. One approach to defensive programming is to provide developers with better languages and libraries that render code less prone to mistakes. For example, Java Runtime checks bounds automatically, and C++ comes with a safe std::string class. Secure coding relies on practices and rules, as illustrated in the following code.

- Practice: Analyze all inputs, whatever they are[26]:
```
char digit_to_char(int i) {
  char convert[] = "0123456789";
  return convert [i];
}
```
  - Think about all potential inputs, no matter how peculiar[27]:
```
char digit_to_char(int i) {
  char convert[] = "0123456789";
  if(i < 0 || i > 9)
    return '?';
  return convert[i];
}
```
  - Enforce rule compliance at runtime.
- Rule: Make use of safe string functions or libraries.
  String library routines typically included in libraries assume target buffers have sufficient length[28]:
```
char str[4];
char buf[10] = "good";
strcpy(str, "hello"); //overflows str
strcat(buf, " day to you"); //overflows buf
```
  Safe versions: check the destination length[29]:
```
char str[4];
char buf[10] = "good";
strcpy(str, "hello",sizeof(str)); //fails
```

27 Code reproduced with permission from Dave Levin
26 Code reproduced with permission from Dave Levin

```
strcat(buf, " day to you",sizeof(buf)); //fails
```

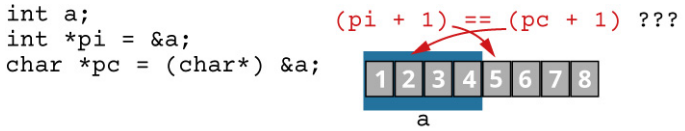
Again, you must know your system's and language's semantics.

Note that strncpy/strncat do not null-terminate if they run up against the size limit; therefore, it is better to use strlcpy/strlcat. These functions are not "insecure," but they are commonly misused.

It is actually even better to use safe string libraries as they are designed to ensure that strings are used safely. The following code illustrates the use of the very secure FTP (vsftp) string library[30]:

```
impl hidden
void str_alloc_text(struct mystr* p_str, const char* p_src);
void str_append_str(struct mystr* p_str, const struct mystr* p_other);
int str_equal(const struct mystr* p_strl, const strct mystr* p_str2);
int str_contains_space(const struct mystr* p_str);
…struc mystr; //impl hidden
void str_alloc_text(struct mystr* p_str, const char* p_src);
void str_append_str(struct mystr* p_str, const struct mystr* p_other);
int str_equal(const struct mystr* p_strl, const strct mystr* p_str2);
int str_contains_space(const struct mystr* p_str);
…
```

- Rule: Understand pointer arithmetic.



(attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

The command sizeof() returns a number of bytes, but pointer arithmetic multiplies by the size of the type[31]:

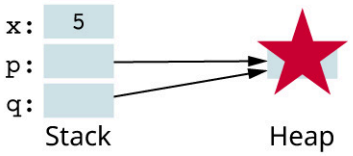
```
int SIZE * sizeof(int);
int buf[SIZE] = { …};
int *buf_ptr = buf;

while (!done() && buf_ptr < (buf + sizeof(buf))) {
    *buf_ptr++ = getnext(); // will overflow
}
so, use the right units:
while (!done() && buf_ptr < (buf + SIZE)) {
  *buf_ptr++ = getnext(); //stays in bounds
}
```

- Practice: Defend against dangling pointers.

28 Code reproduced with permission from Dave Levin
29 Code reproduced with permission from Dave Levin
30 Code reproduced with permission from Dave Levin
31 Code reproduced with permission from Dave Levin

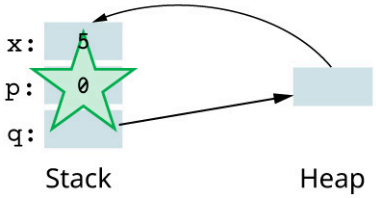
```
int x = 5;
int *p = malloc(sizeof(int));
free(p);
int **q= malloc(sizeof(int*)); //may reuse p's space
*q = &x;
*p = 5;
**q = 3; //crash (or worse)!
```



(attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

- Rule: Use NULL after free

```
int x = 5;
int *p = malloc(sizeof(int));
free(p);
p = NULL; //defend against bad deref
int **q = malloc(sizeof(int*)); //may reuse p's space
*q = &x;
*p = 5; //(good) crash
**q = 3;
```



(attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

- Practice: Manage memory properly.
  Some programmers commonly use `goto` chains in C to avoid duplicating or missing code. This approach is similar to using a `try/finally` clause in Java. A good coding practice is to always review and confirm the logic correctness[32].

```
int foo(int arg1, int arg2) {
  struct foo *pf1, *pf2;
  int retc = -1;
  pf1 = malloc(sizeof(struct foo));
  if (!isok(arg1)) goto DONE;
  …
  pf2 = malloc(sizeof(struct foo));
  if (!isok(arg2)) goto FAIL_ARG2;
  …
  retc = 0;
FAIL_ARG2:
  free(pf2); //fallthru
DONE:
```

```
    free(pf1);
    return retc;
}
```

- Rule: Always use a safe allocator.
  ASLR makes the base address of libraries unpredictable to defeat exploits. Using the same thinking and at the cost of reduced performance, addresses returned by calls to malloc should be made unpredictable to avoid heap-based overflows.
- Rule: Favor safe libraries.
  Libraries encapsulate well-thought-out design, so take advantage of them. For example, smart pointers libraries (part of C++11 standard) limit pointers to only safe operations and manage lifetimes appropriately. Networking libraries such as Google protocol buffers and Apache Thrift are good for dealing with network-transmitted data: they are both efficient and also ensure inputs are handled securely (e.g., validation, parsing).

Secure Software Implementation

The trusted computer base (TCB) of every system may include the monitor, compiler, OS, CPU, memory, keyboard, and other peripherals. Basic security assumes a correct, complete, and secure TCB. A good TCB is small and separates privileges. Using a small and simple TCB ensures that fewer components must work correctly to ensure security and are less susceptible to compromises. As security software in the TCB grows and becomes more complex (e.g., operating systems kernels used to enforce security often include a large amount of code), it becomes vulnerable and may be bypassed. Rather than compromising a device driver's security, it is best to reduce the size of the operating system kernel by creating microkernels that leverage device drivers located outside the kernel. The least privilege, a privilege separation approach, should also be applied to keep privileged operations modules as small as possible. It is important to only give the right level of privilege to a task. There is no reason to give more privileges than needed to a task. For example, it is not necessary for a web server daemon to allow root to bind to port 80. Doing so will enable the web server to run as root. Similarly, email editors should not make it possible to access a shell. You need to remember that trust is transitive, trusting something means that you trust what it trusts, which can lead to trouble.

Thinking about code safety is critical to ensure code safety and correctness. Code modularity is important as it helps to gain confidence in code function by function and module by module. It is necessary to verify that pre- and post-conditions hold before and after a function is called, respectively. This helps define contracts for using modules (e.g., a given statement's post-condition needs to correspond to another statement's pre-condition). Pre- and post-conditions help document code and facilitate reasoning about code. Invariants help set conditions that are always true within parts of a function. All the aforementioned defensive programming techniques make it possible to verify functions based on code and associated annotations every time the code is invoked. Defensive programming allows reasoning about functions' safety each time they are called, and pre-conditions act as constraints that users must follow each time they use functions.

The following code illustrates the preconditions that are required to ensure safety. The approach consists of identifying each memory access and annotating them with the preconditions they require and propagate the requirements up.

32 Code reproduced with permission from Dave Levin

```
/* requires: a != NULL  */
/* requires: n <= size(a)  */
int sum(int a[], size_t n) {
    int total = 0;
    for (size_t i=0; i<n; i++)
        total += a[i];
    return total;
}
```

Memory access

(attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

In this example, the memory access led to the following annotations[33]:

```
/* requires: a != NULL        */
/* requires: Ø <= i          */
/* requires: i < size(a)        */
```

The second annotation is taken care of by `size_t i`, which ensures that `0 <= i` always holds. The two other annotations were not guaranteed by this function code so they were moved up as preconditions to ensure that `n <= size(a)`.

Here is another example of the pre- and post-condition checks that are needed when using or creating pointer respectively to ensure safety[34]:

```
/* requires: p != NULL (and p is a valid pointer) */
/* ensures: retval is the first four bytes p pointed to */
int deref(int *p) {
  return *p;
}
/* ensures: retval != NULL (and a valid pointer) */
void *myalloc(size_t n) {
  void *p = malloc(n);
  if (!p) {
    perror("malloc");
    exit(1);
  }
  return p;
}
```

Testing

The goal of testing software quality is to ensure that the specification and implementation of programs match. Furthermore, testing assumes that the specification is correct but it does not necessarily assume that implementation is correct.

Developers should not be end-to-end testers. A developer should focus on the implementation and unit testing while a tester focuses on the specification, which avoids related mistakes at both levels.

Testing approaches may be classified as illustrated in Figure 14.33.

33 Code reproduced with permission from Dave Levin
34 Code reproduced with permission from Dave Levin

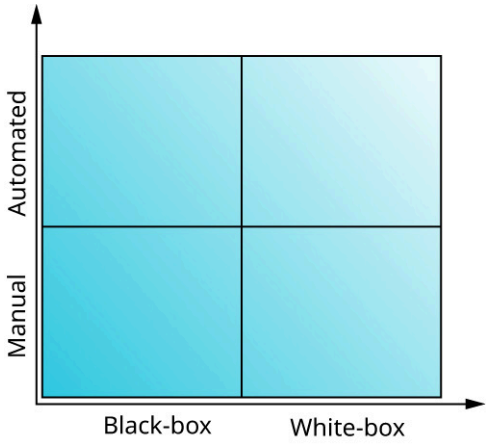


Figure 14.33 Testing approaches can be either manual or automated and test software using black-box or white-box techniques. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

As illustrated, there are various ways to conduct testing. Automated testing involves writing scripts or using testing frameworks to simulate user interactions with a software application (e.g., clicking buttons, entering data, and verifying outcomes). It enhances efficiency by automating repetitive testing tasks, which results in saving valuable time and resources. Automated testing also improves accuracy by minimizing human errors, and it enhances test coverage by enabling regression testing and continuous testing and delivery via CI/CD pipelines. Manual testing assumes the creation of efficient test suites to provide optimal test coverage and may struggle to achieve comprehensive coverage and scalability due to time and resource constraints. Black-box testing does not require analyzing code, which works well for code that cannot be modified or is in a format that makes it difficult to analyze (e.g., obfuscated, managed, or binary code). White-box testing assumes an efficient test suite to provide the detailed tests evaluating the source code.

Test suites must be sized properly. Small numbers of tests cannot identify all the defects and large numbers of tests will slow down testing and make it harder to maintain tests due to bloating and redundancy. For example, the SQLite library (version 3.20.0) included approximately 125.4 thousand source lines of code (KSLOC) as compared to a project using it that had 730 times as much test code and scripts (i.e., 91616.0 KSLOC). It should be noted that KSLOC lines of code exclude blank lines and comments.

Code coverage is a metric used to quantify the extent of program code testing when using a given test suite. Function testing coverage focuses on which functions are called. Statement testing coverage focuses on which statements are executed, and branch testing coverage focuses on which branches are executed. Testing coverage is computed as a percentage of a program's testing aspects covered by a given test suite. Practically, testing 100% of the code in a program is impossible. Cyclomatic complexity refers to the number of paths that exist in a program and should, in theory, be tested. That said, some code may not be accessible, and even if full testing were possible, it could take an infinite amount of time. Safety-critical applications do require 100% coverage. SQLite, as an example, has 100% branch coverage.

In manual white-box testing, tests are written by hand using full knowledge of the source code/deployment/infrastructure. They can be automated (e.g., run on all saves or commits).

In manual black-box testing, the tester interacts with the system in a black-box fashion and crafts ill-formed inputs, tests them, and records how the system reacts.

Automated testing techniques include:

- Code analysis
  - Static: Evaluating the source code can identify many of the bugs we have discussed.
  - Dynamic: Run in a VM and look for invalid writes (Valgrind).
- Fuzz testing

- Generate many random inputs and see if the program fails.
- Typically, it involves many inputs.
- There are various possible kinds of fuzzing:
  - Black-box: The tool knows nothing about the program or its input; it is easy to use and get started, but it will explore only shallow states unless it gets lucky.
  - Grammar-based: The tool generates input informed by grammar; more work is required to use it and to produce the grammar, but it can go deeper into the state space.
  - White-box: The tool generates new inputs at least partially informed by the code of the program being fuzzed; it is often easy to use but computationally expensive.
- Fuzzing inputs may be provided in different ways:
  - Mutation: Take a legal input and mutate it, using that as input; the legal input might be human-produced or automated (e.g., from a grammar or SMT solver query); mutation might also be forced to adhere to grammar.
  - Generational: Generate input from scratch (e.g., from a grammar).
  - Combinations: Generate initial input, mutate, generate new inputs, and generate mutations according to grammar.
- File-based fuzzing mutates or generates inputs and then runs the target program with them to see what happens; an example is Radamsa,[35] a mutation-based, black-box fuzzer, which mutates inputs that are given and passes them along[36]:

```
% echo "1 + (2 + (3 + 4))" | radamsa --seed 12 -n 4
5!++ (3 + -5))
1 + (3 + 41907596644)
1 + (-4 + (3 + 4))
1 + (2 + (3 + 4
% echo … | radamsa --seed 12 -n 4 | bc -1
```

  Another example is Blab, which generates inputs according to grammar (i.e., it is grammar-based), specified as regexps and CFGs[37]:

```
% blab -e '(([wrstp][aeiouy]{1,2}){1,4} 32}{5} 10'
soty wypisi tisyro to patu
```

- Network-based fuzzing can act as half of a communicating pair; inputs could be produced by replaying previously recorded interaction, and altering it, or producing it from scratch (e.g., from a protocol grammar). It can also act as a "man-in-the-middle" by mutating inputs exchanged between parties (perhaps informed by grammar).
- There are many fuzzers out there, such as American Fuzzy Lop (mutation-based white-box buzzer), SPIKE (library for creating network-based fuzzers), Burp Intruder (automates customized attacks against web apps), BFF, and Sulley. Fuzzers help find the root cause of a crash by answering questions such as: Is there a smaller input that crashes in the same spot (makes it easier to understand)? Are there multiple crashes that point back to the same bug? Can you determine if a crash represents an exploitable vulnerability (in particular, is there a buffer overrun)?
- Fuzzing may help find memory errors:
  First, compile the program with AddressSanitizer (ASan), which instruments accesses to arrays to check for overflows, and use-after-free errors, then fuzz it and check if the program crashed with an ASan-signaled error. If that is the case, worry about exploitability; similarly, you can compile with other sorts of error checkers for testing (e.g., Valgrind memcheck).

- Automated black-box testing uses fuzzing components as explained to generate test cases, execute the

35 See https://gitlab.com/akihe/radamsa
36 Code reproduced with permission from Dave Levin
37 Code reproduced with permission from Dave Levin

applications, and perform detection and logging.

- In automated white-box testing, tests are created automatically/dynamically. Tools exist to perform this type of testing and may record a trace of the tested program on well-formed inputs and perform symbolic execution to capture constraints on inputs. Automated testing may then negate a constraint or use a constraint solver to derive a new input and run on that input. When using American fuzzy lop, compile-time instrumentation is provided, and the instrumentation guides genetic algorithms.
- Penetration testing
  Another testing technique is penetration (pen) testing. Fuzz testing is a form of pen testing. Pen testing assesses security by actively trying to find exploitable vulnerabilities, which is useful for both attackers and defenders. Pen testing is useful at many different levels (e.g., for testing programs, testing applications, testing a network, testing a server).

Reverse Engineering

Reverse engineering (RE) is the process of discovering the technological principles of a program through analysis of its structure, function, and operation. It corresponds to the solution development life cycle run backward. Reverse engineering is useful for malware analysis, vulnerability or exploit research, copyright/patent violations check, interoperability assessment (e.g. understanding a file or protocol format), and copy protection removal. The legality of RE is a gray area and usually breaches the end-user license agreement (EULA) software contract. Additionally, the Digital Millennium Contract Act (DMCA) governs reverse engineering in the United States. You "may circumvent a technological measure . . . solely for the purpose of enabling interoperability of an independently created computer program."

There are two techniques used for RE, and a combination of the two works best in general:

- Static code analysis focuses on the code structure and uses a disassembler.
- Dynamic code analysis focuses on the code operation and uses tracing, hooking, and debuggers.

Disassembling code is difficult and often imperfect due to benign optimizations (e.g., constant folding, dead code elimination, inline expansion) and intentional obfuscation (e.g., packing, no-op instructions). Malware uses a lot of packing; overall, 90% of the code is packed.

Dynamic analysis takes advantage of debuggers' features (e.g., trace every instruction a program executes via single stepping; let the program execute normally until an exception; at every step or exception, observe/modify instructions/stack/heap/register set; inject exceptions at arbitrary code locations; use INT3 instruction to generate a breakpoint exception). Debugging has many benefits as it is sometimes easier to see what the code does or allow unpacking to let the code unpack itself and debug as normal. Most debuggers have built-in disassemblers anyway. It is always possible to combine static and dynamic analysis. However, it is possible to run into difficulties with debugging when executing potentially malicious code (using an isolated virtual machine). The attacker may have used anti-debugging methods to detect the debugger and changed the program behavior so that it runs differently than when not being debugged (e.g., used IsDebuggerPresent(), INT3 scanning, timing, VM-detection, pop ss trick). Anti-anti-debugging can be tedious.

A common way of evasion is to detect evidence of monitoring systems (e.g., fingerprint a machine/look for fingerprints) or hide real malicious intents if necessary as follows:

```
IF VM_PRESENT() or DEBUGGER_PRESENT()     Terminate()        // hide real intents ELSE
    Malicious_Behavior()    //real intent
```

The general taxonomy of malware evasion is illustrated in Table 14.5.

Difficulty	Layer of Abstraction	Examples
Easiest	Application	Installation, execution
Easy	Hardware	Device name, driver
Somewhat difficult	Environment	Memory, execution artifacts
More difficult	Behavior	Timing

Table 14.5 Malware Evasion Taxonomy

In general, there is a prevalence of evasion, and 40% of malware samples exhibit fewer malicious events with a debugger attached.

Internet Solutions Cybersecurity

The Internet is a network of networks, which is an interconnected set of nodes. Nodes at the edge of the network are called (end-)hosts while nodes within the core of the network are routers. The network uses IP addresses to name the nodes, while humans use more easily memorable host names that point to the corresponding IP addresses. The Dynamic Host Configuration Protocol (DHCP) can create IP addresses and associate them with hosts as they connect to the network. The Domain Name Service (DNS) maps domain names to corresponding routable IP addresses (Figure 14.34).

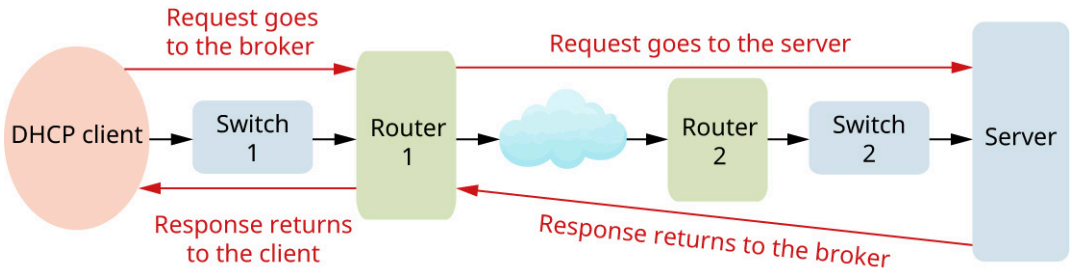


Figure 14.34 Users obtain IP addresses from the DHCP server when they boot up a system. This process helps ensure that only authorized users can access the Internet. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Web Infrastructure Assurance

The World Wide Web (the Web) is an organizational system for information that is accessible by using the Internet. In other words, the Web infrastructure is provided by the TCP/IP network stack. In particular, review the roles of the various layers of the Internet TCP/IP network stack, the protocols associated to each layer, and how packets are created and exchanged over the Internet. It is important to know what each layer is responsible for and what the predominant protocols are at each layer. Finally, experimenting with existing network protocol analyzers (e.g., Wireshark) to study packets and communication is recommended. The overall design principles of the TCP/IP network stack have been critical to making an Internet that can evolve with changing needs (at least for the most part), but the the details really matter. In the following, we will dig into specific protocols to understand the kinds of attacks that can happen at the networking layer and how to protect against them.

As noted earlier, the DHCP protocol can create IP addresses dynamically and associate them to hosts as they connect on the network. The DNS maps domains names to corresponding routable IP addresses (Figure 14.35).

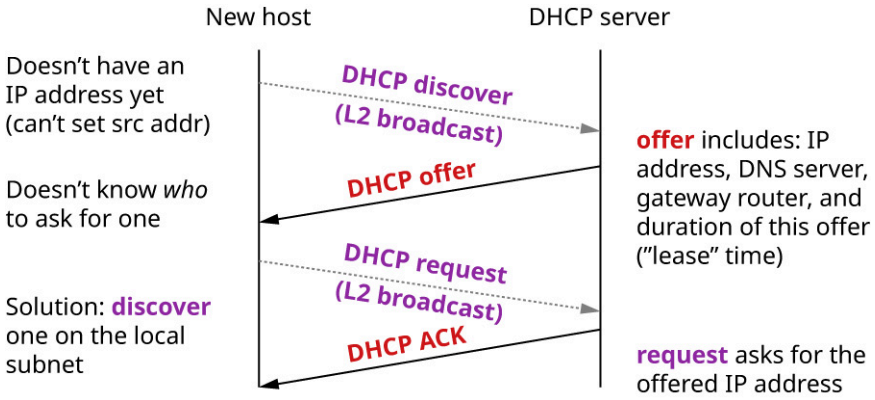


Figure 14.35 The DHCP protocol creates IP addresses dynamically and associates them to hosts as they connect on the network. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

Because DHCP requests are broadcasted to all nodes neighboring nodes initially, attackers on the same subnet can hear a new host request and can race the actual DHCP server to replace the DNS server (i.e., redirect any of a host's lookups such as "what IP address should I use when trying to connect to google.com?" to a machine of the attacker's choice) or the gateway, where the host sends all of its outgoing traffic so that the host does not have to figure out routes by itself; by making a machine of the attacker's choice the gateway, the attacker would be able to act as the MitM and gain access to all of the traffic to and from the user's machine. So, how can a user detect such an attack?

The DNS service divides the domain name namespace into zones for administrative reasons. Subdomains do not need to be in the same zone, which allows the owner of one zone (e.g., nyu.edu) to delegate responsibility to another (e.g., cs.nyu.edu). The name server is the piece of code that answers queries of the form "What is the IP address for cs.nyu.edu?" Every zone must run at least two name servers. Caching is central to the success of the DNS service. Unfortunately, it is also central to attacks such as cache poisoning, which consists of filling a victim's cache with false information (Figure 14.36).

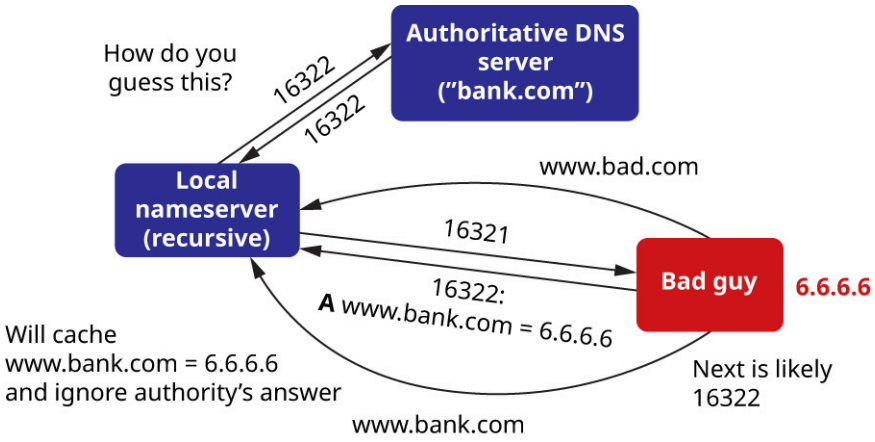


Figure 14.36 A DNS cache poisoning example consists of filling a local name server DNS cache with false information. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

In the diagram, the recursive name server is the name server that does the heavy lifting and issues DNS queries on behalf of the client resolver (i.e., the host that asks DNS queries on behalf of the client) until an authoritative answer returns. Because the local resolver has a lot of incoming/outgoing queries at any point in time, it determines which response maps to which queries by using a query ID (i.e., a 16-bit field in the DNS header shown as 16322 as an example in the diagram). The requester sets the query ID to whatever it wants and the responder must provide the same value in its response. For a cache poisoning attack to work, the attacker must guess the query ID, ask for it, and go from there. Note that a partial defense is to randomize query IDs, but this takes space, and the attacker can issue a lot of query IDs. Once the attacker has guessed

the query ID, it must guess the source port number, which is typically constant for a given server (often always 53). Note that if the answer is already in the cache, the attacker will avoid issuing a query in the first place.

The same cache poisoning approach may be used to poison more than one record (Figure 14.37).

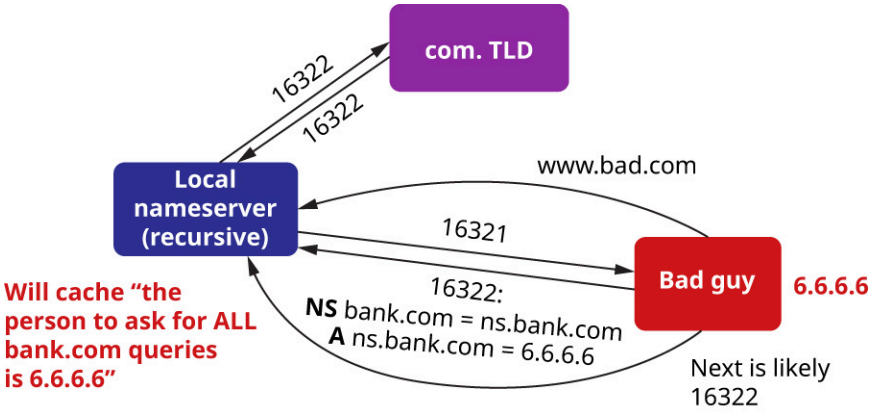


Figure 14.37 The same cache poisoning approach may be used to poison more than one cached record. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

Note that randomizing query ID is not sufficient in itself because there are only 16 bits of entropy. So the source port should be randomized as well because there is no reason for it to stay and it is possible to obtain another 16 bits of entropy in this way. Another solution is to use Domain Name System Security Extensions (DNSSEC). If everyone has deployed it, and if you know the root's keys, then DNSSEC prevents spoofed responses. DNSSEC uses public key infrastructure (PKI) to secure communication between the DNS servers in the various zones, and the authoritative answer is signed. But unlike PKIs, if one or more name servers has not deployed DNSSEC (which is the case in incremental deployments), then DNSSEC is not very useful. While it is possible to ignore name server responses without DNSSEC, this would improve security but it prevents the user from connecting to a number of hosts.

Now, let us focus on the networking protocols and study possible TCP/IP attacks and defenses. In particular, let us look at the (inter)network layer, which works across different link technologies, bridges multiple "subnets" to provide end-to-end Internet connectivity between nodes, and provides global addressing (IP addresses). Note that if the transport layer uses the TCP protocol, it will only result in best-effort delivery of data (i.e., no retransmissions). The IPv4 packet header used by the IP protocol is 20 B long, and one of the header fields is the source IP address. Nothing in the IP protocol enforces that the source IP address is yours. Furthermore, the IP protocol does not protect the payload or headers. Source spoofing exploits this (Figure 14.38).

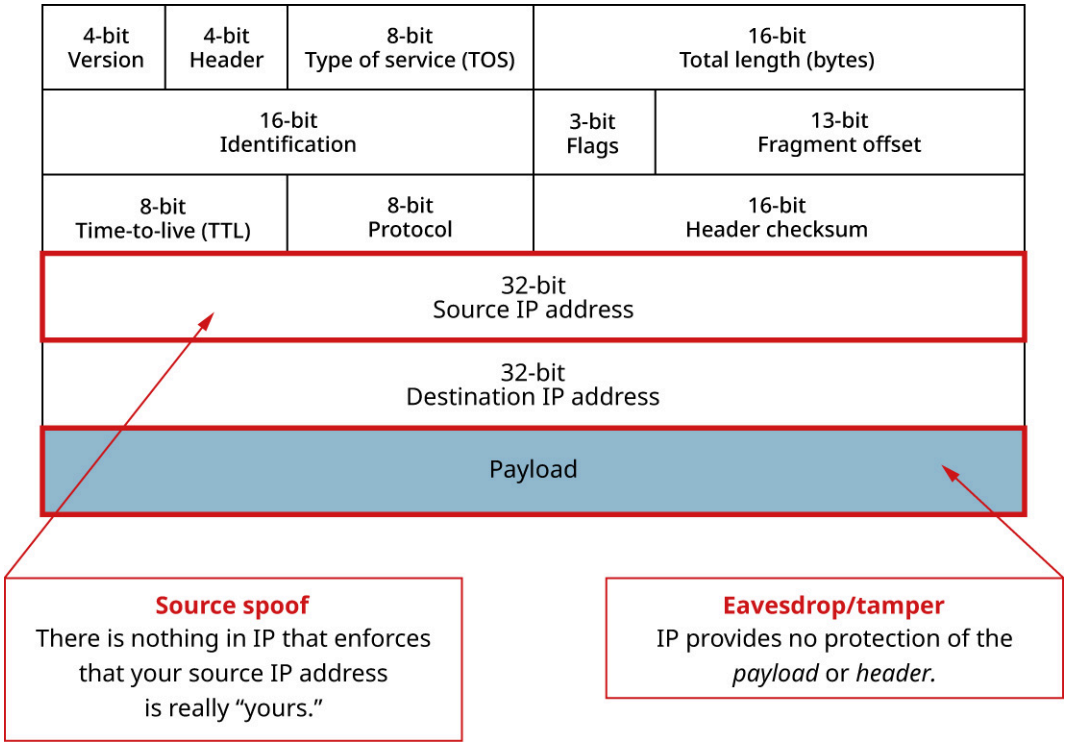


Figure 14.38 Source spoofing exploits the fact that the IP protocol does not protect the payload or headers. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

Source spoofing may be used to send many emails from one computer (i.e., email spamming). The recipient may, in return, block emails from a given (source) IP address but the attacker could spoof the source IP address as a countermeasure. So, does a packet you receive have a spoofed source?

Because the Internet operates via destination-based routing, the response to a spoofed-source message sent by an attacker goes to the spoofed source rather than the attacker (i.e., pkt (spoofed source) -> destination: pkt -> spoofed source). Therefore, to know whether a packet you receive has a spoofed source, you can send a challenge packet to the possibly spoofed source (e.g., a difficult-to-guess, random number used once [nonce]). If the recipient can answer the challenge, then it is likely that the source was not spoofed. The problem with this approach is that you have to do this for every packet (i.e., every packet should have something difficult to guess). This is analogous to the easily predicted query IDs in the DNS query poisoning attacks that facilitated Kaminsky's attack.

Source spoofing may also be used for denial of service (DoS) attacks. The idea is to generate as much traffic as possible to congest the victim's network. An easy defense is to block all traffic from a given source near the edge of your network. An easy countermeasure is to spoof the source address. Challenges will not help here because the damage has been done by the time the packets reach the core of your network. So, ideally, you would need to detect such spoofing near the source, and egress filtering does exactly that. The point (router/switch) at which traffic enters your network is the ingress point, and the point (router/switch) at which traffic leaves your network is the egress point. While you do not know who owns all IP addresses worldwide, you do know who in your network gets what IP addresses. Therefore, your egress point can drop any packets whose source IP address does not match the IP address your network assigned to that machine. This egress filtering approach is not often deployed because your egress point bears the costs but your network does not gain any benefit.

The defense methods suggested earlier to counter eavesdropping/tampering with IP headers are clearly not bulletproof. A better protection method against the fact that no security is built into IP is to deeply secure IP over IP. This is done by using a virtual private network (VPN). The goal of a VPN is to allow a client to connect to a trusted network from within an untrusted network. For example, you could use a VPN to connect to your

company's network for payroll file access while visiting a competitor's office. In that case, a VPN client and server would create an end-to-end encrypted/authenticated channel, as illustrated in Figure 14.39. A predominant way of achieving this is to use Internet Protocol Security (IPSec) to secure IP datagrams (instead of using TLS or secure shell at the application layer level). This was considered a good idea circa 1992–1993 as it would secure all traffic (not just TCP/UDP) and automatically secure applications (without requiring changes). It also provides built-in firewalling/access control. Initial proposed standards were published in 1988, and a revision (i.e., Internet Key Exchange version 2) was approved in 2005.

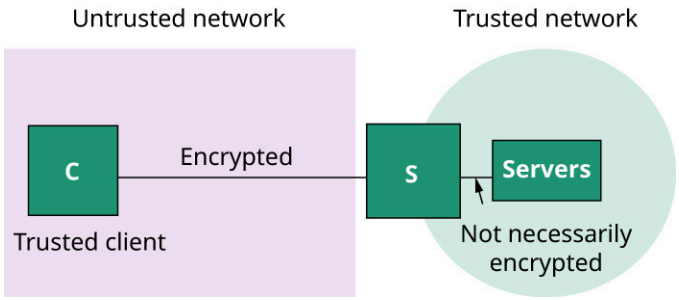


Figure 14.39 A VPN client and server can be used to create an end-to-end encrypted/authenticated channel. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

The Internet Key Exchange addressed anonymity issues and DoS prevention. There have been many implementations of IPSec, and nearly all deployments are in VPN settings. People ended up switching over to SSL/VPN, but that was not how SSL was intended to be used. IPSec is regarded today as a semifailure as it is complex, hard to use, and exhibits design flaws. IPsec did not get the usage model right, but SSL/TLS and SSH (discussed later in this subsection) got it right.

IPSec operates in a few different modes:

- Transport mode: Encrypts the payload but not the headers
- Tunnel mode: Encrypts the payload and the headers

The corresponding packet formats are illustrated in Figure 14.40.

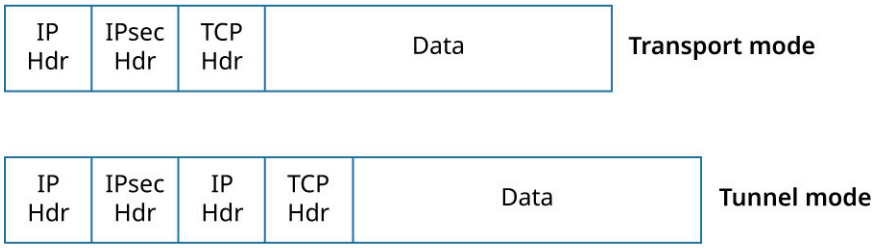


Figure 14.40 IPsec uses specific packet formats for transport mode and tunnel mode. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

For routing to work when tunnel mode encrypts the headers, IPSec encrypts the entire IP packet and makes it the payload of another IP packet.

Figure 14.41 illustrates using IPSec in tunnel mode. In this case, the VPN server decrypts and then sends the payload (itself a full IP packet) as if it had just received it from the network. From the client/server's perspective, it looks like the client is physically connected to the network.

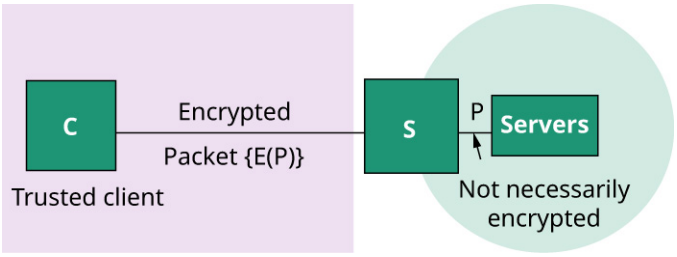


Figure 14.41 In IPsec tunnel mode, the VPN server decrypts and then sends the payload (itself a full IP packet) as if it had just received it from the network. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

Now, let us focus on the transport layer of the TCP/IP stack and study possible TCP/IP attacks and defenses. The transport layers ensure end-to-end communication between processes. It provides different services, including UDP (unreliable datagrams) and TCP (reliable byte stream). *Reliable* means that it keeps track of the data appropriately received and retransmits packets as necessary. Given best-effort delivery, the goal is to ensure reliability. All packets are delivered to applications in unmodified order (reasonably high probability) and TCP must robustly detect and retransmit corrupt or lost data. TCP's second job is flow and congestion control. The idea is to try to use as much of the network as is safe (not adversely affecting others' performance) and efficient (using network capacity). The TCP solution is to dynamically adapt how quickly it sends packets based on the network path's capacity. Furthermore, when an ACK doesn't return, the network may be beyond capacity and slow down. TCP is a connection-driven protocol, and there are various TCP flags in the TCP header that are used to manage connections as indicated:

- SYN: Used for setting up a connection
- ACK: Acknowledgments for data and "control" packets
- FIN: Used for shutting down a connection (two-way) using FIN and FIN+ACK
- RST: Used for shutting down notification (says "delete all your local state, because I do not know what you are talking about")

Various attacks are known to take advantage of the transport layer vulnerability. For example, SYN flooding takes advantage of a vulnerability in TCP's connection setup's three-way handshake as illustrated in Figure 14.42.

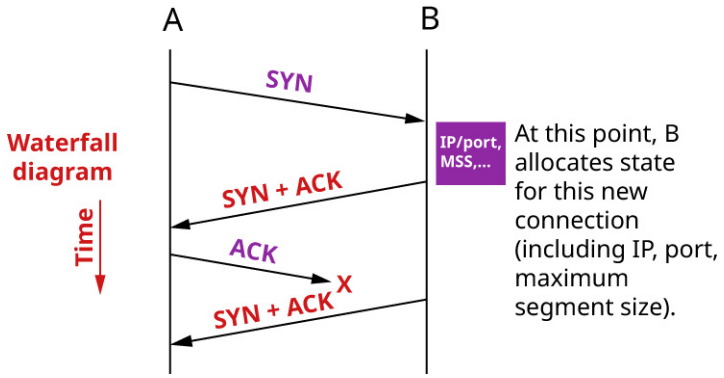


Figure 14.42 SYN flooding takes advantage of a vulnerability in TCP's connection setup's three-way handshake. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

If B does not receive an acknowledgment, it will hold onto this local state and retransmit SYN+ACK until it hears back or times out (up to 63 s). It is easy to detect many incomplete handshakes from a single IP address and then spoof the source IP address as illustrated in Figure 14.43 (it is just a field in a header as described earlier that can be set to whatever the attacker "C" likes). A possible problem is that the host who owns that spoofed IP address may respond to the SYN+ACK with a RST, deleting the local state at the victim. Therefore, an attacker should spoof an IP address of a host that it knows will not respond.

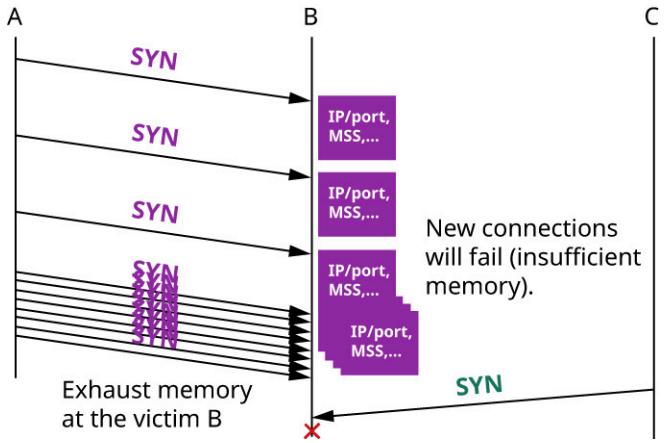


Figure 14.43 This IP address spoofing example illustrates how to detect many incomplete handshakes from a single IP address and then spoof the source IP address. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

A defense against SYN flooding is to use SYN cookies, as illustrated in Figure 14.44.

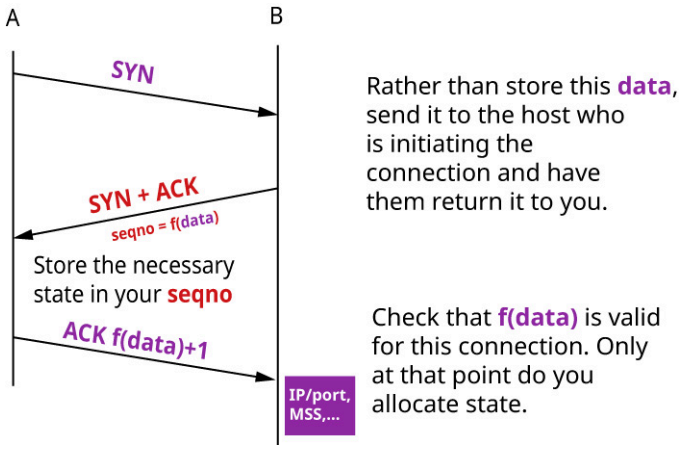


Figure 14.44 Using SYN cookies provides a defense against SYN flooding. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

The SYN cookie format is illustrated in Figure 14.45.

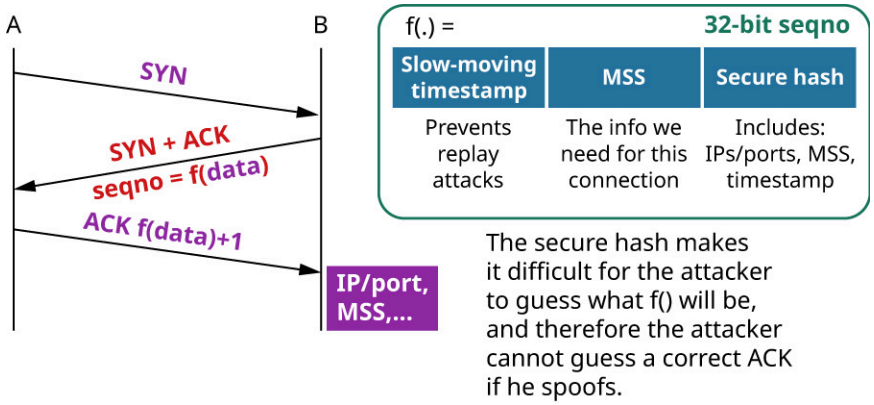


Figure 14.45 SYN cookies use a specific format. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

Injection attacks take advantage of having a node on the path between source and destination. In that case, injecting packets with the correct sequence number is trivial. If the node is not on the path, it would need to guess the sequence number, which is difficult. Initial sequence numbers used to be deterministic, and it was easy to wreak havoc by sending RSTs, injecting data packets into an existing connection (i.e., TCP veto attacks),

or initiating and using an entire connection without ever hearing the other end. Figure 14.46 illustrates one type of attack known as the Mitnick attack.

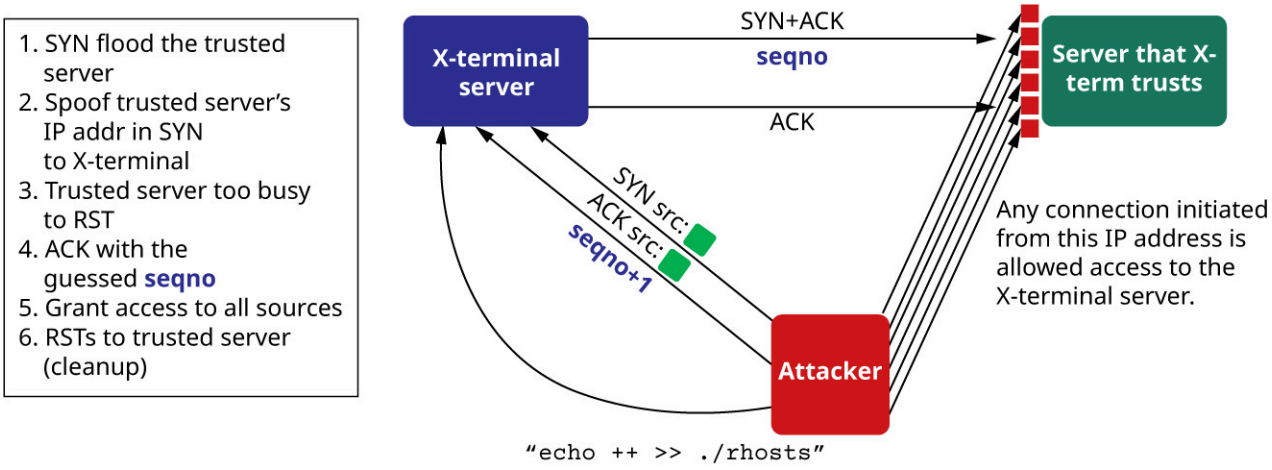


Figure 14.46 A Mitnick attack spoofs a trusted server's IP address to gain access to the X-terminal server to which it connects. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

A typical defense is to ensure that the initial sequence number is difficult to predict.

OPT-ACK attacks take advantage of the fact that TCP uses ACKs not only for reliability but also for congestion control (i.e., the more ACKs come back, the faster it can send), as illustrated in Figure 14.47.

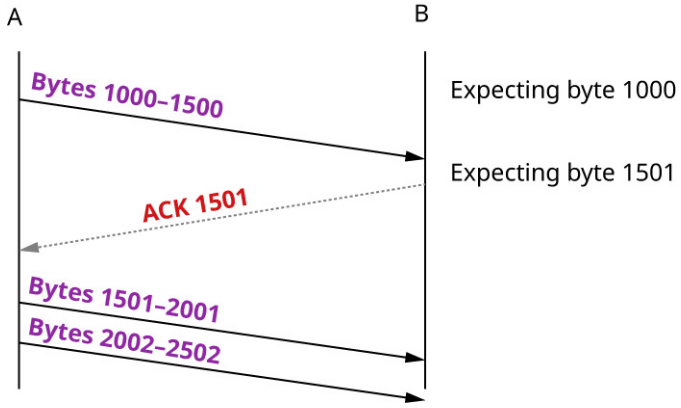


Figure 14.47 An OPT-ACK attack takes advantage of the fact that TCP uses ACKs not only for reliability but also for congestion control. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

An attacker can exploit this as illustrated in Figure 14.48.

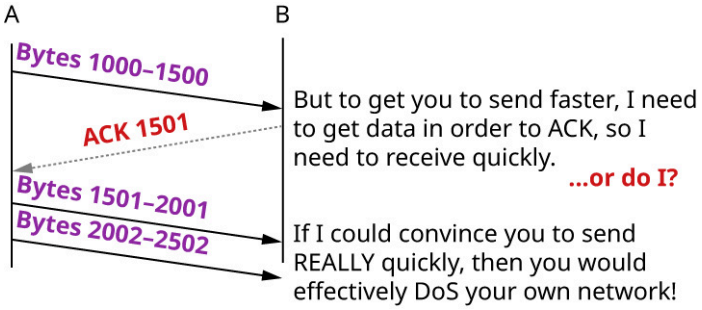


Figure 14.48 An OPT-ACK exploit convinces TCP to send packets quickly, which results in a DoS attack on the network. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license; credit: reproduced with permission from Dave Levin)

The actual attack scheme is illustrated in Figure 14.49.

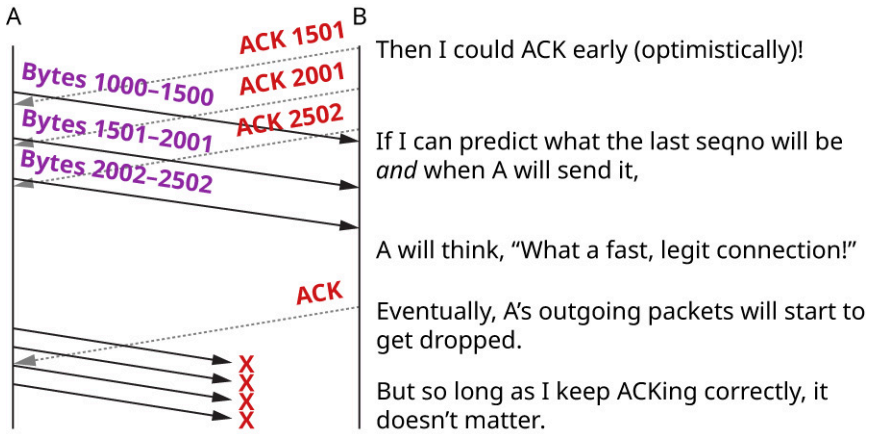


Figure 14.49 The OPT-ACK attack scheme results in packets getting dropped while TCP is fooled into thinking that packets were acknowledged. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

The big deal with this attack is its amplification factor. An attacker sends many bytes of data, causing the victim to send many more in response. There are examples of such attacks on NTP and DNSSEC. The attack is amplified in TCP due to its support for cumulative ACKs (i.e., "ACK *x*" says "I've seen all bytes up to but not including *x*"). Figure 14.50 illustrates the maximum number of bytes that can be sent by a victim per ACK.

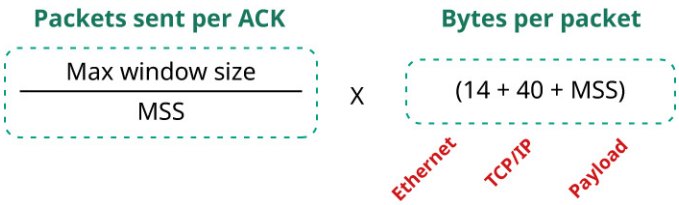


Figure 14.50 This illustration shows the maximum number of bytes per ACK. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Figure 14.51 shows the maximum number of ACKs that an attacker can send per second.

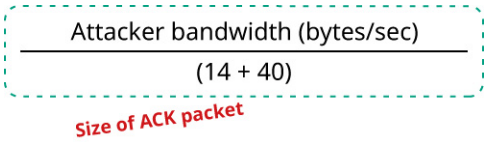


Figure 14.51 This illustration shows the maximum number of ACKs sent by the attacker. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Therefore, the amount of damage is the value of max window size and MSS (e.g., default max window size: 65,536 and default MSS: 536). In that case:

- Default amp factor: 65536 * (1/536 + 1/54) ~ 1336x
- Window scaling lets you increase this by a factor of 2^14
- Window scaling amp factor: ~1336 * 2^14 ~ 22M
- Using a minimum MSS of 88: ~ 32M

A challenge is to find a solution to defend against OPT-ACK in a way that is still compatible with existing implementations of TCP. Also, note that an essential goal in networking is incremental deployment. Ideally, we should be able to benefit from a system/modification when even a subset of hosts deploy it.

Now, let us focus on the application layer of the TCP/IP stack and study possible TCP/IP attacks and defenses. The Secure Socket Layer (SSL) protocol was originally a Netscape proprietary protocol that targeted e-commerce applications (i.e., what people thought the Web was for in 1994). The objective was to address outcomes such as "send my credit card to Amazon securely." The basic principle (circa 1994) was to authenticate the server (via certificate) and let the client access the server unauthenticated. SSLv1 was designed by Kipp Hickman and had serious security flaws. He addressed some of the flaws in SSLv2, which still

had security issues but was widely deployed. SSLv3 fixed these problems. The Transport Layer Security (TLS) 1.0 protocol was the first standardized version of SSL with some improvements for key derivation (refer to ietf.org RFC 2246). TLS 1.1 (RFC 4346) addressed some security flaws, and TLS 1.2 (RFC 5246) added more flexibility in using hash functions. TLS 1.3 brought significant changes (e.g., no RSA key exchange for forward secrecy, authenticated encryption modes, no RTT handshakes). As explained earlier, a trusted CA may vouch that a certain public key belongs to a particular site and issue a TLS certificate that abides by the x.509 format.

Web applications use HTTP over SSL/TLS (HTTPS), in which case the client knows that the server expects HTTPS (because it is specified in the URL and supported on a separate port on the server). Furthermore, the server certificate has its domain name. HTTP is stateless, and the lifetime of an HTTP session is as follows:

- The client connects to the server.
- The client issues a request.
- The server responds.
- The client issues a request for something in the response.
- The interaction continues until client has received all the information it needs.
- The client disconnects.

Because asymmetric (private key operations) are expensive (and HTTPS tends to involve a lot of SSL/TCP connections), caching pays off. Each handshake establishes a session, and clients can resume the session with the same keying material, skipping the key exchange. If the client and servers do not know each other's capabilities, they can discover them and automatically upgrade to TLS. This, however, may allow downgrade attacks.

DoS attacks on SSL/TLS rely on the SSL/TLS connection requiring TCP handshake, and TCP connections are easy attack with a DoS. Protection against these types of attacks needs to be at a lower layer and is provided by Datagram TLS (RFC 4347). DTLS is a slight modification of TLS that provides reliability for the handshake and ensures that data records are independent.

Tatu Ylonen originally designed the Secure Shell (SSH) Protocol, which is a replacement for rsh. It is now the standard tool for secure remote login, and it provides a lot of authentication mechanisms, such as remote X, file transfer, and port forwarding. The transport protocol used by SSH looks a lot like TLS. SSL does not use certificates; the server just has a raw public key and it provides the key when the client connects. The client stores the server's key on the first connection. Any changes in the key result in an error. The key can be authenticated from the band (i.e., the server operator tells the client the key fingerprint/hash over the phone; only the most concerned people do this). The SSH leap of faith authentication was considered extreme initially but is now considered clever. SSH client authentication first requires server authentication and then authenticates the client using various negotiated mechanisms (e.g., raw password, challenge-response, public key, GSS-API, Kerberos). SSH provides port forwarding/tunneling features. SSH port forwarding redirects network traffic to a particular port/IP address so that a remote host is made directly accessible by applications on the local host. The destination may be on the remote SSH server, or that server may be configured to forward to another remote host. SSH tunnels are powerful tools for IT administrators and malicious actors because they can transit an enterprise firewall undetected. As a result, tools are available to prevent unauthorized use of SSH tunnels through a corporate firewall. Figure 14.52 illustrates how an X11 remote connection can be established using SSH.

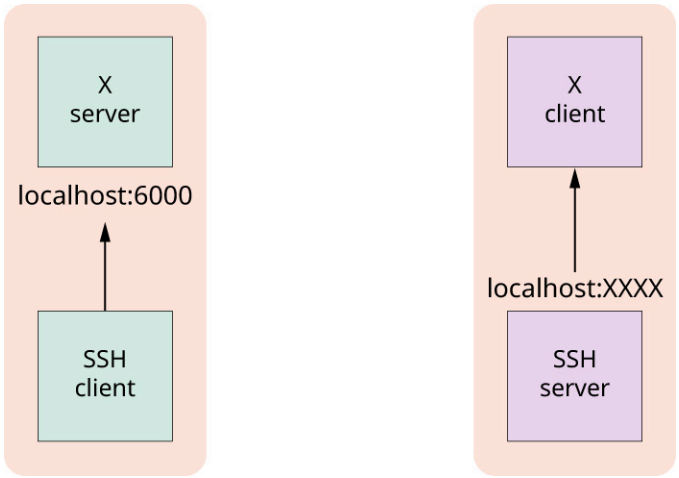


Figure 14.52 With an X11 remote SSH connection, the server simply has to "setenv DISPLAY localhost: XXXX" and the applications just automatically work. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Finally, SSH is backward and compatible with rsh, so other applications can be securely remoted without port forwarding. This is also useful when applications need insecure remote access.

The various application layer protocols we discussed are subject to a variety of attacks, including attack vectors, site design attacks, UI interface-based attacks, PKI attacks, implementation attacks (e.g., null termination attacks), Goto fail, heartbleed, BERserk attack, Logjam, Cloudbleed, and client-side HTTP interception.

Attack vectors are designed to attack the weakest certificate authority. They attack browser implementations. They may also find and exploit a key generation library bug that leads the attacker to discover all the private keys issued by that authority. They also attacked the cryptographic primitives, although this was more difficult to achieve.

SSLStrip attacks are examples of attacks that go after site design by proxying through the content without HTTPS. The defense is to default to HTTPS for all websites. You can also use HSTS (hypertext strict transport security), which is enforced by browsers; the header states to always expect HTTPS. HTTPS everywhere can also be forced using a browser extension. Some site design attacks use mixed content attacks. In this case, a page loads over HTTPS but contains content over HTTP (e.g., JavaScript). An active attacker can tamper with HTTP content to hijack the session. The defense is to issue browser warnings (e.g., "This page contains insecure content"), but the use of these warnings is inconsistent and the warnings are often ignored.

UI interface-based attacks exploit invalid certs (i.e., expired, misidentified URL, unknown CA such as a self-signed certificate). The defense is to issue browser warnings and require users to go through an anti-usability page to continue. Another type of UI interface-based attack is a picture-in-picture attack that spoofs the user interface (i.e., the attacker page draws a fake browser window with a lock icon). In this case, the defense is to create an individualized image.

PKI attacks compromise CAs. There was an example of such an attack in 2011 against a Dutch CA named DigiNotar. It issued a *.google.com certificate to an attacker that subsequently used it to orchestrate MitM attacks in Iran. Nobody noticed the attack until someone found the certificate in the wild. DigiNotar later admitted that dozens of fraudulent certificates were created. Google, Microsoft, Apple, and Mozilla all revoked the root DigiNotar certificate. The Dutch government took over DigiNotar, and the company subsequently went bankrupt and closed. In general, MD5/SHA1 is known to break and can generate collisions. In 2008, researchers showed they could create a rogue CA certificate using an MD5 collision. The attack consisted of colliding messages, A and B, with the same MD5 hash as follows:

- A: Site certificate: "cn=attack.com, pubkey=…."
- B: Delegated CA certificate: "pubkey=…. is allowed to sign certs for *"

- Get CA to sign A -- Signature is Sign(MD5(message))
- Signature also valid for B (same hash)
- The attacker is now a CA!
- Make a cert for any site, browsers will accept it

MD5 CA certificates still exist, but CAs have stopped signing certificates with them. SHA-1 should not be used either.

There are numerous other types of attacks:

- Goto fail (Feb. 2014) was an Apple SSL bug that resulted in skipping certificate check for almost a year.
- Heartbleed (April 2014) was an OpenSSL bug that leaked data, possibly including private key!
- Mozilla BERserk vulnerability (Oct 2014) was a bug in verifying certificate signatures that allowed spoofing certificates.
- Logjam (Oct 2016) took advantage of a TLS vulnerability to man-in-the middle "downgrade" attacks.
- Cloudbleed, one of the most popular "content delivery networks," acts as the SSL endpoint for many servers; a buffer overflow attack caused it to leak HTTPS data.
- Client-side HTTP interception leverages the fact that most antivirus software intercepts your HTTPS, generating poor implementations and introducing new vulnerabilities.

Web/Mobile Applications Frameworks Assurance

The most typical web/mobile application frameworks are web servers and web browsers. Other web/mobile application frameworks (e.g., application servers, business process management suites) operate using web browsers and extended web servers and use the same security mechanisms. This subsection will only focus on web servers and web browsers. Various types of attacks affect web information and interactions, including SQL injection, cross-site scripting (XSS), path (directory) traversal, cross-site request forgery, remote file inclusion (RFI), phishing, clickjacking, authentication/authorization attack, buffer errors, web browser attack, information leak/disclosure, and web server attack.

Here are several security risks we try to protect web servers (and web browsers) against:

- Risk 1: We want data stored on a web server to be protected from unauthorized access.
- Risk 2: We do not want malicious (or compromised) sites to be able to trash files/programs on user computers.
- Risk 3: We do not want a malicious site to be able to spy on or tamper with information or interactions with other websites.

The Federal Communications Commission has identified cybersecurity tips to help organizations protect against cyber threats when using the Internet. These include[38]:

- Ensure employees are trained regarding cybersecurity, including the organization's security principles.
- Keep the system updated with the latest security software and frequently run antivirus software.
- Install and maintain a firewall, which is a collection of programs designed to prevent hackers and other unauthorized users from accessing a system.
- Ensure that mobile devices and laptops are included in the security plan and secure all such devices.
- Back up important files and data and store the backed-up information separately.
- Ensure all employees have passwords and other credentials required to access the system and provide access on an as-needed basis.
- Ensure that Wi-Fi networks are hidden and secured through encryption.

The following design and implementation guidelines can be accessed online through a browser search:

- OMG cybersecurity initiatives (and related standards, guidelines, best practices, and other resources)

38 Federal Communications Commission. No Date. "Cybersecurity for Small Businesses." https://www.fcc.gov/communications-business-opportunities/cybersecurity-small-businesses#:.

- NIST cybersecurity standards (in particular NIST SP 800-53, SP 800-171, CSF, SP 1800 Series)
- Other global IT security frameworks and standards: ISO 27000 Series, COBIT, CIS Controls, HITRUST Common Security Framework, GDPR, COSO
- Industry IT security standards: PCI, HIPAA, PCI DSS, Sarbanes-Oxley (SOX), GLBA
- CyBok
- SAFECode
- *Open Source Security Testing Methodology Manual* (OSSTMM)
- Open Web Application Security Project (OWASP)
- Web Application Security Consortium Threat Classification (WASC-TC)
- Penetration Testing Execution Standard (PTES)
- Information Systems Security Assessment Framework (ISSAF)

Cloud-Centric Solutions Cybersecurity

The need to reduce costs and make IT more responsive to business changes are driving more and more Internet solutions (e.g., web/mobile information systems) to various cloud platforms. There are numerous obstacles that make it difficult for end users/organizations to adopt the cloud from a TRM assurance standpoint. Providing security for cloud environments that matches the levels found in commercial internal data centers is essential to helping modern organizations compete, and to allowing cloud service providers (CSPs) to meet their end users' needs.

Managing risk in the cloud requires that users fully consider exposure to threats and vulnerabilities, not only during procurement but also as an on-going process. Security in the cloud is a constant process and cloud users should continually monitor their cloud resources and work to improve their security posture. Threat actors in the cloud may target the same types of weaknesses as the ones found in traditional system architectures. However, when organizations use the cloud, they face additional cyber threats, including the following:

- Malicious CSP administrators:
  They can leverage privileged credentials or position to access, modify, or destroy information stored on the cloud platform. They can also leverage privileged credentials or positions to modify the cloud platform to gain access to networks connected to or consuming cloud resources.
- Cyber criminals and/or nation-state-sponsored actors:
  They can leverage a cloud architecture or configuration weakness to obtain sensitive data or consume cloud resources at the victim's expense. They may exploit weak cloud-based authentication mechanisms to obtain user credentials (e.g., password spray attacks). They may leverage compromised credentials or incorrect access privileges to access cloud resources. They may gain privileged access to the cloud environment to compromise tenant resources. They may leverage the trust relationship between an end user or organization's networks and cloud resources to pivot from clouds into protected networks or vice versa.
- Untrained or neglectful customer cloud administrators:
  They may expose sensitive data or cloud resources unintentionally.

Cloud Infrastructure Assurance

To match the levels of security that end users experience on premise, CSPs must make the proper investments in providing, proving and assuring appropriate levels of security over time. This requires building security and trust architectures that can assure each end user's applications and data are isolated and secured from those of others. Before moving mission-critical information systems to the cloud, end users require robust cybersecurity, trustworthy cybersecurity assurance, and cloud governance as follows:

- Robust security requires moving beyond a traditional perimeter-based approach to a layered cloud security architecture and an approach that assures the proper isolation of data, even in a shared,

multitenant cloud. This includes content protection at different layers in the cloud infrastructure, such as at the storage, hypervisor, virtual machine, and database. It also requires mechanisms to assure confidentiality and access control. These may include encryption, obfuscation and key management as well as isolation and containment, robust log management, and an audit infrastructure. The security architecture provides the isolation, confidentiality, and access control required to protect end users' data and applications.

- Trustworthy cybersecurity assurance requires that the end users have confidence in the integrity of the complete cloud environment. This ranges from the physical data centers to the hardware and software, as well as the people and processes, employed by the CSP. This requires establishing an evidence-based trust architecture and control of the cloud environment provided by the CSP. It requires that the CSP provide adequate monitoring and reporting capabilities to assure the end user of transparency around security vulnerabilities and events. This should include audit trails that help the end user meet internal or external demands for provable security. A CSP should also deliver automated notification and alerts that support the end user's existing problem or incident management protocols so they can manage their total security profile most easily. All of these collectively help assure the end user of the operational quality and security of the CSP.
- Cloud governance requires the CSP to offer utilities that allow the end user to monitor their environment for security and other key performance indicators (KPIs) such as performance and reliability almost as well as they could in their own on-premises environment (or data center).

THINK IT THROUGH

Cybersecurity on the Internet vs. the Cloud

Think about cybersecurity for systems that use the Internet and compare that to cybersecurity for systems that access the cloud. Note that moving systems to the cloud requires robust cybersecurity, trustworthy cybersecurity assurance, and cloud governance as explained earlier in this section. Evaluate how these requirements differ from those set forth for systems that use the Internet to determine whether cybersecurity on the cloud can be handled in the same way as security on the Web.

Cloud Services Assurance

Because end users leverage various service types such as Infrastructure as a Service (IaaS) vs. Platform as a Service (PaaS) to create solutions in the cloud, CSPs and cloud end users share unique and overlapping responsibilities to ensure the security of services and sensitive data stored in public clouds. Shared responsibility considerations include threat detection, incident response, and patching/updating. An example of a PaaS service provided by cloud platforms today is IoT PaaS.

As noted earlier, there are lots of challenges faced by IoT cloud platforms that affects the TRM security and integrity/privacy assurance quality. IoT devices and data are vulnerable to various threats, such as cyberattacks, data breaches, unauthorized access, and malicious manipulation. These threats can compromise the functionality, integrity, and confidentiality of IoT systems, as well as expose sensitive and personal information of customers and users. IoT cloud platforms also need to adhere to the evolving regulations and standards that govern the collection, storage, and use of IoT data, such as the General Data Protection Regulation (GDPR) and the California Consumer Privacy Act (CCPA). This is why it is important to have a cloud security and trust architectures as well as cloud governance that provide robust security and privacy measures, such as encryption, authentication, authorization, monitoring, and compliance, that can protect IoT devices and data from end to end.

Another example of a PaaS service provided by cloud platforms today is big data analytics PaaS. As noted earlier and from a TRM assurance/security/privacy quality standpoint, security is clearly one of the major

concerns with big data analytics. Hacking and various attacks to cloud infrastructure do happen and may affect multiple clients even if only one site is attacked. To optimize making sense from the big data, organizations need to integrate parts of their sensitive data into the bigger data. To do this, companies need to establish security policies which are self-configurable. These policies must leverage existing trust relationships, and promote data and resource sharing within the organizations, while ensuring that data analytics are optimized and not limited because of such policies. This is why it is important to have a cloud security and trust architectures as well as cloud governance to mitigate risks using security applications, encrypted file systems, data loss software, and buying security hardware to track unusual behavior across servers.

One last example of PaaS service provided by cloud platforms today is cloud robotics PaaS. From a security standpoint, when robots are connected to the cloud, they are susceptible to hacking and cyberattacks. This can pose a serious risk to both the safety of robots and the privacy of the data that they are collecting. This is again why it is important to have a cloud security and trust architectures as well as cloud governance to help companies develop cloud-connected robots and invest in robust cloud security measures.

Cloud Applications Frameworks Assurance

Cloud application frameworks are fully managed by CSPs or designed to leverage IaaS/PaaS services on secure cloud platforms. A cloud server (e.g., AWS, GCP, Azure, IBM Cloud) can replace traditional application frameworks at the cost of migrating traditional applications that used these frameworks to the cloud, which is a costly and time-consuming proposition. For that reason, big tech application frameworks are now available on secure cloud platforms (e.g., IBM WebSphere Hybrid Edition, Oracle WebLogic Server for Oracle Cloud Infrastructure). Traditional database management systems/frameworks are easier to migrate to the cloud. For example, traditional database management systems such as MySQL, PostgresSQL, or SQL Server, can be migrated to Google Cloud SQL. Google also provides cloud-based NoSQL database systems (e.g. Firestone document database, Bigtable key-value database, Memorystore in-memory database). Similar cloud database systems/frameworks are available on Microsoft Azure (e.g., Azure SQL database) and other cloud platforms. Cloud support for application frameworks is not limited to database systems/frameworks. For example, Azure App Service is a cloud platform framework that may be used to securely host web applications, REST APIs, and mobile applications. More recently, VMware developed a cloud application server called Tanzu, previously Cloud-Foundry, that fully operates on the cloud and leverages the latest container management and cloud cybersecurity technology. IBM Bluemix is also derived from Cloud-Foundry. Furthermore, all social media platforms (e.g., Facebook, Twitter, TikTok) can be considered as examples of secure cloud application frameworks. Because there are no standards for using/implementing secure application frameworks on the cloud, end users need to consult the various CSPs' websites and stay up-to-date regarding the availability of such application frameworks.

Cloud Applications Assurance

Cloud applications are typically implemented on cloud servers, and developers take full advantage of established cloud security/trust architectures as well as cloud governance processes. Big tech organizations typically provide security best practices, models, and patterns to facilitate the creation of secure cloud applications on their own platform.

Cloud Information Assurance

In cloud computing, much of the large and critical databases are under the control of CSPs. These resources are located away from the end user's physical location, and often in physical locations unknown to the end user. The possibility, or even likelihood, of data being stored in other regions and countries, also requires meeting those region's legal and regulatory requirements for data protection. All this makes it more challenging to create trustworthy controls for the monitoring, governance, and auditing of the CSP environment. Therefore, as explained earlier, it is necessary to develop cloud security/trust architectures as

well as cloud governance processes prior to developing information systems on the cloud or migrating traditional/legacy information systems and their data to the cloud.

Cloud Assurance Methodologies

Various organizations provide cloud security best practices and cloud cybersecurity assessment methodologies. In particular, the Cloud Security Alliance (CSA) and the European Union Agency for Cybersecurity (ENISA) promote best practices developed for providing security assurance within cloud computing. The CSA Security Guidance provides fourteen domains of cloud security best practices. It is built on dedicated research and public participation, incorporating advances in cloud, security, and supporting technologies. The Security Assurance Methodology (SECAM), is a security assurance framework developed by the 3rd Generation Partnership Project (3GPP) specifically for network products used in mobile communications. Big tech organizations typically provide risk assessment methodologies geared toward using their cloud platform. Microsoft, for example, publishes a risk assessment guide for Azure. Other CSPs provide similar.

THINK IT THROUGH

Cloud Computing vs. Privacy

You are a software engineer, and you work for a company that provides open access to a mapping software for realtors. Your company merges with a real estate company that provides online services, such as sales and appraisals that your company did not previously do. As part of this new company, your boss wants you to take the customer database from the real estate company and add the personal information of homeowners in the local market to a private software package that lists all the houses in neighborhoods. As part of the new software, you are required to provide one-click access to this information from the mapping software. While creating these new features, you realize that the database of information contains fields such as social security number, mother's maiden name, and primary email addresses. The mapping software company will be stored in the cloud and shared with ten offices. All employees in the company will have access to the new software.

- What are some of your concerns?
- What are some security concerns?
- What recommendations can you make to help ensure the security of the PII?

Hardware Crisis

In October 2020, a warehouse fire severely damaged the Asahi Kasei Microdevices (AKM) semiconductor plant in Miyazaki, Japan. At the time, this was one of two RAM manufacturers worldwide. The global crisis for computer memory soon commenced shortly after this disaster. Hardware manufacturers had to slow the production of computers because the memory was not available for production, and the cost of the existing memory escalated. Japan's other chip manufacturing plants could not meet worldwide demand. Today, three manufacturing plants account for more than 90% of the world's RAM production. It seems that the world has not learned its lesson and diversified the production of this precious resource.

The other side of this story is that software advancements also stopped. New software packages are typically developed to incorporate newer technology features. If the newer computer manufacturing slowed down, the new innovative software packages also slowed down. This was not the same for cloud technologies. Because the resources are distributed and the hardware is not as essential, the cloud environment continues to thrive, and there has only been an increase in cloud usage since 2020.

Industry 4.0 Metaverse Smart Ecosystems Cybersecurity

While the metaverse and Web3 offer organizations new frontiers for customer engagement and business growth, these also create the potential for new cybersecurity risks that could lead to financial losses, brand and reputational damage, and legal challenges. These threats include system outages and disruptions because of data overload and threats that apply to other areas of computing, including ransomware and bots.

Smart Ecosystems Platforms Assurance

As noted earlier, Industry 4.0 smart ecosystems combine various platforms/services (e.g., 3-D Modeling, AR/VR, Edge Computing, Blockchain, AI/ML, and 3-D/4-D printing) and are typically deployed on top of a hybrid cloud/blockchain environment today. For example, they may use AI/ML platforms/services to support the rendering and management of realistic models of a 3-D world or digital twins. The various platforms/services operate within secure cloud/blockchain platforms that are managed using established cloud/blockchain security/trust architectures as well as cloud governance processes. Because the platforms/services can be assembled as mashups by combining offerings from multiple cloud platforms, it is necessary to consider the cybersecurity mechanisms discussed in Cloud Applications Frameworks Assurance in order to understand how to best secure cloud mashups. In this subsection, we briefly discuss the security vulnerability and defenses required when leveraging specific smart ecosystems platforms/services.

As artificial intelligence and other advanced technologies become more prominent and create supersocieties, we also face additional cybersecurity threats. For example, cybercriminals can use AI to leverage more sophisticated cyberattacks. At the same time, AI can be a tool against cybercrime, providing organizations with sophisticated technology to handle security tasks such as detecting suspicious activity in the system. In addition, AI can be used in testing, such as simulating system attacks to help cybersecurity professionals identify areas of risk and vulnerabilities that should be addressed. According to IBM, AI can help protect data in hybrid cloud environments with tools such as shadow data identification and monitoring for data abnormalities. AI can also create incident summaries and automate responses to these incidents, improving investigations and outcomes. AI's ability to analyze login attempts and verify users can reduce fraud costs by as much as 90%.[39]

TECHNOLOGY IN EVERYDAY LIFE

Using AI in Cybersecurity

AI creates new cyber threats but also provides additional tools to improve cybersecurity. Think about how AI is used in everyday life and consider AI as both a threat and a tool in cybersecurity.

Provide a few scenarios illustrating the benefits and drawbacks of AI-driven cybersecurity (e.g., monitoring and analyzing behavior patterns, preventing bad actions and outcomes) and explain your opinion.

Metaverse Smart Ecosystems Platform Cybersecurity Assurance Methodologies

To protect against cybersecurity threats in the metaverse, organizations should use many of the same security measures to protect against cyber threats on the Internet. They also need to address new cybersecurity risks that could lead to financial losses, brand and reputational damage, and legal challenges.

Figure 14.53 illustrates the typical four layers of metaverse platforms along with eight major threats in virtual world of the metaverse.

39 IBM. 2024. "Artificial Intelligence (AI) Cybersecurity." https://www.ibm.com/ai-cybersecurity#:.

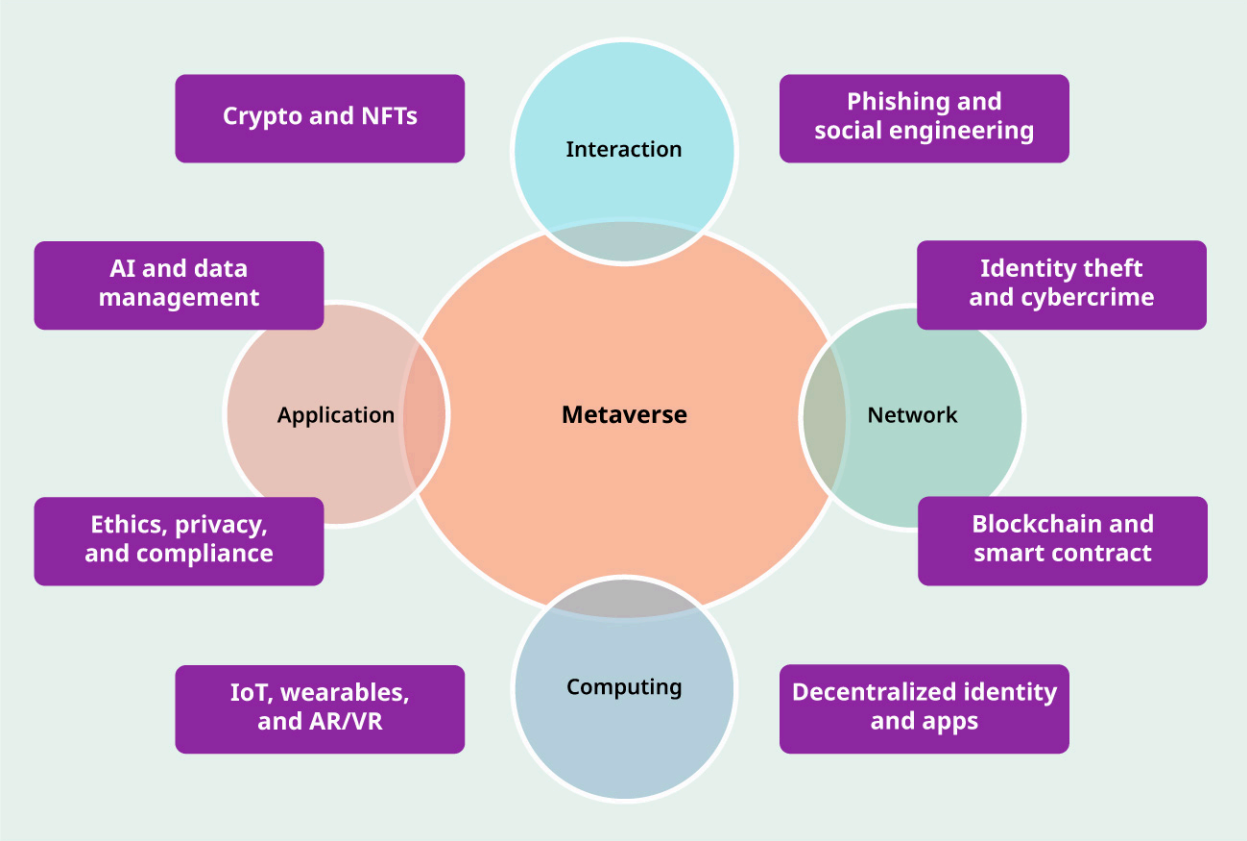


Figure 14.53 The metaverse has four platform layers that generally face eight categories of cyber threats. (attribution: Copyright Rice University, OpenStax, under CC BY NC-SA 4.0 license)

Industry 5.0 Supersociety Solutions Cybersecurity

In addition to the smart ecosystem services mentioned in the previous subsection, Industry 5.0 supersociety solutions combine various platforms and technologies such as autonomous systems platforms, advanced robotics platforms, nanotechnology, super compute, and autonomous super systems platforms. These solutions are typically deployed on top of a hybrid cloud/blockchain environment today. Supersociety platforms enable AI-powered robots, and supersociety technologies (e.g., nanotechnology, super compute) make it possible to improve existing services to further enable smart ecosystems services. The various supersociety platforms operate on top of secure cloud/blockchain platforms that are managed using established cloud/blockchain security/trust architectures as well as cloud governance processes. In general, because the underlying services that are part of these platforms can be assembled as mashups by combining services from multiple cloud platforms, it is necessary to consider the cybersecurity mechanisms discussed in Cloud Applications Frameworks Assurance in order to understand how to best secure cloud mashups. In this subsection, we briefly discuss the security vulnerability and defenses required when leveraging specific supersociety platforms and technologies.

Supersociety Autonomous Systems Platform Assurance

Autonomous systems platforms are an essential component of the future of artificial intelligence. They provide the tools and frameworks for building, testing, and deploying autonomous systems (e.g., self-driving cars, drones) that can operate in a variety of environments. However, there are various data security/privacy, regulatory challenges, and ethical concerns associated with autonomous systems platforms. The fact that autonomous systems rely on large amounts of data raises concerns around data security and privacy. As was the case for smart ecosystems AI/ML platforms discussed earlier, which face the same type of issues, it may not always be possible to simply rely on the establishment of cloud security/trust architectures and cloud governance processes. Using third-party tools to address the lack of scalability of a CSP's cloud platform may

introduce vulnerabilities and requires additional defense mechanisms. The deployment of autonomous systems is also subject to various regulations and standards that are not typically covered by cloud security/trust architectures and governance processes. Therefore additional security architecture components and processes will need to be researched and provided based on the domain of application of the autonomous platform. Finally, autonomous systems raise ethical concerns around issues such as accountability, transparency, and bias. These aspects should be covered in the cloud security/trust architectures and cloud governance processes.

Supersociety Advanced Robotics Platform Assurance

Supersociety advanced robotics platforms provide the tools and frameworks for building, testing, and deploying AI-powered robots (e.g., cyborgs, swarmbots) that can work alongside humans. The technical challenges associated with securing advanced robotics platforms are analogous to those of securing autonomous systems platforms, which were covered in Supersociety Autonomous Systems Platform Assurance. Refer to that discussion to review the security challenges and associated defenses that must be put in place. In addition to these technical challenges, there are also allied social, legal, and ethical issues for seamless integration of humanoids into our societies. There is a lot of research focused on this aspect today. One question is whether there should be special laws to govern robots.

Supersociety Nanotechnology Platform Assurance

Nanotechnology is one of the supersociety technologies that make it possible to improve existing services to further enable smart ecosystems services and support supersociety platforms. The emergence of nanotechnology presents an entirely new set of potential risks, as well as potential solutions to cybersecurity. Because nanotechnology involves the manipulation of matter on an atomic or molecular scale, it is not too far-fetched to think that it could enable the development of “smart” materials that could detect and react to malicious software or threats. Nanotechnology could also enable the creation of tiny sensors that could detect unauthorized access to networks or data. The use of nanotechnology in cybersecurity could provide users with a greater level of privacy and security. For example, nanomaterials could be used to create encryption keys that are much more difficult to crack than current methods. In addition, the use of nanotechnology could make it easier to detect and prevent data breaches. However, the use of nanotechnology also presents some potential risks. For example, the use of nanomaterials could create new vulnerabilities that could be exploited by malicious actors. Additionally, the use of nanotechnology could lead to the creation of devices or systems that are too complex for humans to understand or control.

One of the most promising applications of advanced materials in cybersecurity today is the use of graphene. This two-dimensional material, which is composed of a single layer of carbon atoms, has many properties that make it ideal for security applications. It is highly conductive, strong, and lightweight, and is impermeable to many substances. Graphene has already been used in a variety of devices, including computer chips, touchscreens, and RFID tags, and its potential for cybersecurity applications is vast. Another application of advanced materials in cybersecurity is the use of nanomaterials. Nanomaterials, such as nanotubes and nanowires, are incredibly small, making them difficult to detect. This makes them ideal for use in encryption and authentication systems, where the smallest of details can make all the difference. Additionally, nanomaterials can be used to develop new types of sensors that can detect intrusions and unauthorized access attempts.

Overall, nanotechnology has the potential to revolutionize the world of cybersecurity, both in terms of the solutions it offers and the risks it creates. As this technology continues to develop, it is important that the security industry works to ensure that the benefits of nanotechnology are maximized while minimizing the risks. This appears to be the only way to implement supersociety nanotechnology platforms assurance.

Supersociety Supercompute Platform Assurance

Supercompute is yet another supersociety technology that makes it possible to improve existing services to

further enable smart ecosystems services and support supersociety platforms. It is also a technology that has the potential to revolutionize the world of cybersecurity, both in terms of the solutions it offers and the risks it creates. With respect to risks, the vastly increased processing speed associated with the use of quantum or neuromorphic computers will definitely have an impact on some of the cryptography algorithms that are in use today. In particular, while symmetric key encryption and collision-resistant hash functions are considered to be relatively secure against attacks by quantum computers, signature schemes based on the integer factorization problem, the discrete logarithm problem, or the elliptic curve discrete logarithm problem can be solved with Shor's algorithm with a quantum computer that is powerful enough. On the positive side, noninteractive ZKPs that only collision-resistant hash functions are plausibly post-quantum secure and can be used to replace traditional signature schemes based on public-key cryptography that are not quantum-resistant. Supercompute will also improve the speed of cryptographic computations that are needed to operate secure platforms such as blockchain and further optimize the verification of the transactions. Again, as this technology continues to develop, it is important that the security industry works to ensure that the benefits of supercompute from a cybersecurity standpoint are maximized while minimizing the risks. This appears to be the only way to implement supersociety supercompute platform assurance.

Supersociety Autonomous Supersystems Platform Assurance

The technical challenges associated with securing autonomous supersystems are analogous to those of securing autonomous systems platforms, which were covered in Supersociety Supercompute Platform Assurance. Refer to that discussion to review the security challenges and associated defenses that must be put in place. The only difference is the fact that supersystems will make use of new supersociety technologies that are emerging such as nanotechnology and supercompute. Given the ethical and social usability challenges (from a TRM quality standpoint), there are growing concerns that the combined use of the various supersociety technologies described earlier to power autonomous supersystems could cause threats to humanity and future civilizations. It will therefore be important for the security industry to ensure that the benefits of these technologies are maximized while minimizing the risks. This appears to be the only way to implement supersociety autonomous supersystem platform assurance.

14.3 Governing the Use of Cyber Resources

Learning Objectives

By the end of this section, you will be able to:

- Understand cyber economics
- Relate to responsible computing
- Understand how cyber economics and responsible computing apply to Internet web/mobile solutions
- Understand how cyber economics and responsible computing apply to cloud solutions
- Understand how cyber economics and responsible computing apply to smart ecosystems solutions
- Understand how cyber economics and responsible computing apply to supersociety solutions
- See what cyber economics and responsible computing means to supporters and careers in IT

Optimizing the quality of cloud-based solutions, smart ecosystems, and supersociety solutions is quite difficult. The previous section focused on cybersecurity assurance and illustrated how difficult it is to protect IT solutions against undesirable use. In this section, we will look at cyber economics and understand the importance of responsible computing.

Cyber Economics

The sectors of the economy driven by digital information and the need for cybersecurity are referred to as **cyber economics**. This includes the risks of online economic transactions and the need for regulatory oversight to govern cybersecurity and cyber economics.

In cyber economics, at least three crucial aspects of cybersecurity require policy and legislation to help mitigate risks. The first is online **identity theft**, which refers to the illegal possession and use of an individual's PII. Identity theft is the primary way cybercriminals steal money from consumers. The second is **industrial espionage**, which is the process of spying on an organization to steal trade secrets. The third is **critical infrastructure**, which refers to the network of utilities, roadways, railroads, and buildings necessary to support our transportation, commerce, and other systems vital to sustain daily life.

Organizations worldwide, including government agencies, are concerned about cyber economics and the associated risks. The U.S. Department of Homeland Security's Cyber Risk Economics project, under the Science and Technology Directorate, supports research to study cyber economics and look at the vulnerabilities and existing, as well as needed, controls.[40] The International Monetary Fund also published a paper examining cyber threats worldwide and concluded that the global financial system is facing increasing cyberthreats, and global cooperation is required to manage the threats.[41] Regrettably, thus far, the U.S. and other nations have taken limited actions to address cyber economics risks.

GLOBAL ISSUES IN TECHNOLOGY

Global Cyber Strategies

The Center for Strategic and International Studies maintains an index of each country and territory's global cyber strategies. The strategies cover overarching national doctrines, military strategies, digital content regulations, privacy laws, critical infrastructure strategies, commerce laws regarding internet services, and strategies/regulations regarding cybercrime.

Of 253 countries and territories in the world, 114 have guidance for commerce and 113 for privacy; 91 countries and territories cover crime, while 78 have national overarching strategies, and critical infrastructure is addressed by 63. The military is a focus for 31 countries and territories, while 35 have digital content regulations. The countries and territories with strategies in all areas is a short list that includes China, France, Germany, and Russia. The United States does not have digital content regulations, but all other areas are covered to some extent.

Responsible Computing Basics

Despite the lack of guidance from governments and other institutions, organizations can use **responsible computing**, a systemic approach addressing current and future challenges in computing, including sustainability, ethics, and professionalism, to help protect against cyber threats. On May 10, 2022, the responsible computing consortium set forth a definition of responsible computing created by the Object Management Group (OMG) with IBM and Dell as founding members.

LINK TO LEARNING

You can read the definition of responsible computing (https://openstax.org/r/76computing) from OMG, IBM, and Dell.

Although the initial idea of a responsible computing ladder rapidly turned into the familiar hexagon diagram framework (refer to Figure 14.2), the core principles and guiding questions have remained the same throughout its development and implementation.

40 U.S. Department of Homeland Security, Science and Technology, "Cyber risk economics," August 2, 2024. https://www.dhs.gov/science-and-technology/cyrie

41 T. Maurer and A. Nelson, "The global cyber threat," *International Monetary Fund, Finance and Development*, 2021. https://www.imf.org/external/pubs/ft/fandd/2021/03/global-cyber-threat-to-financial-systems-maurer.htm

For several years, IBM's Academy of Technology (AoT) has worked on various aspects of Responsible.Computing(), detailing the framework dimensions, validating initial ideas and concepts with the client council, and jointly developing the Responsible.Computing() manifesto. The manifesto—the first outcome and deliverable—includes the six values of technologies and innovations, exclusive systems, data, conscious code, efficient use, and data centers. It also includes the principles of sustainability, circularity, openness, inclusivity, authenticity, and accountability.

Responsible computing is a systemic approach addressing current and future challenges in computing, including sustainability, ethics, and professionalism. It stems from the belief that we need to start thinking about our organizations differently regarding their impact on people and the planet. Some examples of responsible computing include designing data centers with a focus on efficiency and sustainability, emphasizing green energy and improving the handling and disposal of chemicals, toxic materials, and rare metals, responsible data usage providing high-quality inputs for computing systems, taking a holistic approach to decision-making, among others. Following is a discussion of each pillar of responsible computing.

THINK IT THROUGH

Global Cooperation on Cyber Economics and Cybersecurity

A major stumbling block to implementing responsible computing is the lack of cooperation and coordination among the world's governments and institutions. How can we resolve this? Who should be in charge of regulating cyber economics and global cybersecurity efforts? Why? How do we develop a global plan and implement regulations that benefit everyone equally worldwide?

Responsible Data Center

The first pillar of responsible computing is responsible data centers, which should be designed and operated with an emphasis on sustainability. They rely on a technology infrastructure that emphasizes green energy, focusing on reducing technology's carbon footprint. This includes using green energy sources and tracking the energy required for cooling. Responsible data centers also strive to minimize water usage.

Responsible Infrastructure

The second pillar of responsible computing is responsible infrastructure, which considers the physical resources needed for a system, including hardware, software, and other network components. The infrastructure is designed to use as little energy as possible while relying on more sustainable components, with less waste going to landfills.

Responsible Code

With responsible code, the third pillar of responsible computing, organizations make conscious code choices that optimize environmental, social and economic impacts over time. This includes practices such as ensuring that code is efficient and results in fewer HTTP requests and smaller page sizes.

TECHNOLOGY IN EVERYDAY LIFE

Dealing with Ethical Programming Issues

Try it yourself:

You have achieved your goals. You have secured an entry-level programming position at TechWorks. Your first day on the job, your boss brings you in the project development meeting, and the new software package is set to be finalized this week. As a new team member, you are asked to be part of the quality

assurance team and test the software of functional performance and are given the test plan. You are told that there are four levels of bug reporting:

- Bug level 1: Critical functionality is missing or not present in the software
- Bug Level 2: Critical functionality is not working correctly
- Bug Level 3: Non-critical functionality is either not present or is not working properly
- Bug Level 4: Cosmetics and other user functionality issues

During your analysis, you found several Bug Level 3 issues and several Bug Level 4 issues that were all programmed by one programmer at the company. When you bring this up to the lead developer of the team, they say, "that's OK, the customer didn't expect that part of the software to work anyways. We will be able to bill them for more hours to fix the software." You know from the development meeting that if the product doesn't ship, your company will be liable for a breach of contract. How do you handle reporting this to your supervisor? What suggestions do you have for fixing the problems? What lessons can you learn from this type of experience for future use?

Responsible Data Usage

Responsible data usage is the fourth pillar of responsible computing, and this relates to using data securely in ways that drive transparency, fairness, and respect for the users. Initiatives like Data Augmentation for Discrimination Prevention and Bias Disambiguation, Social Media and Freedom of Speech, and General Data Protection Regulation (GDPR) are related efforts that try to promote responsible data usage by ensuring that such usage is not discriminatory and does not cause harm to users.

Responsible Systems

The fifth pillar of responsible computing is responsible systems, which are inclusive systems that address bias and discrimination to promote equality for all, regardless of personal characteristics such as race, age, gender, and disability. This relates to parallel efforts being conducted in areas like explainable AI, large language models regulation, and human-centered computing in the metaverse, which strive to ensure that algorithms are based on trustworthy data. This includes efforts by the World Economic Forum to transform from Society 4.0 to Society 5.0 (Figure 14.54).

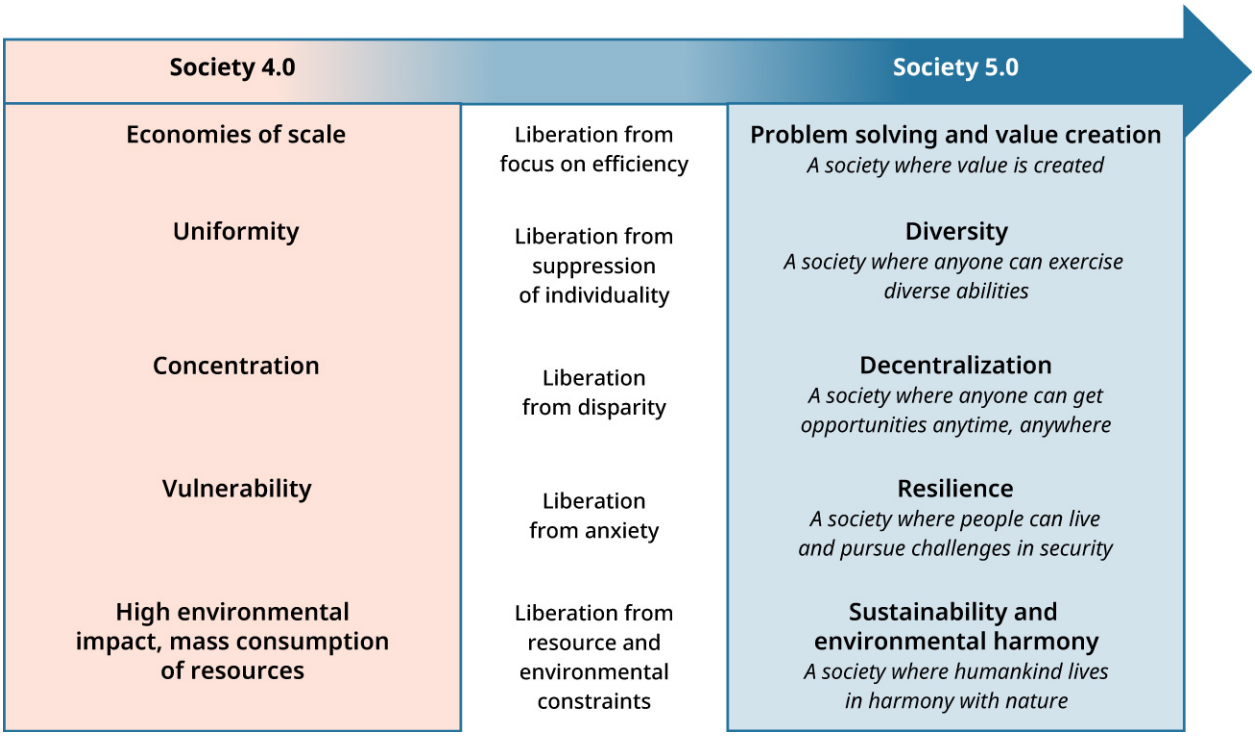


Figure 14.54 The World Economic Forum's goals are to create a society that enables anyone from any background to use their abilities to create value. (attribution: modification of work from "Social Media Usage by Different Generations as a Tool for Sustainable Tourism Marketing in Society 5.0 Idea" by B. Hysa, et al./Sustainability, CC BY 4.0)

CONCEPTS IN PRACTICE

From Society 4.0 to Society 5.0

Transforming from Society 4.0 to Society 5.0 is more than just words on paper. Japan is actively striving to create Society 5.0, with some objectives slated to be reached by 2030. This includes providing clean and sustainable energy, making advanced health care available to everyone, and optimizing food production and availability. While it remains to be seen whether Japan will achieve these and other goals, the nation is progressing.

Responsible Impact

The final pillar of responsible computing is responsible impact, which refers to using technologies and innovations that drive positive impact for society and making efficient use of available and future technology. This is related to the World Economic Forum's goals for Society 5.0/Industry 5.0, which considers the fact that while technology can bring about improvements, such as higher standards of living and greater convenience, it also can have negative effects, such as detrimental impacts on employment, as well as growing disparity and unequal distribution of wealth and information.

Cyber Economics and Responsible Computing for Internet Web/Mobile Solutions

Taking shortcuts for cyber economics and other reasons typically causes harm to society and the planet. This applies to any networked software solution when organizations take shortcuts to limit investments in software security, common cyber threat defenses, and software solutions assurance methodologies. Internet solutions are particularly affected by organizations taking shortcuts in the various layers of their solutions to limit costs or other quality enforcement measures at the infrastructure, application frameworks, applications, and

information levels. In addition, responsible computing requires that when organizations use the Internet, they should do things like verify information before they post it and make every effort to protect users' privacy.

Cyber Economics and Responsible Computing for Cloud Solutions

When organizations operate in the cloud, responsible computing should guide their efforts to ensure they have responsible data. Particularly since the cloud is often used to share data, organizations must take particular care to protect data privacy while maintaining transparency.

Cyber Economics and Responsible Computing for Smart Ecosystems Solutions

Organizations often rely on third-party providers when using smart ecosystems solutions that involve platform services to support activities such as 3-D modeling, AR/VR, IoT, blockchain, AI/ML, and 3-D/4-D printing. It can be tempting to take shortcuts in implementing these services to limit costs or other quality assurance measures, such as software security, cyber threat defenses, and metaverse smart ecosystems platforms' cybersecurity assurance methodologies. To comply with responsible computing expectations, organizations should make every effort to be diligent about cybersecurity.

INDUSTRY SPOTLIGHT

Responsible Computing in Health Care

Responsible computing is important in every industry today. Can you elaborate on how useful it is in the health-care industry specifically? (*Hint*: Think about the combined use of IoT, ML, and XR to analyze and diagnose patients' medical issues and their possible negative impact and how responsible computing can help.)

Cyber Economics and Responsible Computing for Super-Society Solutions

Finally, responsible computing should be a priority in super-society solutions that involve platform services supporting autonomous systems, advanced robotics, nanotechnology, super-compute, and autonomous super-systems. High-performance computing (HPC), AI, and quantum computing systems have the potential to help humanity make progress against some of the most complex scientific problems facing the planet, including climate change, food insecurity, and new treatments for diseases. Unlocking the full potential of these computing technologies requires global collaboration among public and private institutions with responsible use at the center of their development and deployment. The recently launched Open Quantum Institute or OQI is an example of the importance of global collaboration bridging science, diplomacy, industry, and non-governmental organizations (NGOs) to realize this vision of inclusivity in technology and its applications. The responsible development of nanotechnology tackles environmental, health, and safety implications of nanotechnology and ethical, legal, and societal implications of nanotechnology—and embraces new ideas, including an emphasis on inclusion, diversity, equity, and access, the responsible conduct of research, product stewardship, and the circular economy.

What Cyber Economics and Responsible Computing Means to Supporters and Careers in IT

Responsible computing is creating jobs as organizations recognize the need for change. This includes positions in computer science and information security and jobs in law, public policy, and other disciplines that support responsible computing. An important part of responsible computing is ethics. To be responsible, organizations must be ethical, and some organizations offer positions focused on ethics in technology, such as the ethics of artificial intelligence.[42]

LINK TO LEARNING

As more organizations embrace responsible computing, more resources are available to help organizations as they implement responsible computing policies and procedures. For example, IBM and Dell founded the Responsible Computing membership consortium (https://openstax.org/r/76consortium) managed by the Object Management Group, which offers a variety of resources.

42 D. R. Polgar, "How to build a career in responsible tech," *Built In*, March 16, 2021. https://builtin.com/articles/responsible-tech-careers

Chapter Review

Key Terms

access control process of regulating the people and devices that can use a computer system's resources
adaptability ability to change or modify the current system to meet the needs of a different industry requirement
advanced persistent threat (APT) intruder or group of intruders infiltrate a system and remain undetected while leaving the networks and systems intact, allowing the intruder to spy on business activity and steal sensitive data while remaining undetected
affordability ability to create a system that is cost-efficient, not only monetarily but also with resource usage
anonymity being able to interact on the Internet, even publicly, while concealing your identity
application security provides processes that help protect applications operating on-premises and in the cloud
autonomous system system that can operate with limited human control
composability ability to incorporate services within applications
container lightweight package that bundles together applications to form a solution to specific problems
critical infrastructure network of utilities, roadways, railroads, and buildings necessary to support our transportation, commerce, and other systems vital to sustain daily life
cyber economics sectors of the economy driven by digital information and the need for cybersecurity
cybersecurity policies, procedures, technology, and other tools, including people on which organizations rely to protect their computer systems and information systems environments from digital threats
cybersecurity assurance confidence that every effort is made to protect IT solutions against undesirable use
data security platform automates the proactive protection of information via monitoring and detecting data vulnerabilities and risks across multiple environments, including hybrid and multicloud platforms
distributed denial-of-service (DDoS) attack overloading a server with traffic in an attempt to crash a server, website, or network; usually occurs from multiple coordinated systems
evolvability ability to adapt the system to new standards and practices
extensibility ability to modify the system to include new requirements or remove old requirements that are no longer needed
identity and access management (IAM) roles and access privileges for each user, as well as the conditions under which they are granted or denied their privileges
identity theft illegal possession and use of an individual's PII
ilities "abilities" of architectural properties
industrial espionage process of spying on an organization to steal trade secrets
information security protecting the data, digital files, and other information maintained in a system
infrastructure security practices for protecting the computer systems, networks, and other assets that society relies upon for national security, economic health, and/or public safety
insider threat threat posed by current or former employees, partners, or contractors who misuse their access; can also include vulnerabilities intentionally created by programmers as malware
interoperability ability for two or more computers or processes to work together
malware malicious software variants—such as viruses, worms, Trojans, spyware, and botnets—that provide unauthorized access or cause damage to a computer
man-in-the-middle eavesdropping attack that allows cybercriminals to intercept communications between two parties in order to steal data, often on unsecured Wi-Fi networks
nanotechnology studies and manipulates atoms and molecules to support advancements in energy, medicine, and other fields
network security security measures for protecting a computer network from intruders, including both wired and wireless (Wi-Fi) connections
nomadicity ability to work in a self-contained environment or the ability to move the system from location to

location, when system location is a requirement

non-repudiation proof of the origin, authenticity, and integrity of data

Open Web Application Security Project (OWASP) launched in 2001 with the purpose of securing web applications

password secret string of characters used to gain entry into a system

phishing form of social engineering that tricks users into providing personal information through fake emails or text messages posing as legitimate companies

privacy process of keeping your actions online concealed from the public, such as messages intended only for certain individuals

ransomware malware that encrypts data and demands a ransom to unlock or prevent data exposure

reliability ability of the system to perform as needed and to specification

responsible computing systemic approach addressing current and future challenges in computing, including sustainability, ethics, and professionalism

scalability ability to enhance or retract system requirements for the number of users involved in the system

security information and event management (SIEM) practice that focuses proactively on the automated detection and remediation of suspicious user activities based on the analysis of security events

software security manner through which software safeguards system resources, including data, to provide access to only authorized users

supersociety environment that is technologically rich

survivability ability to survive an attack or disruption of service within a system

tailorability ability to customize the system for the needs of the users or industry

Technical Reference Model (TRM) framework that details the technologies and standards used to develop a system and deliver services

understandability (also: learning curve) ability of the system to be used

virtual local area networks (VLANs) virtual local area networks that connect devices and nodes from various LANs

walled garden approach limits openness and prevents users from having access to a platform; conflicts with the intentions of the Open Web Platform

zero-knowledge proof (ZKP) cryptographic system that functions as a useful tool to protect privacy

Summary

14.1 Cyber Resources Management Frameworks

- Cyber refers to anything relating to computers or information technology. At the same time, cyber resources are cyber tools, platforms, and solutions that store, process, and manage data and other assets electronically and make them available via networks, including the policies and procedures for handling cyber.
- The qualities that comprise cyber resources include security, safety, performance, usability, reliability, and autonomy.
- The security and control of a company start with an information security policy, which should outline security practices for employees and systems and apply to all architectural buckets in an organization, including its business, applications, data, pyramid of knowledge layers, and infrastructure.
- To create an information security policy, organizations need the Technical Reference Model (TRM), which applies to their needs.
- In 2022, IBM and Dell formed the Responsible Computing Framework, a systematic approach to design and development that addresses the soft skills needed in the industry. The framework stresses the developer to be wary of potential harm from their development. It puts a lot of responsibility onto the developer to know how to code securely and properly develop systems with an adversarial mindset.
- Traditional architectural styles like OMA and SOA are generic models that can be applied to the TOGAF TRM.
- The OMA applies to creating and assembling components that can be used for information systems,

including interfaces. At the same time, the SOA relates to the assembly of services that can be applied to the TOGAF TRM.

- The TOGAF and the TRM provide a good framework for analyzing most system challenges.
- Different architectural styles have been used for different project managerial styles. Different software development models like the waterfall model and the Agile development style exist.
- Cyber resource qualities are developed and measured within software models. The ISO standard 35733, which is part of the ISO/IEC 25010 standard, details different software quality models and provides a strong definition for the "ilities."
- The OMA RM, OMA Guide (OMG), and TOGAF TRM help guide functional and non-functional requirements by creating a taxonomy of service qualities. TOGAF recommends a combination of quantitative and qualitative methodologies to measure any system properly.
- Web platforms have had to change continuously in direction and architectural design. Knowing that the Internet had to change was a design principle.
- The biggest challenge to the modern web is the growth and variety of technology found on the Internet.
- A major challenge for the workforce is the lack of qualified professionals in the right position.
- The threat landscape is a significant challenge. The Web platforms and technology are growing at alarming rates. The use of cloud technology provides its own set of challenges, as well as the lack of engineers who can adequately secure those environments.

14.2 Cybersecurity Deep Dive

- Cybersecurity refers to the policies, procedures, technology, and other tools, including people, on which organizations rely to protect their computer systems and technological environments from digital threats. Cybersecurity focuses on five categories of security: network, application, critical infrastructure, IoT, and cloud.
- In 2023, the average cost of a data breach was $4.45 million globally, which, in just three years, was a 15% increase over 2020. Global cybercrime financial damage likely will reach $10.5 trillion by 2025.
- Cybersecurity domains include infrastructure, network, application, and information security.
- An important pillar of cybersecurity assurance is nonrepudiation, which is achieved through cryptography.
- Beyond basic securing of information systems, cybersecurity requires the creation and governance of processes that protect organizations and individuals against costly breaches. End-user education, disaster recovery/business continuity planning, and data storage are critical parts of this process.
- Cybersecurity must include tools and procedures for responding to unplanned events—such as natural disasters, power outages, or cybersecurity incidents—with minimal disruption to key operations.
- Cybersecurity must include data storage protection measures that promote data resilience with safeguards, including encryption and immutable and isolated data copies that can quickly be restored to recover data and minimize the impact of a cyberattack.
- In 2001, the Open Web Application Security Project (OWASP) was launched to secure web applications. The controls were focused on securing the risks involved with the development and deployment of the applications.
- Evolvability is an important cyber quality, but the evolution of platforms on which information systems may be deployed creates a need for new security measures.
- Not all cybercriminals are outsiders. Many cybersecurity breaches result from malicious insiders working for themselves or in concert with outside hackers.
- The cybersecurity risk surface is expanding, with thousands of new vulnerabilities reported in old and new applications and devices. The opportunities for human error (specifically by negligent employees or contractors who unintentionally cause a data breach) continue to increase.
- Attack vectors are not contained, and cybercriminals constantly find new attack vectors via Linux operating systems, operational technology (OT), IoT devices, and cloud environments.
- Never assume that the industry where you work is safe. Every industry has its share of cybersecurity risks, and cyber adversaries exploit the necessities of communication networks within almost every government

and private sector organization.

- Common cyber threats include malware, ransomware, phishing, insider threats, distributed denial-of-service (DDoS) attacks, advanced persistent threats (APTs), and man-in-the-middle attacks.
- Key cybersecurity technology and associated best practices typically fall under three categories: identity and access management (IAM), a comprehensive data security platform, and security information and event management (SIEM).
- A comprehensive data security platform protects sensitive information across multiple environments, including hybrid multicloud environments. The best data security platforms provide automated, real-time visibility into data vulnerabilities and ongoing monitoring that alerts them to data vulnerabilities and risks before they become data breaches; they should also simplify compliance with government and industry data privacy regulations. Backups and encryption are also vital for keeping data safe.
- An essential tool to secure information systems is cryptography, which encrypts information and makes it accessible only to those who are authorized to decrypt and use the information. Cryptography can help ensure properties such as confidentiality (i.e., secrecy, privacy), integrity (i.e., tamper resilience), authenticity, availability, and nonreputability (or deniability).
- Authentication, passwords, and access control are critical tools in cybersecurity. The three most common access control designs include mandatory access control (MAC), discretionary access control (DAC), and role-based access control (RBAC).
- Protecting anonymity and privacy is an important aspect of cybersecurity.
- To protect systems, software solutions architects and developers must consider security as a property of the systems they build. Security should be part of the software design process to proactively approach cyber threats and risks.
- Cybersecurity is vital to protecting organizations as they do business via the Internet, in the cloud, and through the metaverse.
- Cybersecurity policies and procedures must cover mobile devices, laptops, and other system components.
- Smart ecosystems and supersociety solutions are subject to additional cybersecurity threats.

14.3 Governing the Use of Cyber Resources

- Cyber economics refers to the sectors of the economy driven by digital information and the need for cybersecurity. This includes the risks of online economic transactions and the need for regulatory oversight to govern cybersecurity and cyber economics.
- In cyber economics, at least three crucial aspects of cybersecurity require policy and legislation to help mitigate risks, including online identity theft, industrial espionage, and critical infrastructure.
- Organizations can use responsible computing to help protect against cyberthreats. The pillars of responsible computing are responsible data centers, responsible infrastructure, responsible code, responsible data usage, responsible systems, and responsible impact.
- Responsible computing can be used for solutions on the Internet, in the cloud, and in super-societies.
- As more organizations implement responsible computing policies and procedures, jobs are being created for individuals in computer science and information security and related fields like law and public policy that support responsible computing. These jobs often focus on ethics, an important part of responsible computing.



Review Questions

1. The acronym TRM, refers to what?
   a. Technical Reference Model
   b. Tactical Reference Model
   c. Technical Requirements Model
   d. Tactical Requirements Model

2. What part of the domain is more directed to technology of the IBM Responsible Computing Framework?
   a. responsible impact

b. responsible data usage
c. responsible infrastructure
d. responsible systems

3. The OMA applies to the creation and assembly of what?
a. components that assemble services
b. components that can be used for networking backends
c. components that communicate with databases
d. components that can be used for interfaces

4. The OMA-RM was created to detail communications and what other service for request brokers?
a. interactions
b. solicitations
c. cloud technologies
d. cyber resources

5. The ability to adapt the system to new standards and practices is known as what?
a. tailorability
b. evolvability
c. extensibility
d. scalability

6. What ability is also referred to as the learning curve?
a. nomadicity
b. performance
c. adaptability
d. understandability

7. The TOGAF model is designed to provide several techniques for what process?
a. quality assessment
b. requirements gathering
c. modeling
d. compliance testing

8. The concept of a walled garden approach to web design limits the number of what?
a. colors and shapes
b. controls and plug-ins
c. design layouts
d. third-party software packages

9. Cloud-centric technologies face a lot of challenges, but what is the biggest challenge?
a. interoperability
b. usability
c. scalability
d. portability

10. Containers provide a quick solution to load different features into what?
a. an instance
b. a wall garden approach to web design
c. a software package
d. an operating system

11. What does Kubernetes specialize in?
a. web design

b. network solutions
c. enterprise solutions
d. containers

12. What is a big challenge to big data analytics from a TRM perspective?
a. cost
b. resources
c. time
d. vendors

13. The largest challenge to IoT as a PaaS is what?
a. scalability
b. adaptability
c. maintainability
d. performance

14. Backhauling refers to what?
a. back-up systems for networking
b. load balancing for cloud solutions
c. directing network traffic in a longer, out-of-the-way route
d. reverse engineering a software package

15. What are non-technical countermeasures in cybersecurity?
a. network applications, critical infrastructure, and IoT
b. laws, policies, procedures, training, and auditing
c. confidentiality, integrity, and service availability
d. anonymity, authenticity, and assurance

16. What cybersecurity domain focuses on protecting the data and digital files maintained in a system?
a. network security
b. application security
c. information security
d. infrastructure security

17. As a hacker, you have decided that the best way to achieve your goals is to add a hidden feature or command to a program that will enable you to perform unauthorized actions. What type of malware will you use to achieve this?
a. Trojan horse
b. rootkit
c. worm
d. backdoor

18. You are designing a website that will encourage your customers to share opinions, and you want to ensure their civil liberties are protected while keeping their personal data safe from hackers. What aspect of cybersecurity are you concerned about?
a. privacy
b. access control
c. anonymity
d. role-based control

19. What process should you use to ensure that Wi-Fi networks are hidden and secured?
a. encryption
b. malware
c. zero knowledge proofs

d. pseudonymity

20. In cyber economics, what do we call the network of utilities, roadways, railroads, and buildings necessary to support our transportation, commerce, and other systems?
a. industrial infrastructure
b. responsible data
c. critical infrastructure
d. responsible systems

21. What pillar of responsible computing is concerned with using technologies and innovations that make efficient use of available and future technology?
a. responsible impact
b. responsible systems
c. responsible infrastructure
d. responsible code

22. In responsible computing, what does it mean to have a responsible system?
a. The system uses little energy and strives for sustainability.
b. The system is inclusive and promotes equality for all.
c. The system emphasizes the use of green energy.
d. The system produces efficient code to have a positive impact on the environment.

Conceptual Questions

1. In IBM's Responsible Computing Framework, please compare the areas of responsible infrastructure and responsible systems. Give at least three differences or three similarities.

2. Cyber resource quality requirements are a theme all throughout the chapter. Summarize what you have learned and apply it to the TRM.

3. The TOGAF Reference Model is a resource that can be used to define application platform frameworks. Explain the relationship between the application platform interface, the qualities, and the operating system services.

4. As previously discussed, the "ilities" are part of the OMA-NG framework. Explain the concepts of tailorability and evolvability and what differentiates them.

5. Modern web/mobility challenges include ease-of-use. This is slightly different from adaptability, which includes scalability and flexibility. Explain the differences between ease-of-use and adaptability.

6. Explain what a container is and how it enhances web/mobile platforms.

7. PaaS is a standard acronym for Platform as a Service. Explain how cloud PaaS works to enhance data analytics. Explain how PaaS can help with the TRM performance quality.

8. No security mechanism is free; what are the direct and indirect costs associated with implementing security mechanisms?

9. You are training a colleague to handle your organization's software security. What key points will you share with this colleague?

10. In cloud-centric solutions for cybersecurity, the Cloud Security Alliance regards encryption as the most important control to secure data in the cloud. Based on your knowledge of encryption, why is encryption so important for cloud cybersecurity?

11. What cyberthreats do organizations face if they participate in the metaverse?

12. What is non-repudiation, and how is it achieved?

13. How can responsible data usage help with building a super-society?
14. How can you apply the concepts of responsible computing to help with risks of identity theft when using the Internet?
15. Explain why responsible impact is important to cloud computing?

Practice Exercises

1. Your supervisor has asked you to lead a team to adopt a Technical Reference Model (TRM) framework for your organization's computing system. What must you and your team consider to determine the best TRM for your organization?
2. As you and your team adopt a TRM framework, you are considering OMA and SOA. Explain the difference between the two architectures.
3. The Open Web Application Security Project (OWASP) identified the top ten risks for cybersecurity. Select any three of these risks and explain how you will address these concerns in your organization's information security policy.
4. Explain why the walled garden approach goes against the principles of the OWP.
5. How can backhaul help with poor network bandwidth issues?
6. As your organization's cybersecurity chief, you must assess your organization's cyber risks. To do this, what questions will you ask?
7. As a cybersecurity expert, you have been asked to handle penetration testing for your organization's software cybersecurity solutions. Explain the steps you will follow in this process.
8. Your organization's cloud network has become vulnerable to phishing attacks. What does this mean, and how will you address it?
9. After your organization began operating in the metaverse, they were warned about cyber-attacks from botnets. What are botnets, and how do you deal with them?
10. While AI poses additional cyberthreats, it can also be a tool to promote cybersecurity. How?
11. Create a list of questions you could ask during an interview with a company to ensure that it follows the principles of responsible computing.
12. What are some of the things that you should consider when applying responsible computing to smart ecosystems solutions?
13. You share files in the cloud with colleagues in multiple countries. As you do this, which of the pillars of responsible computing are you most concerned about and why?

Problem Set A

1. Your company is receiving bad press because they allowed another business access to their customer mailing list. Which domains in the Responsible Computing Framework should you focus on to resolve this issue?
2. The security and control of your company starts with an information security policy. Explain what this policy should include.
3. Your company uses PaaS for data analytics and is experiencing a bandwidth issue. How will you resolve this?
4. The president of your organization has informed you that most of the threats to the organization's network are from insiders. As the chief of your organization's cybersecurity team, how will you deal with

this?

5. Your organization deals with sensitive information, and you must ensure that this information is accessible only by authorized personnel. Explain how you could use cryptography to achieve this.
6. You learned that your web designer is using his pet's name as his password to log into the Internet and conduct maintenance on your organization's website. What policies will you require the web designer to follow to develop a stronger password that is less likely to be misused by hackers and other malicious actors?
7. Identify specific scenarios or issues encountered in web/mobile solutions and explain how they can be addressed by responsible computing.
8. Identify specific scenarios or issues encountered in cloud solutions, and explain how they can be addressed by responsible computing.
9. Identify specific scenarios or issues encountered in smart ecosystems solutions and explain how they can be addressed by responsible computing.

Problem Set B

1. Your company has asked you to lead a team that will develop a network system that incorporates the company services within applications, has the ability for computers in one division to interact with computers in other divisions, can adapt as new standards and practices are implemented, and will be able to survive if the system experiences an attack or disruption of service. Which qualities is your company most concerned about?
2. Your company does business with several customers who access the company's system using IoT devices. Explain to your supervisor why this is a security concern?
3. Your company wants to expand into 3-D modeling, but you are concerned about data management. Explain to your supervisor why data management is an issue with 3-D modeling.
4. Your organization has begun using the Cloud to share files, creating cybersecurity issues. As the cybersecurity chief, you have decided to assign a central policy administrator who will make security decisions about each file before it is shared. How will you achieve this?
5. Your organization is executing contracts through the metaverse, and you need to verify information without providing access to the data supporting the contract. How can you achieve this?
6. You've learned that one of your employees has been accessing your company's website but is hiding their IP address by using a public Wi-Fi network. Why is this a concern, and how will you address this?
7. You inherited your grandfather's business and intend to launch a website to promote the business and handle Internet sales. Prepare a cybersecurity policy that outlines how you will apply the pillars of responsible code and responsible data usage to your new website.
8. As your grandfather's business evolves, you begin conducting business in the Cloud. What changes will you make to your cybersecurity policy to apply the pillars of responsible code and responsible data usage to your Cloud operations and why are you making these changes?
9. As your grandfather's business grows, you attract customers in Japan. Explain how you will use responsible computing to respect Japan's efforts to develop a super-society.

Thought Provokers

1. You have been tasked with developing the architecture for a small company that provides landscaping services to about 1,000 customers. What are the company's activities that must be considered, and what

inputs are required to develop this company's architecture?

2. Explain why AR has become an issue for addiction and how this can be overcome.
3. Your organization is hesitant to use AI because management is concerned about the cybersecurity risks that AI will create. Do you agree or disagree with management?
4. As a website moderator encouraging visitors to offer opinions about various topics, you have noticed that some users bully and harass others. How can you handle this while respecting the users' anonymity and privacy?
5. Ethics is an important part of responsible computing. Assume that you have been hired as an ethicist to advise your organization's cybersecurity professionals. What issues will you cover with them, and how will you apply responsible computing pillars to the organization's efforts to be ethical?

Labs

1. Using the Responsible Computing Framework, how will you include guidance for ethical behavior in your company's architecture? Why does this matter? As you develop the architecture, what ethical behaviors are you most concerned about and why?
2. The term "white hat hacker" is used to describe a career field or title in cybersecurity where cyber experts purposefully attack systems in legal and ethical ways to provide feedback to improve cybersecurity. Research the tasks and things that cybersecurity experts do in this role, and explain why it is a crucial part of systems development? Should a system that has already been deployed still have this level of cybersecurity analysis?
3. Research a cyber-attack against critical infrastructure and why it is so important for both private organizations who own and operate such networks and why it is also important for government organizations to oversee and ensure proper governance of such cybersecurity measures. Focus on the impact some of the incidents have had on the local communities that depend on these types of infrastructure.

A Appendix A: Network Design Application of Algorithms

Introduction to Data Communications and Networks

In computer science, a network is a system of computers and other technology that are interconnected to store and share information. Networks enable data communications, allowing users to interact on computers and other technology, such as cell phones. This gives users the ability to chat and exchange information on applications such as social media, emails, and video chats. For additional information on any of these topics, we suggest visiting *Computer Networks: A Systems Approach* by Larry Peterson and Bruce Davie, using the links included here.

Network Components

Networks are comprised of various components. Applications are computer programs and software that enable us to use a network (https://book.systemsapproach.org/foundation/applications.html). Examples of applications include the World Wide Web, instant messaging, streaming services for movies and music, file-sharing, social media, and emails.

To build a robust network, computer scientists must consider the network's requirements (https://book.systemsapproach.org/foundation/requirements.html). The requirements include the stakeholders who develop the network, as well as those who manage and operate the network. Scalable connectivity is a critical requirement to understand the needs that the network must currently meet, as well as the needs that must be met by the network as the organization grows and changes. Other requirements include the cost-effectiveness of the network, the support that the network provides for common services, such as an organization's email application, and the network's ability to be managed.

Like blueprints to construct buildings, computer networks need architecture to ensure the component parts are arranged and structured appropriately to ensure the network functions as needed (https://book.systemsapproach.org/foundation/architecture.html).

Computer networks cannot function without software, which refers to the programs that instruct computers on the tasks to perform. The different types of software include system, utility, and application. Examples of software include word processing and spreadsheet programs like Microsoft Word and Excel (https://book.systemsapproach.org/foundation/software.html).

Communication Links

Communication links are vital to connect the various nodes and users in a network. This requires tools like copper wire, optical fiber, and the air for wireless links (https://book.systemsapproach.org/direct/perspective.html). Then processes like encoding (https://book.systemsapproach.org/direct/encoding.html), framing (https://book.systemsapproach.org/direct/framing.html), and error detection (https://book.systemsapproach.org/direct/error.html) are necessary to develop a reliable transmission (https://book.systemsapproach.org/direct/reliable.html). Also, it's important to recognize that different types of communication links are needed for different types of networks, such as multi-access (https://book.systemsapproach.org/direct/ethernet.html), wireless (https://book.systemsapproach.org/direct/wireless.html), and access (https://book.systemsapproach.org/direct/access.html).

The Internet

To understand the concept of a computer network, consider the Internet, which is a global network that connects millions of people, enabling them to communicate and share information (https://book.systemsapproach.org/scaling/global.html).

The Internet is a good example of a network of networks, since it actually is comprised of many smaller networks that are interconnected (https://book.systemsapproach.org/internetworking/basic-ip.html).

To ensure that the Internet and other computer networks can function, they must have protocols, which are standardized rules to guide how data is formatted and processed. Protocols enable networks to transmit information (https://book.systemsapproach.org/e2e/problem.html). Common protocols include simple demultiplexor (https://book.systemsapproach.org/e2e/udp.html), reliable byte stream (https://book.systemsapproach.org/e2e/tcp.html), remote procedure call (https://book.systemsapproach.org/e2e/rpc.html), and transport for real-time (https://book.systemsapproach.org/e2e/rtp.html). Web services also have protocols, including standards issued by the World Wide Web Consortium (https://book.systemsapproach.org/applications/traditional.html#web-services).

Network Edge and Core

To use a network, such as the Internet, users rely on end systems, such as personal computers, tablets, and cell phones. End systems also include components like servers for email and even game consoles that are connected to the Internet. End system devices operate at the edge of the Internet, which is the point where such devices are connected to the network (https://book.systemsapproach.org/direct/trend.html).

Hosts as Clients and Servers

To enable communication, networks rely on the client, which sends a request to gain access to a network's file, and the server, which makes a file available by providing access (https://book.systemsapproach.org/foundation/requirements.html#support-for-common-services).

Access networks are an example of the client-server relationship in networks (https://book.systemsapproach.org/direct/access.html). Internet service providers and cable service are examples of access networks, which enable users to connect to a network via personal devices like computers, cell phones, and TVs.

While networks often rely on wired links to function, many networks are wireless. Popular wireless technologies include Bluetooth, Wi-Fi, and 4G cellular (https://book.systemsapproach.org/direct/wireless.html).

Network Core

To function, networks must have scalable connectivity, which means the computers in the network are connected and the network has the ability to grow to a larger scale (https://book.systemsapproach.org/foundation/requirements.html#scalable-connectivity). This is achieved through switched networks, which have hardware components that connect the devices using a network that enables them to share data packets.

The different types of switched networks include packet switched and circuit switched. Packet switched networks are commonly used for computer networks and process data into packets that are sent through the network to nodes. With packet switching, data may be broken into smaller packets that travel independently on different routes in the network. With circuit switched, networks which are commonly used by the telephone system, messages travel over a dedicated route to reach the destination.

Network Core Functions

In addition to switching, a vital function of networks is routing, which refers to the process that networks use to determine the optimal path for data packets to travel as they move through the network (https://book.systemsapproach.org/internetworking/routing.html and https://book.systemsapproach.org/internetworking/switching.html#source-routing). Routers ensure that data actually reaches its destination.

Network Performance

Performance is an important part of networks that determines how efficiently the network functions. Performance has two important aspects, including bandwidth, which refers to the number of bits that a network can transmit in a given time, and latency, which is the amount of time it takes to transfer data (https://book.systemsapproach.org/foundation/performance.html).

Network Security

As networks share information, it is vital that they are secure to protect the privacy and sensitive information of users. The concepts important in network security include security trust and threats, ciphers, authenticators, public and secret keys, authentication protocols, and firewalls (https://book.systemsapproach.org/security.html).

Network Layers and Service Models

An important part of network functionality is abstraction, which is a fundamental tool that enables network designers to manage a system's complexity by simplifying computer code. With abstraction, code is organized into functions and the underlying complexity is hidden. This makes it easier for programmers to understand the code, and they can work with it more efficiently, writing code more quickly with fewer errors.

Abstractions are the foundation of layering, which is the process of breaking a network into layers that make it easier to transmit information across the network (https://book.systemsapproach.org/foundation/architecture.html#layering-and-protocols). Each layer serves a different purpose, such as providing host-to-host connectivity and supporting application programs.

One of the first layering protocols was the OSI Model, which uses seven layers in its architecture (https://book.systemsapproach.org/foundation/architecture.html#osi-model).

Encapsulation is an important part of this process. With encapsulation, data is protected and access to computer code is controlled, enabling users to interact with the network without the risk of comprising data integrity (https://book.systemsapproach.org/foundation/architecture.html#encapsulation).

Internet Network Protocols

The Internet network's architecture includes the main protocols of Transmission Control Protocol (TCP) and Internet Protocol (IP), as well as User Datagram Protocol (UDP) (https://book.systemsapproach.org/foundation/architecture.html#internet-architecture). Other protocols important for the Internet to function include protocols for application layer (https://book.systemsapproach.org/applications/traditional.html), web application, file transfer (https://book.systemsapproach.org/foundation/requirements.html#identify-common-communication-patterns), email (https://book.systemsapproach.org/applications/traditional.html#electronic-mail-smtp-mime-imap), video streaming, and transport layer (https://book.systemsapproach.org/security/systems.html#secure-shell-ssh). In addition, framing protocols are important, including byte-oriented protocols (https://book.systemsapproach.org/direct/framing.html#byte-oriented-protocols-ppp), bit-oriented protocols (https://book.systemsapproach.org/direct/framing.html#bit-oriented-protocols-hdlc), and clock-based framing (https://book.systemsapproach.org/direct/framing.html#clock-based-framing-sonet). For host-to-host delivery services, the simple demultiplexor (UDP) is useful (https://book.systemsapproach.org/e2e/udp.html).

Routers Forwarding Functionality

Routers also have protocols, and it's important to understand router architectures, including switching and input port functions. Routing uses graph-theory and algorithms to ensure that packets take the appropriate route and make it to their intended destination (https://book.systemsapproach.org/internetworking/routing.html). Protocols such as Routing Information Protocol (RIP) (https://book.systemsapproach.org/internetworking/routing.html#routing-information-protocol-rip) and link-state routing (https://book.systemsapproach.org/internetworking/routing.html#link-state-ospf) help ensure routing functionality.

Routing is unable to function without switching (https://book.systemsapproach.org/internetworking/switching.html), and this includes datagrams (https://book.systemsapproach.org/internetworking/switching.html#datagrams), virtual circuit switching (https://book.systemsapproach.org/internetworking/switching.html#virtual-circuit-switching), and source routing (https://book.systemsapproach.org/

internetworking/switching.html#source-routing).

Internet Control Message Protocol

Errors are an inevitable part of any network, and this can impact network functionality. To deal with this issue, the Internet relies on a companion protocol for IP called Internet Control Message Protocol (ICMP) (https://book.systemsapproach.org/internetworking/basic-ip.html#error-reporting-icmp). ICMP defines a collection of error messages and returns them to the source host if a router or host cannot process the IP datagram and deliver the messages. Network management also has protocols, including Simple Network Management Protocol (SNMP) and OpenConfig (https://book.systemsapproach.org/applications/infrastructure.html#network-management-snmp-openconfig). For address translation, the Address Resolution Protocol (ARP) is important (https://book.systemsapproach.org/internetworking/basic-ip.html#address-translation-arp).

Encoding

Network functionality depends on encoding, which is the process of converting a character sequence into the appropriate format to store or transmit the data (https://book.systemsapproach.org/direct/encoding.html). Encoding can be approached using compression, which seeks to encode bits of data in the smallest set possible. Compression techniques include run length encoding, differential pulse code modulation, and dictionary-based methods (https://book.systemsapproach.org/data/multimedia.html#lossless-compression-techniques). Encoding is also an important process to prepare files like JPEG (https://book.systemsapproach.org/data/multimedia.html#encoding-phase), MPEG (https://book.systemsapproach.org/data/multimedia.html#transmitting-mpeg-over-a-network), and videos (https://book.systemsapproach.org/data/multimedia.html#video-encoding-standards).

Wireless and Mobile Networks

Wireless networks function similarly to wired networks, except all wireless links use the same medium. To ensure this medium is shared efficiently, it is divided using the dimension of frequency and space. Government agencies, such as the U.S. Federal Communications Commission make many of the determinations regarding which networks can use the medium, and this includes allocating certain frequency ranges for specific uses, such as television and cell phones (https://book.systemsapproach.org/direct/wireless.html#basic-issues). The physical properties of wireless networks include bandwidths (https://book.systemsapproach.org/direct/wireless.html#physical-properties). Collision avoidance is a focus of wireless protocols (https://book.systemsapproach.org/direct/wireless.html#collision-avoidance), and wireless distribution networks use the link layer for operations (https://book.systemsapproach.org/direct/wireless.html#distribution-system). As with all networks, security is a chief concern for wireless systems (https://book.systemsapproach.org/direct/wireless.html#security-of-wireless-links). Bluetooth is a popular wireless network (https://book.systemsapproach.org/direct/wireless.html#bluetooth-802-15-1).

Routing can be challenging for wireless, mobile devices. For example, IP addresses and mobile hosts must be handled differently in wireless networks, making it necessary to use different protocols when developing wireless networks (https://book.systemsapproach.org/scaling/mobile-ip.html#routing-among-mobile-devices).

Distributed and Decentralized Systems

Some networks use virtual circuit switching as an alternative technique for packet switching (https://book.systemsapproach.org/internetworking/switching.html#virtual-circuit-switching). Asynchronous Transfer Mode (ATM) is a well-known example of networking technology that relies on virtual circuit switching (https://book.systemsapproach.org/internetworking/switching.html#asynchronous-transfer-mode-atm). Another option is multiprotocol label switching, which leverages the robustness and flexibility found in datagrams with virtual circuit switching properties (https://book.systemsapproach.org/scaling/mpls.html).

Peer-to-peer networks are also useful in networks that allow a community of users to share their

resources—including bandwidth, storage, and content—to give them greater network access than they would have individually. Examples of peer-to-peer networks include Gnutella and BitTorrent (https://book.systemsapproach.org/applications/overlays.html#peer-to-peer-networks).

History of the Internet and Inner-Workings

Historically, the Internet has relied on specialized devices built with application-specific integrated circuits (ASICs), and this has made it time consuming to develop networks. Hardware switches have helped improve this situation (https://book.systemsapproach.org/internetworking/impl.html#hardware-switch). With this change, software-defined networks have become more common (https://book.systemsapproach.org/internetworking/impl.html#software-defined-networks).

Metrics, or link costs, are an important consideration in routing and selecting the correct algorithm to handle the process. The ARPANET provided the testing ground to develop approaches for link-cost calculations (https://book.systemsapproach.org/internetworking/routing.html#metrics).

The Ethernet was developed in the mid-1970s and eventually dominated technology for local area networking. The basis for Ethernet was in Aloha, a packet radio network developed at the University of Hawaii as a support network to enable computer communications throughout the Hawaiian Islands (https://book.systemsapproach.org/direct/ethernet.html).

Index

Symbols

3-D printing 782
4-D printing 783

A

abacus 10
abstract data type (ADT) 91
abstract method 334
abstract model 144
abstract representation 546
abstraction 42, 52, 54, 63, 70, 90, 198, 199, 204, 299, 332, 335, 337
acceptance testing 434, 474
access control 834, 841, 878, 887
access control list (ACL) 280
access enforcement 279, 281
access modifier 327, 333
accessibility 788
accessibility problem 363
accessibility testing 475
adaptability 816, 818, 820, 821, 827, 829, 830
address space 244, 247, 248, 250, 251, 253, 258, 259, 261
address translation 271
addressing mode 215
adjacent 96
Advanced Package Tool (apt) 725
advanced persistent threat (APT) 838
adversarial attack 28
affordability 816
Agile EA Management (AEAM) 536
Agile Manifesto 442
Agile software development 508
Agile Software Development Ecosystem (ASDE) 443
algorithm 10, 11, 14, 26, 31, 52, 78, 91, 99, 102, 106, 107, 124, 439
algorithm analysis 102
algorithm design 99
algorithm design pattern 101
algorithmic paradigm 110
algorithmic problem-solving 98, 101
allocated memory 158
allocation 242, 246, 249, 256, 267, 269, 272, 274
Amazon Elastic Compute Cloud (EC2) 725
Amazon Simple Storage Service 758
Amazon Web Service 702
Amazon Web Service (AWS) 724
American Standard Code for Information Interchange (ASCII) 208
Analytical Engine 11, 124
analytics process model 405
Android Studio 568
anonymity 841, 842, 845, 846, 847, 878
Apache Hadoop 370
API 252, 326, 666, 671, 682, 687, 706, 709, 759
API gateway 668, 673, 697
application 195
application architecture 63, 73
application programming interface 245
application programming interface (API) 247
application security 836
application software 434
applications 228
ArchDev (SecOps) 536
archetype 546
architectural pattern 52, 71, 547
architectural style 502, 539, 547
architecture continuum 509
Architecture Continuum 527
Architecture Development Method (ADM) 524
architecture management 455
architecture model 53, 72
architecture scope 70
archival backup 389
archive file 167
Arduino 180
argument 327, 331, 343
arithmetic 216
arithmetic logic unit (ALU) 196
ARPANET 12, 15
array 313, 332, 341
array initializer 313
array list 92, 117
Array of Things (AoT) 409
artificial human 791
artificial intelligence 21, 25, 26
artificial intelligence (AI) 15, 410
assembler 148, 157, 162, 165, 201
assembly 305, 343
assembly language 148, 157, 201, 211, 215
assignment statement 315, 332
Association for Computing Machinery (ACM) 487
asymptotic analysis 106
asymptotic notation 108
asynchronous call 386
Asynchronous JavaScript and XML (AJAX) 564
atomicity, consistency, isolation, and durability (ACID) 372
attribute 363, 373, 392, 399
augmented reality (AR) 780
authentication 239, 241, 279, 280, 281, 834, 839, 840, 844, 845, 883, 885, 886, 887, 892
authorization 239, 246, 279, 280, 281
autocomplete 101
automated testing 474
automation 42
autonomous system 830
autonomous systems 891, 893, 898
availability 437, 451, 456, 482
AVL tree 94, 117
AVL tree property 94
AWS 673, 676, 704, 705, 724
AWS CLI 727
AWS Portal 725
Azure 702, 705, 733, 736, 742
Azure Function App 733, 735

Azure Portal 706

B

B-tree 385
B+ tree 385
back end 339, 341
Backend as a Service (BaaS) 697
badge 280
balanced binary search tree 94
bare metal server 693, 695, 699
base container image 767
BASIC 152
basic block 345
best practice 301, 315, 319, 327
best-case situation 106
biased information 363
big data 21
big data analytics 78, 787, 827, 887
big design up front (BDUF) 449
Big O notation 106, 108
BigTable 395
binary 380
binary code 147
binary heap 95
binary logarithm 109
binary operator 316
binary search 100, 111, 117
binary search algorithm 10, 102, 117
binary search tree 93
binary tree 385
binary tree property 93
biomimetic robotics 791
bit 195
black box testing 470
block 224, 225
block storage 761
blockchain 76
Blockchain 2.0 776
blockchain DBMS 396
blockchain network 776
blocked state 247, 263
blueprint 52, 528, 532, 535, 546
Boolean 312
bottom-tested loop 323
breadth-first search 121
bring your own cloud (BYOC) solution 760
brute-force algorithm 112
bug 432, 434, 450, 471
bug tracking system 478
bus 197, 213, 219
business intelligence (BI) 402, 438
business logic layer 54
business model 57, 80
business process 510
business process hierarchy 59
business service choreography 506
business service orchestration 506
busy waiting 336
byte 195
bytecode 310, 343

C

C 149, 154, 156, 159, 161, 164, 167, 169, 175, 177, 179, 440
C# 440
C++ 149, 156, 156, 158, 161, 167, 169, 177, 440
cache hit 222, 224
cache memory 222, 222
cache miss 222, 225
cache-only memory architecture (COMA) 249
call stack 328
canonical algorithm 90
canonical searching algorithm 100
CAP theorem 395
capability list 280
capacitor 220, 222
cascading style sheets (CSS) 562
case analysis 106
case-sensitive 314
Cassandra 395
Cassandra DB 768
central processing unit (CPU) 144
centralized DBMS architecture 371
change data capture (CDC) 399
ChatGPT 429
child node 93
child thread 178
Church-Turing Thesis 147
Clang 161
class 329, 332, 344
client-side script 339
closed-source 484
cloud 543, 548
cloud computing 671, 675, 684, 686, 688, 692, 705, 724
cloud DBMS architecture 371
cloud infrastructure 253, 256
cloud mashup 704
cloud-native application 659, 660, 668, 671, 672, 673, 677, 678, 701, 702, 704, 705, 709, 715, 733
CLR 341, 344
cluster 382, 395, 405, 683, 696, 706, 719, 726, 728
code block 320, 334, 345
code coverage 471
code generation 342
code relocation 165
code review 451
coercion 312
cognitive computing 410
cognitive robotics 791
collection 100
collision 120
column-oriented database 370, 394
combinatorial explosion 112, 124
combinatorial problem 112
combined assignment 318
command line interface (CLI) 762
comment 319, 340
Common Gateway Interface (CGI) 307, 339
Common Language Runtime (CLR) 310
community cloud 685, 691
comparison operation 111, 118
comparison sorting 111, 117
compilation 309, 340
compiler 145, 151, 156, 157, 161, 175, 179, 314, 319, 326, 335, 340, 343
complex data 90, 98, 111
complex data type 311
complex instruction set

computer (CISC) 212
complexity 104
component 502, 508, 533, 546, 661, 666, 686
composability 815
compression 98
computational model 144, 147, 156
computational science 21, 24, 25, 26
computational thinking 40, 52, 71
compute service 763
computer program 14
computer science 11, 13, 15, 17, 21, 23, 24, 25, 27, 29
computer science (CS) 10
computer scientist 365, 402
computer system 194, 197, 204, 207, 218, 223
computer systems 195, 221, 228
computing 10, 11, 12, 16, 19, 21, 21, 24, 26, 28, 31, 31
concurrency 260, 261, 267, 343
concurrency control 387
concurrent processing 257, 262
concurrent programming 174, 309, 342
condition variable 266
condition-controlled 323
conditional expression 321
connection manager 369
consistency problem 363
constant 107
constant-time operation 117
construction phase 450
constructor 333
contact tracing 97
container 258, 284, 659, 661, 668, 669, 672, 674, 675, 680, 682, 683, 696, 700, 702, 821, 888
Container as a Service (CaaS) 696
container image 674, 696
container management services 766
container registry (CR) 767
containerization 659, 667, 674, 682, 700, 725
containerized 451
containers 821, 836
Containers as a Service 692, 696
content delivery network (CDN) 764
content moderation 104
context switch 263
continuous delivery 659, 677, 677
continuous integration 659, 677, 701
continuous integration and continuous deployment (CI/CD) 677
core 150, 173, 180
correctness 103
cost model 106
count sorting 119
count-controlled 323
CPU 241, 242, 243, 245, 248, 249, 251, 258, 259, 261, 263, 263, 267, 268, 271, 274, 275, 279, 282
CPU state 258, 259
critical infrastructure 833, 894, 894
critical infrastructure. 903
critical section 265, 266, 281
crosscutting activity 452
cryptography 98, 836, 839, 841, 843, 851, 893
cyber economics 893, 893, 895, 897
cybersecurity 812, 831, 833, 835, 836, 836, 838, 840, 854, 885, 887, 888, 890, 891, 893, 895, 898, 903
cybersecurity assurance 811, 833, 836, 886, 893, 898
cyborg 791

D

data 360
data accuracy 363
data analysis 20
data architecture 508, 526
data architecture model 65
Data as a Service (DaaS) 396
data completeness 363
data compliance 364, 405
data consistency 363, 372, 393
data consolidation 403
data control language (DCL) 368
data description language (DDL) compiler 369
data dictionary 362, 367
data federation 403
data governance 364
data integration 399, 403
data lake 400
data management 360, 378, 404
data management layer 54
data manipulation language (DML) 368
data mart 399
data model 362, 366, 377, 390, 397
data modeling 66
data movement 216
data owner 365, 404
data packet 96
data propagation 403
data quality (DQ) 363
data quality dimension 403
data quality problems 363
data query language (DQL) 368
data redundancy 368, 378, 385
data replication 387
data representation 91
data science 21, 21, 24, 26, 30
data scientist 366, 400, 404
data security 364, 404
data security platform 839
data steward 365, 404
data structure 90, 92
data structure problem 98
data swamp 401
data type 90, 179
data types 311, 314
data virtualization 403
data warehouse 372, 397, 403, 412
database administrator (DBA) 365
database administrators 432
database application 367, 391, 395
database architecture 385

database description language (DDL) 368
database designer 365, 367, 381
database language 368, 377
database management system (DBMS) 362, 366, 419
database normalization 378, 390
database recovery 389
database security 389
database transaction 387
database user 367
DBMS interface 369
DBMS utilities 369
deadlock 266, 309
debugger 476
debugging 43, 52, 75
decentralized Apps (DApps) 570
declarative programming 151
decomposition 42, 58, 65
decrement operator (--) 317
deep learning network 410
deep machine learning 770
demand paging 267, 272
Denial of Service Attacks (DoS) 389
denial-of-service attack 834
Dennard scaling 227
denormalizing 406
deployment 434, 451, 482
depth-first search 120
descriptive analytics 406
design component 52
design pattern 448, 482
destructor 333
detail-level design (DLD) 447
determinative 324
device driver 149, 158, 245, 253
device manager 254
device register 251
DevOps 659, 673, 677, 682, 687, 701
DevOps model 465
DFS 278
dictionary 91, 100
Difference Engine 11
Dijkstra's algorithm 123
directory 267, 274, 276, 281
disaster recovery 679
disk 195
disk storage 382
distributed computing 174
distributed denial-of-service (DDoS) attack 838
distributed file system (DFS) 277
distributed transaction 387
divide and conquer algorithm 110
Django 674, 674
Django project 580, 620
Docker 659, 680, 682, 705, 706, 715, 719, 725
Docker Compose 726, 727
Docker Engine 725
domain constraint 378
Dr. Edgar F. Codd 373
DRAM 220, 222
dual in-line memory module (DIMM) 221
dual mode 252
duplication 363
dynamic library 211
dynamic linker 343
dynamic linking 267, 269
dynamic method binding 335
dynamic programming 12, 115
dynamic quality 436
dynamic random access memory (DRAM) 220
DynamoDB 395

E

EA domain 55
EAF 55
EC2 704, 725
edge 96
elaboration phase 446
Electronic Communications Privacy Act (ECPA) of 1986 404
Electronic Numerical Integrator and Computer 13
element 90, 92, 95, 111, 298, 332, 338, 341
elementary business process (EBP) 58
ELSE 47
embedded script 339
embedded software 434
enabler 516
encapsulation 300, 332
encryption 275, 277
ENIAC 13, 14
enterprise architecture (EA) 503, 510
enterprise architecture framework (EAF) 519
enterprise architecture management (EAM) 510
enterprise or solution portfolio architect 432
enterprise search 410
enterprise service bus 666
enterprise service bus (ESB) 666
entity integrity constraint 378
entity model 59
equi-join 376
Ethereum blockchain 571, 637
Ethereum platform 76
evaluation 383
event-driven 308
event-driven architecture 697
evolvability 815, 836
exception 245, 263, 301, 343
executable 195
executable and linkable format (ELF) 166
execution 383
exokernel 256
experimental analysis 105
exploratory analysis 406
exponentiation operator (**) 317
Express.js 706, 709
expression 304, 317, 331, 345
extended reality 760
extended reality (XR) 780
extensibility 437, 816, 818, 821, 831
external references 167
extraction, transformation, and loading (ETL) 399

F

FaaS 697
Faceted search 53
facial recognition 99
fact constellation 398

fat client variant 386
fault containment 243, 253
fault recovery 243
fault tolerance 243
federated DBMS 371
file 274
file storage 761
file system 250, 254, 260, 274, 276, 277, 279
file versioning 275
Firebase 568
firmware 180
first come, first served 243
first come, first served (FCFS) 264
first-class function 332
flag 214, 216
flash memory 224, 241, 283
flat file database 390
flexibility 437
floating point 209, 217, 312
floating point number 208
flow of control 302, 308, 320
flowchart 46
for loop 325
formal parameter 327
fragmentation 246, 269
frame buffer 250
framework 502, 508, 510, 511, 517, 524, 543, 545
free and open-source software (FOSS) 468
freed memory 158
front end 339, 341
full node 572
full virtualization 268
full-text search 410
function 48, 54, 63, 77, 151, 158, 164, 167, 177, 300, 305, 308, 331, 336, 340, 343
Function as a Service 673, 692, 696
Function as a Service (FaaS) 697
function call 326
function signature 326
functional dependency (FD) 378
functional programming 146
functional requirement 436, 474
functionality 90

G

garbage collection 333
garbage in, garbage out (GIGO) 363
gas price 574
GCC 161
GenAI 45
General Data Protection Regulation (GDPR) 404
general-purpose register (GPR) 258
Generative AI (GenAI) 429
Geographic Information Systems (GIS) 367
Git 172
GitHub 172, 479, 701
GitLab 682, 701
global optimization 345
global positioning system (GPS) 366
Google Maps 367
GOTO 152
governance 520, 546
graph 91, 96
graph problem 98, 120
graph-based database 394
graphical user interface 249
graphical user interface (GUI) 247
graphics processing unit (GPU) 173
GraphQL 567
gray box testing 470
greedy algorithm 113, 122
guest modification 269
guest operating system 269, 284

H

Hadoop 366, 408
HAL 247
Hard disk 223
hard disk drive 761
hard disk drive (HDD) 223
hardware 13, 15, 18, 21, 23, 29, 194, 197, 361, 401, 405, 408
hardware abstraction layer 247
hardware abstraction layer (HAL) 254
hardware model 144, 150
hash table 119
hashing 98, 119
heap 213
heap allocation 271
heap data 258, 271
heap property 95
heapsort algorithm 118
heterogeneous 226, 228
heuristic 54
heuristics optimization 383
hierarchical DBMS 370
hierarchical model 390
high availability 679, 698
high-level design (HLD) 447
high-level language (HLL) 200
high-level programming language 144, 149
high-order function 332
high-performance computing (HPC) 685
hit rate 224
Homebrew 725
homogeneous 226
horizontal fragmentation (sharding) 387
HPC 699
human-computer interaction 25
human-computer interaction (HCI) 15
hybrid cloud 661, 661, 685, 763, 767
hybrid cloud application 78
hybrid implementation 309, 340, 342
hypertext markup language (HTML) 562
hypervisor 243, 268, 283, 694, 699

I

i-number 276
IaaS 686, 686, 688, 692, 696, 701, 702
IBM Research 373
identifier 314, 342
identity and access management 839
identity and access management

(IAM) 839
identity theft 894, 903
idiom 503, 549
IEEE 431
IEEE 754 209, 210
IEEE-CS established the Committee on Professional Ethics (COPE) 487
IF 47
ilities 815, 817, 822, 822
image recognition 26, 28
immediate backup 389
imperative language 304, 332
imperative programming 151
in-memory DBMS 371
inception phase 444
inclusion 788
inconsistency 281
increment operator (++) 317
incremental backup 275
incremental model 459
index 91, 117, 368, 393, 410
indexed array 313
indexed organization 382
industrial espionage 894, 903
informatics 412
information architect 364, 367, 413
information hiding 330
information retrieval 409
information science 23, 24, 26
information security 812, 815, 816, 823, 832, 836, 898
Infrastructure as a Service 684, 686
infrastructure as a service (IaaS) 760
Infrastructure as a Service (IaaS) 686
infrastructure security 835
Infrastructure Security 840
inheritance 300, 333, 341
initialization 315
inner join 376
inode 275, 276, 281
input 100, 102
INPUT 47
input/output (I/O) 197
input/output (I/O) devices 197
input/output devices 193
insider threat 838
instantiation 332
instruction set architecture (ISA) 144
integer 312
integer data type 90
integers 208
integrated development environment 670
integrated development environment (IDE) 161
intelligent autonomous networked supersystems (IANS) 784
inter-process communication 257
inter-process communication (IPC) 260
Interaction design (IxD) patterns 53
interface 297, 302, 334
intermediate code 343
intermediate form (IF) 341
intermediate language 310
International Society for Technology in Education (ISTE) 41
Internet 96
Internet of Things (IoT) 78
interoperability 813, 815, 816, 820, 822, 823, 826, 827, 829, 873
interpreter 145
interrupt 250
interrupts 245
interval scheduling problem 112
intractable 125
invalid pointer 160
iOS Files 434
IPC 257
isolation 240, 246, 255, 260, 267, 271, 275, 283
IT automation 522
IT context management 522
IT governance 522
iteration 308, 323

J

Java 445
Java virtual machine (JVM) 310, 343
JavaScript (JS) 562
JavaScript Object Notation (JSON) 567
Jetpack Compose 568
John R. Mashey 402
just-in-time (JIT) compilation 343
just-in-time (JIT) translation 310

K

Kanban Agile 449
kernel 150, 158, 179, 180, 243, 245, 250, 252, 255, 261, 263, 281, 283
key constraint 373, 378
key-value store 370, 394
keyword 300, 314, 333, 336
Kruskal's algorithm 115, 117, 122
Kuard 724
Kuard (Kubernetes Up and Running Demo) 732
Kubectl 720
kubelet 682, 682
Kubernetes 659, 675, 681, 682, 683, 683, 696, 702, 705, 706, 715, 724, 726, 732

L

label 216
Lambda calculus 145
large language models 406
late binding 335, 341
layered OS architecture 253
leaf node 93
legacy software 435, 452
level of abstraction 147, 149
lexical analysis 341
library 164, 169, 171, 177
limited computing resources 363
line coverage 471
linear 107
linear data structure 92
linked list 92
linker 162, 164, 166, 167, 170, 185, 316
linking 162, 165, 170
Linux 150, 157, 161, 166, 172,

181
list 91
literal 315
load time linking 170
loader 211
local optimization 345
locality 218, 222, 225, 225
lock 266, 276
logarithm 108
logarithms 11
logging and monitoring management 765
logic gate 203
logic operations 216
logical data independence 368
logical design 377
longest path 125
loose coupling 563
low-level programming language 144
lvalue 316

M

machine code 147, 158, 162, 172, 203, 213
machine learning 21, 25, 26, 40, 78
machine learning algorithm 99
MacOS Pages 434
macro life cycle 413
main memory 198
maintainability 332, 335, 337, 437, 471, 473, 482, 817
maintenance 431, 452, 487
malware 837, 864, 865, 873
man-in-the-middle 838, 872
managed code 310, 341
manual testing 474
map 91, 96
MapReduce 396, 407, 412
mashup 759
master data management (MDM) 404
matching 98
mechanism 243, 247, 260, 266, 269, 272, 279, 281, 283
member 333
memory 13, 14, 26
memory allocation 249, 268, 270
memory deallocation 249
memory hierarchy 219, 225
memory leak 158
memory management 158, 167
memory multiplexing 267
memory technology 222
merge sort 111
merge sort algorithm 118
merging process 406
message passing 177
Message Passing Interface (MPI) 175
meta-framework 522
metadata 343, 374, 388, 392, 399, 404
metadata modeling 362
metamodel 533
method 520, 546
Metro bundler 628
micro life cycle 413
microarchitecture 203
microkernel 253, 254
microservice 503, 543, 548, 663, 667, 668, 671, 672, 673, 676, 678, 682, 705, 709, 711, 715, 720, 721
microservices 77
Microsoft Azure 74, 761
Microsoft Word 434
middle end 341
middle-level programming language 149, 158
migrating legacy business solutions 77
minimum spanning tree 98, 122
minimum spanning tree algorithm 114
minimum spanning tree problem 117
miniworld 367
missing value 406
mixed fragmentation 387
mixed reality 786
mixed reality (MR) 780
ML toolkits 775
mobile robot 792
model 98
model of computation 99
Model-View-Controller (MVC) 71
modeling 115
modularity 154, 158, 164, 179
modularization 329
module 329, 338, 340
modulo operator (%) 317
MongoDB 366, 370
MongoDB Atlas 768
monolith 661, 665, 676
monolithic architecture 661, 665
monolithic design 253
monolithic structure 54
Moore's law 226, 561
multicloud solution 758
multicore 173
multifile relational database 390
multimedia DBMS 372
multiple inheritance 334
multitask 335
multitasking 242, 261, 268
multiuser DBMS 371
multivalued dependency (MVD) 378
mutable 332
mutual exclusion 265

N

n-tier DBMS architecture 371
name-value pair 314
named constant 316
NAND 224
NAND gate 224
nanotechnology 790, 831, 891, 892, 898
natural join 376
Neo4j 370
network 13, 18
Network Attached Storage (NAS) 388
network DBMS 370
network security 835
neural network 27, 28, 29
neuroinformatics 794
neuromorphic 194
neuromorphic computer 205
neuromorphic computing 795
Next.js 706, 709
node 93, 96, 604, 619, 624, 635, 637, 706, 709

nomadicity 816
non determinative 324
non-first normal form (NFNF) 390
non-fungible token (NFT) 576
non-privileged system program 253
non-uniform memory access (NUMA) 249
Non-Volatile Memory Express 761
nondeterministic algorithm 125
nondeterministic polynomial (NP) time complexity class 125
nonfunctional requirement 436, 444, 448, 474
nonrelational database 390
nonrepudiation 836
NoSQL database 768
NoSQL DBMS 370, 393
NP 127
NP-complete 125, 127
numerical weather prediction 21

O

object 151, 155, 162, 165, 169, 172
object code 213, 340, 343
Object Management Group 813
Object Management Group (OMG) 894
object persistence 385, 391
object storage 761
object-oriented DBMS 370
object-oriented programming 155
object-relational mapping (ORM) 709
objectivity problem 363
Octant 725, 732
one's complement 206
online analytical processing (OLAP) 372, 402
online mapping 102
online transaction processing (OLTP) 372
ontology 393
Open Web Application Security Project (OWASP) 834, 886
open-source 484
open-source DBMS 372
OpenMP 177
operand 215, 216
operating system 149, 157, 161, 166, 181, 239, 240, 242, 249, 252, 260, 268, 271, 272, 273, 283, 434, 481, 485
operating system (OS) 204, 240
operational data store (ODS) 400
operations specialists 432
operator 145, 158, 316
optimistic rollup 574
optimization 341
optimizer 362, 369, 383
orchestration platform 676, 681
order of growth 107
OS 253, 256, 262, 262, 281
outer join 376
outlier 406
output 100, 102, 102
OUTPUT 47
override 335

P

P 127
PaaS 689, 692, 696, 700, 701, 704
page fault 271, 273
page fetching 273
page manager 254
page replacement 273
paging 272, 273, 282
parallel computer 173
parallel computing 173
parallel processing 388, 412
parallel programming 173, 177, 179, 335
parallelism 335, 343
parent node 93
parent thread 178
parsing 341, 395
pass by reference 327
pass by value 327
password 840, 840, 883
passwords 820, 885
path coverage 471
pattern 439, 448, 481, 517, 525, 533, 543, 545, 547
pattern catalog 505, 539
pattern hierarchy 502
pattern language 508
Pattern recognition 42
patterns management 508
PCB 263
peer-to-peer (P2P) 561, 756
perfectly balanced 94
performance 437, 443, 470, 477, 482
performance engineer 105
persistence independence 391
persistence orthogonality 391
phishing 838, 840, 885
physical data independence 368
physical database design 381
physical design 447
PI 125
PID 263
Pig 408
pipe 260
pipelining 227
Platform as a Service 683, 685
Platform as a Service (PaaS) 687
pod 676, 682
pointer 158, 179, 314, 326
policy 243, 263
Pólya 433
polymorphism 335, 341
polynomial (P) time complexity class 125
portability 179, 345, 437
Postman 613
precedence 319
predictive analysis 406
preprocessing 340
preprocessor directive 340
prescriptive process model 441
presentation layer 54
Prim's algorithm 115, 122
primary key 368, 373
primary memory 249
primary storage 275
primitive data type 311
principle 520
principles 546
priority queue 91, 95, 118
Prisma 711

privacy 821, 822, 824, 827, 832, 833, 836, 839, 839, 841, 841, 843, 849, 851, 852, 887, 891, 894
Privacy Shield 404
private cloud 685, 690, 779
privileged instruction 245
privileged system program 252
problem 98
problem model 98
procedural language 304
procedural programming 154, 155, 185
procedure 154, 180
process 241, 243, 244, 247, 249, 252, 253, 257, 259, 260, 261, 262, 265, 268, 269, 271, 272, 273, 277, 280, 281, 307, 315, 329, 340, 343
process control block 263, 280
process control block (PCB) 259
process ID 263
process ID (PID) 258
process map 59
process synchronization 247
processor 13, 17, 198, 211
product owners 432
profiler 105, 477
program 90
program counter 244
program counter (PC) 214
programming 124
programming language 14, 15, 147, 150, 156, 177, 185
programming language paradigm 151
programming model 145, 175
project manager 432
properties 241
protection 239, 241, 246, 256, 260, 274, 276, 279, 281
protein-folding algorithm 113
prototyping model 460
pseudo-assembly 213
pseudocode 46
public cloud 661, 683, 685, 689, 691
public interface 329, 333
pure function 332

Q

quadratic-time algorithm 110
quality assurance tester 432
quantum computer 796
query processor 369
query tree 376
query-by-example (QBE) 377
quicksort algorithm 118
quota 275

R

race condition 309, 335
RAM 250
Random Access Machine 145
random access memory 250
random access memory (RAM) 198, 220
Ransomware 837, 838, 890
reachable vertex 121
React 604, 619, 624, 637, 645, 706
React Native 624, 637
readability 300
ready state 247, 263, 263
recovery 279, 281
recursion 47, 158, 179, 185
reduced instruction set computer (RISC) 212
reduction 126
reduction algorithm 115
redundant array of inexpensive disks (RAID) 382
refactoring 449, 473
reference variable 314
refresh cycle 220
register 196, 214, 258, 261, 263
register file 196
reinforcement 410
relation 373
relational algebra 374
relational database design (RDD) 380
relational database service (RDS) 768
relational DBMS 370
relative organization 382
relevance problem 363
reliability 240, 241, 255, 267, 275, 279, 283, 812, 816, 825, 833, 879, 881, 883, 887
remote procedure call 665
remote procedure call (RPC) 666
repetition 106
replication 278
repository 172
representation 90
representational state transfer 675
representational state transfer (REST) 675
requirements model 546
requirements modeling 445
resiliency 673, 679
resource group 706, 733
responsible computing 811, 814, 893, 894, 896, 898, 899
Responsible Computing 813, 832, 901
REST 706, 709
return 326, 332, 336
return on investment (ROI) 407
road map 510
robot manipulator 792
robotics 791
root node 93
round-robin scheduling (RR) 264
router 96
RPC 666
running state 247, 263, 263
runtime 106, 112, 117, 118, 121
runtime analysis 104
runtime error 331
runtime linking 171
Rust 179
rvalue 316

S

SaaS 689
sampling 406
scalability 337, 437, 451, 469, 482, 815, 816, 818, 821, 823, 827, 851, 871, 891
scenario 430, 446, 469, 473, 487
scheduling 204, 245, 257, 263, 264, 267, 275
SciDB 367
scope 304, 328, 332, 341
scripting language 307, 338
Scrum 449, 464

search tree property 93
searching 98, 100, 117
secondary memory 249, 283
secondary storage 275
secret and configuration management 765
security 437, 441, 452, 456, 469, 482, 486
security information and event management (SIEM) 839
security manager 369
segmentation fault 160
selection 321, 338
semantic analysis 341
semantic error 161
Semantic Web 570
semaphore 247
semistructured data 368, 373, 393
sentinel 323
separate compilation 343
Sequelize 709, 711
sequential execution 320
sequential file organization 382
sequential search 100, 111, 117
sequential search algorithm 117
serializer 586
server-side script 339
serverless computing 685, 692, 696, 696
service-oriented architecture (SOA) 666
set 91, 92
shallow machine learning 770
shared data 309, 332
shared library 169
shared memory 175
sharing 241, 244, 246, 261, 267, 268, 269, 274, 277, 279, 281
shell scripting 338
short circuiting 320
shortest path 98
shortest paths tree 122, 123
shortest remaining processing time (SRPT) 265
shortest time to completion first (STCF) 265
sidechain 574
signed integer 205, 206
simultaneous multithreading (SMT) 228
single inheritance 334
single-user DBMS 371
singleton 503
smart contract 76, 637, 640
snowflake schema 398
SOA 667
social determination of technology 28
social network 96
socket 154, 245, 252
software 14, 17, 18, 20, 21, 27, 29, 434, 451, 481
software architecture 447, 482
Software as a Service 685, 686
Software as a Service (SaaS) 687
software design 433, 449
software developer 432
software development kit (SDK) 762
software development life cycle (SDLC) 434, 440
software engineering 22, 99, 441, 449, 455, 468, 508, 545
Software Engineering Code of Ethics and Professional Practice 487
software license 484
software process improvement 467
Software Quality Management (SQM) 455
software security 853, 866, 897
software stack 548
solid-state drive (SSD) 223
Solidity 571, 639
solution architecture 52
solution architecture management 545
solutions architect manager 501, 545
solutions continuum 72
sorting 98, 117
source code 145, 162, 171, 176, 180, 329, 340, 343
space complexity 106
spanned record 382
Spark 366
spatial 225
spatial parallelism 228
spiral model 461
spot/not urgent compute 763
spreadsheet 20
sprint 465
SRAM 222
stack 213
stack allocation 270
stack frame 328
stack overflow 329
stack pointer (SP) 258
star schema 397
START 47
starvation 336
state 332
stateful application 563
stateless application 563
statement coverage 471
static data 258
static library 167
static linker 343
static quality 436
static random access memory (SRAM) 222
step 106
storage 13, 22, 193, 195, 198
storage access point 762
Storage Area Network (SAN) 388
storage manager 369
storage service 761, 781
string 299, 311, 338, 341
string data type 90
string problem 98
strongly typed 301, 313, 339, 341
structured data 368, 373, 412
structured programming 152, 158, 179
Structured Query Language (SQL) 308, 373
subclass 334
subject matter experts (SMEs) 432
subproblem 110
subsystem 508, 546, 548
superclass 334
superscalar capability 228
supersocieties 837, 890

supersociety 813, 830, 891, 892, 893
survivability 816, 817
symbol 164, 167, 172
symbol resolution 164
symbol table 165
symmetric multiprocessor (SMP) 177
synchronization 257, 265, 266, 309
synchronous call 386
syntax analysis 341
system architecture 67, 73
system call 245, 248
system interrupt 245
system software 434
system testing 451, 474
systems software 150, 156

T

tablespace 383
tailorability 815
Tanzu 683, 702, 725, 726, 727, 732
target 100, 117
technical architecture 67
Technical Reference Model 813
Technical Reference Model (TRM) 812
technological fix 31
telecommand 768
telemetry 768
temporal locality 225
temporal parallelism 228
ternary operator 317
Terraform 682, 701
test-driven development (TDD) 472
The Open Group Architecture Framework (TOGAF) 446, 524
the Web 560
THEN 47
theoretical computer science 25
theta join 376
thin client variant 386
thrashing 273
thread 175, 244, 248, 261, 265, 266, 273
thread of control 335
threads 177
throughput 437
time complexity 106, 107
time slice 261, 268
TLB 271
TOGAF 55, 70
tokens 341
top-down approach 92
top-tested loop 323
total cost of ownership (TCO) 407
tour 126
tractable 125
traditional process model 440
transaction 367, 372, 387, 392, 397
transaction management 387
transfer learning 410
transistor 202, 203
translation 383
translation layer 224
translation lookaside buffer (TLB) 271
traveling salesperson problem (TSP) 126
traversal 98, 120
tree 93
trigger 374, 392, 402
Truffle Framework 639
truth table 318
tuple 376, 396
tuple and document store 394
Turing machine 124
Turing-complete 14
two-factor authentication 280
two-phase commit (2PC) 388
two's complement 207
type cast 313

U

umbrella activity 452
unary operator 317
understandability 816
Unified Modeling Language (UML) 445
Unified Process (UP) model 462
uniform memory access (UMA) 249
unikernel 694
UNIQUE 385
uniqueness constraint 378
unit testing 438, 470
Unix 166
unsigned integer 206, 207, 210
unstructured data 366, 368, 391, 400, 402, 410
unweighted shortest path 121, 122
usability 437, 469, 475
usability testing 474
use case 444
user experience (UX) 54
user experience (UX) designer 432
user interface (UI) 305
user interface/user experience (UI/UX) 474
user story 444

V

V-model 458
vacuum tube 13
validation 458, 470
value 403
variable 301, 311, 315, 332, 341
variable declaration 315
variety 403
vector instructions 217
velocity 403
veracity 403
verification 458, 470
version control 172
version control system 451, 477
vertex 96
vertical fragmentation 387
View 563, 586
virtual compute service (VCS) 763
virtual data mart 400
virtual data warehouse 399
virtual functional and serverless compute service 763
virtual local area network (VLAN) 821
virtual local area networks 821
virtual machine 248, 253, 253, 275, 282, 283
virtual machine (VM) 243, 693
virtual memory 204, 248, 254, 264, 267, 269, 272, 277

virtual reality (VR) 780
virtualization 243, 256, 267, 283, 695, 699
Visual C++ 161
VM 247, 254, 282, 283
void 327
volume 403

W

walled garden approach 820
waterfall model 457
weak entity 380
weakly typed 301
Web 96
Web 1.0 560
Web 2.0 73
Web 3.0 73, 637
web application (web app) 74
web application framework 74, 706
web page 560, 613
web publishing 560
web server 560, 606, 618, 635
web service 665, 666, 706, 709, 711
weighted shortest path 123
white box testing 470
Windows Paint program 45
working set size (WSS) 272
workload 679, 682, 691, 696, 701
World Wide Web 560
World Wide Web Consortium (W3C) 73
worst-case situation 106
writability 300

X

Xcode 569, 625
XML DBMS 370

Y

Yet Another Resource Negotiator (YARN) 408

Z

zero-knowledge proof 850
zero-knowledge proof (ZKP) 849
zero-knowledge rollup (zk-rollup) 575
ZKPs 849