



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. Be sure to retain key word tagging on computational thinking when sentence is revised. 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 \(https://openstax.org/r/76CompThinking\)](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.

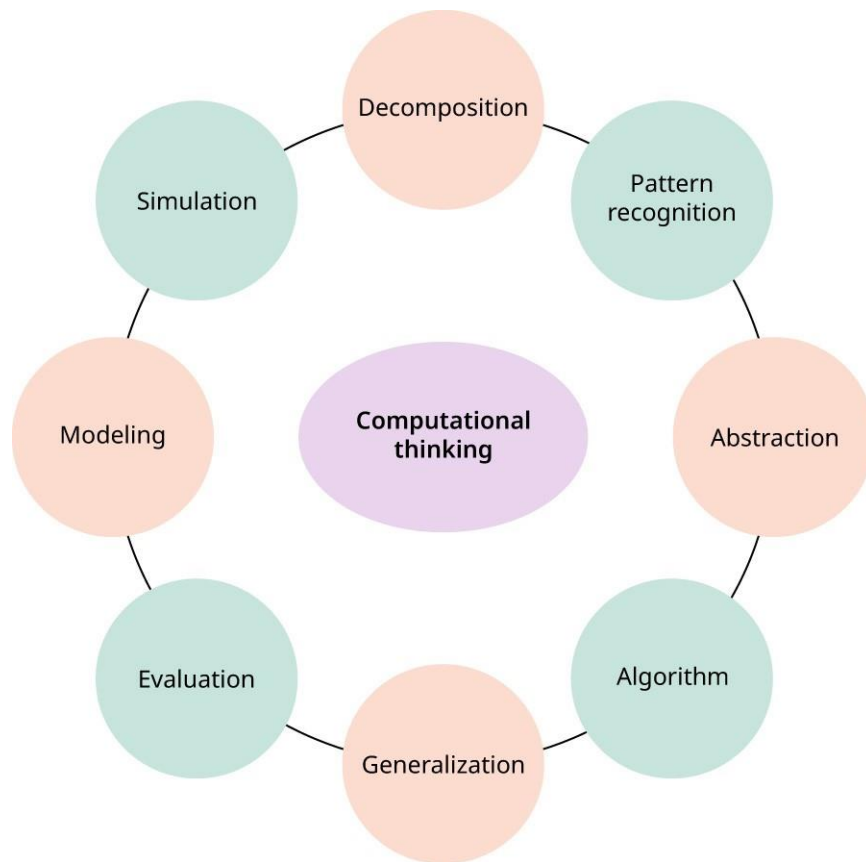


Figure 2.2 This diagram illustrates the main components of computational thinking.

(attribution: Copyright Rice University,

OpenStax, under CC BY 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.

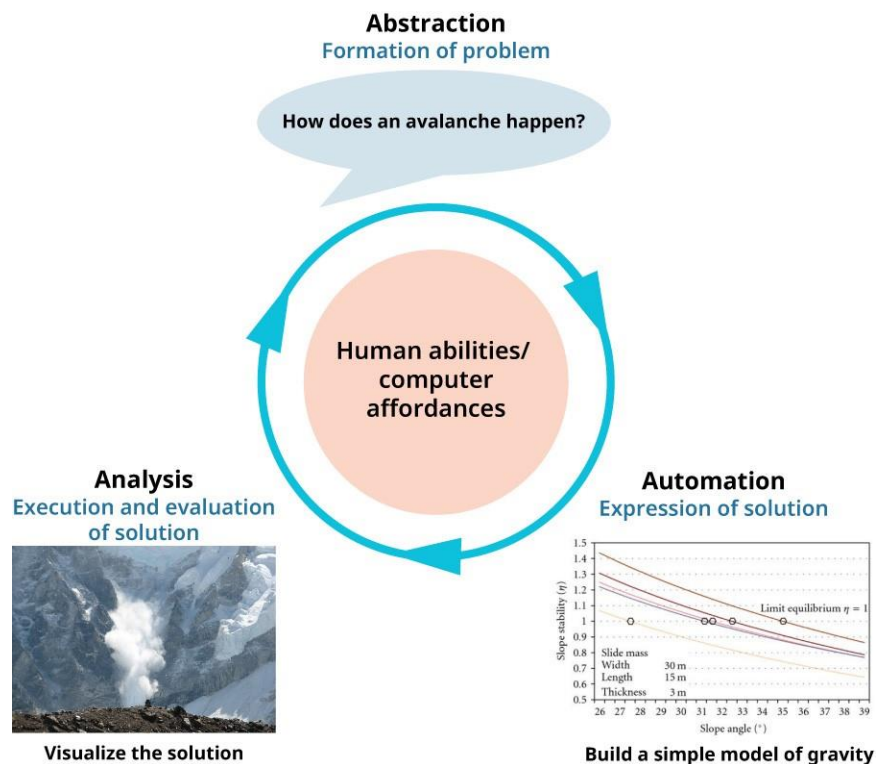


Figure 2.3 The three As—abstraction, automation, analysis—illustrate the power of computational thinking. (credit photo: modification of “Avalanche on Everest” by Chagai/Wikimedia Commons, Public Domain; credit graph: modification of “Slope stability calculation for a model landslide” by B. Terhost and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

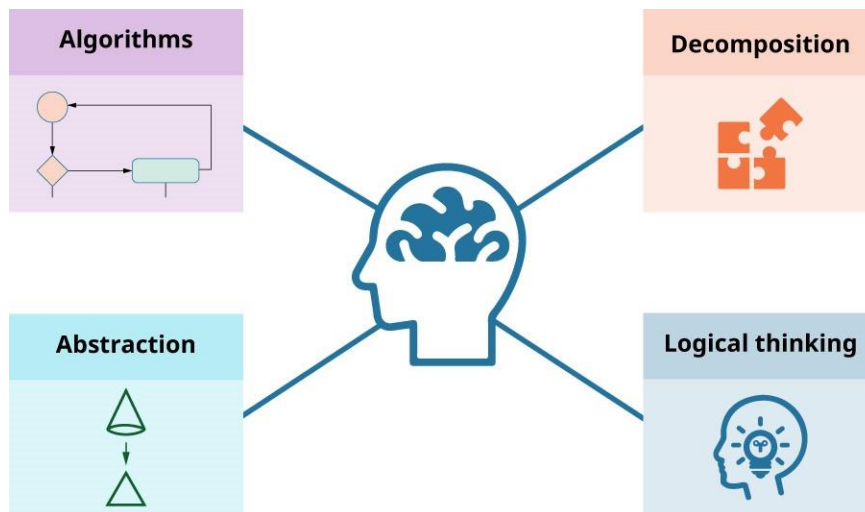


Figure 2.4 Users can explore the essence of computational thinking through decomposition, logical thinking, abstraction, and algorithms. (attribution: Copyright Rice University, OpenStax, under CC BY 4.0 license)

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](#).

Ingredient	Equipment	Action
Bread	Plate	Repeat
Jam	Knife	Left hand
Butter		Right hand
		Pick
		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](#).

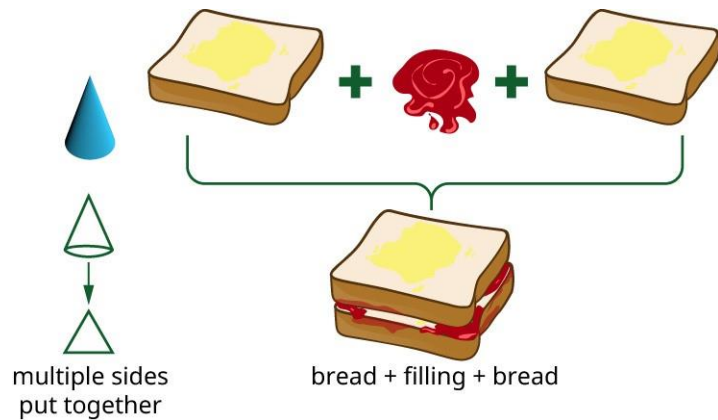


Figure 2.5 This jam sandwich abstraction example illustrates what the final product should look like. (attribution: Copyright Rice

University, OpenStax, under CC BY 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](#)).

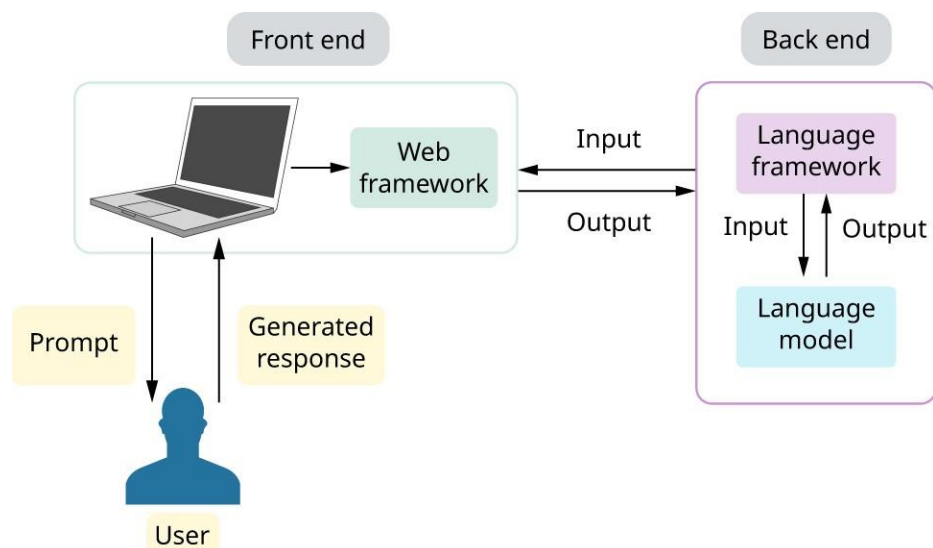


Figure 2.6 When using GenAI, a user interacts with the interface while the application processes the prompt with layers of abstraction on the back end. (attribution: Copyright Rice University, OpenStax, under CC BY 4.0 license)

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](#).

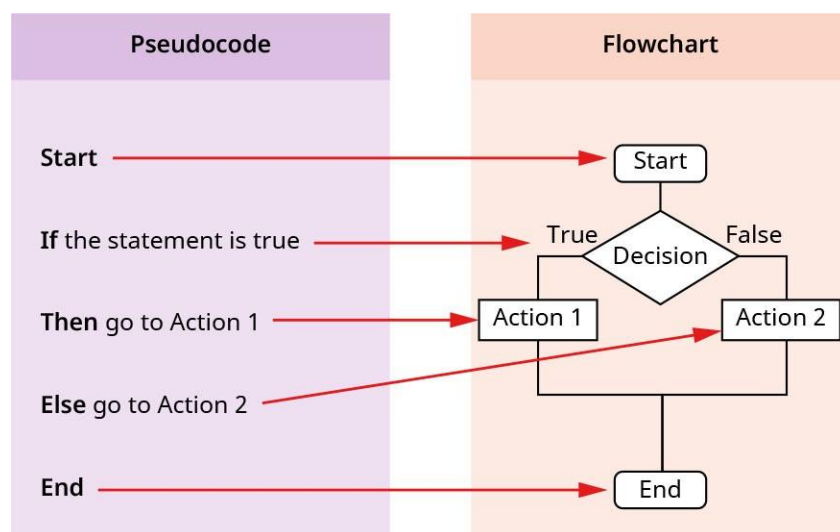


Figure 2.7 Pseudocode lists each step, while a flowchart visually outlines the process of decision-making. (attribution: Copyright Rice

University, OpenStax, under CC BY 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](#).

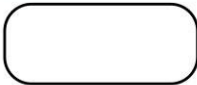
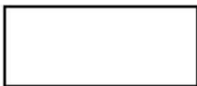
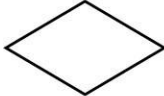

Symbol	Instruction
	Start/end
	Task (e.g., spread jam)
	Decision (e.g., do you want butter?)
	Direction of flow

Figure 2.8 The symbols used in a flowchart are associated with their instructions.
(attribution: Copyright Rice University, OpenStax, under CC BY 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: LowLevel 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.

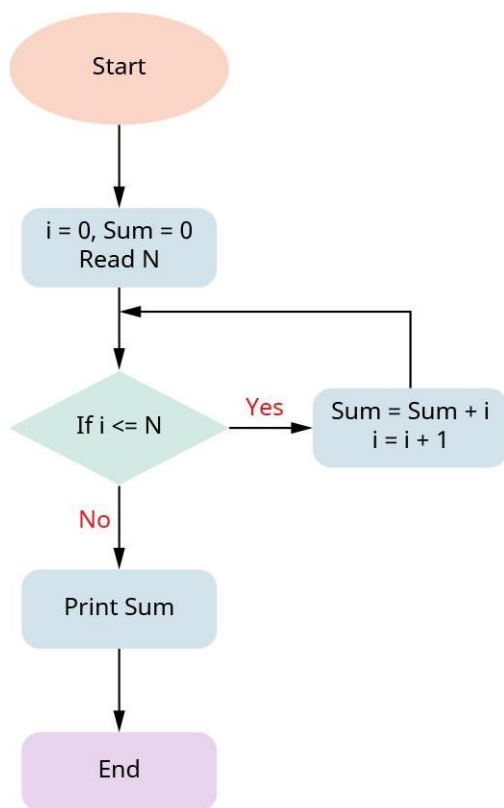


Figure 2.9 A flowchart represents an iterative solution for adding N numbers. (attribution: Copyright Rice University, OpenStax, under CC BY 4.0 license)

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.

Tes Cas	Inpu	Proble	Expecte Outcome	Observe Outcome	Improvement	Responsib Use
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.

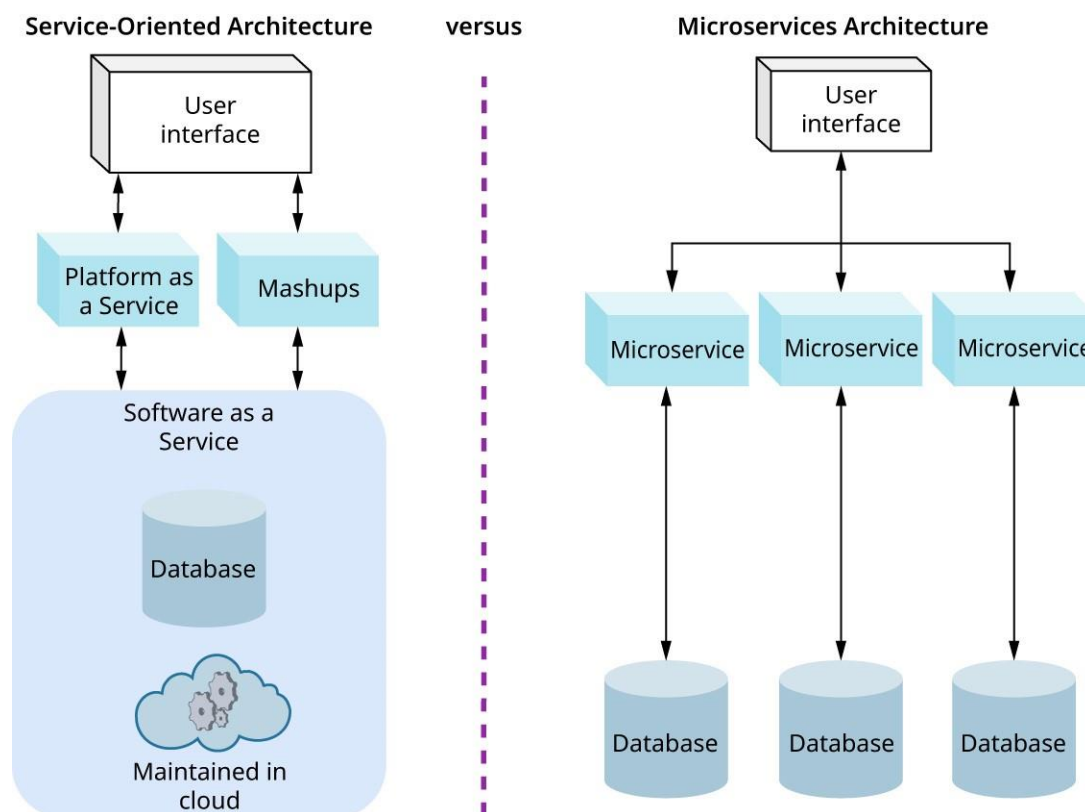


Figure 2.10 The different layered architectures include tiered architectures, service-oriented architectures (SOA), and microservices architectures. (attribution: Copyright Rice University, OpenStax, under CC BY 4.0 license)

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
--------------	---------	------------

	Defines the organization's business	
--	-------------------------------------	--

Business	Business models, processes, capabilities, and strategy, goals, processes, and architecture	organizational structure functions
----------	--	------------------------------------

Information	Data models, information flow diagrams, data	
-------------	--	--

Manages data and information assets		
architecture	governance, data standards, and metadata	
Application	Designs and organizes software	Application portfolio, application
integration,		
architecture	applications and interface design	
Technology	Specifies the hardware, software, and	Servers, networks, databases, cloud
services, architecture	technology infrastructure	security protocols
Security	Ensures the protection of information	Security policies, access controls,
and architecture	assets, systems, and networks	encryption mechanisms
Integration	Facilitates seamless communication and	Middleware, messaging systems,
and		
architecture	data exchange integration patterns	

Process Defines and optimizes business Process models, workflow diagrams, and architecture processes 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 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	Responsibiliti	Reportin
DBA administrato	Managing and monitoring the activitie	IT manager or VP of
CIO (chief officer	Overall IT	CEO (chief officer
HR	Human resources	HR

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](#).

Titl	Roles and
Trader	<ul style="list-style-type: none"> • Manage trading • Develop strategies to maximize profits and manage
Sales	<ul style="list-style-type: none"> • Build and maintain client • Understand client needs and offer suitable products and
Back-office ³	<ul style="list-style-type: none"> • Process and settle • Handle trade confirmation and
Risk	<ul style="list-style-type: none"> • Monitor and manage risk • Ensure compliance with risk management
Legal	<ul style="list-style-type: none"> • Provide legal advice and • Ensure compliance with laws and
Technology	<ul style="list-style-type: none"> • Develop and maintain trading systems and IT • Ensure the security and efficiency of technology
Human	<ul style="list-style-type: none"> • Manage recruitment, training, and employee • Ensure the organization is staffed with skilled and motivated

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

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.

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 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.

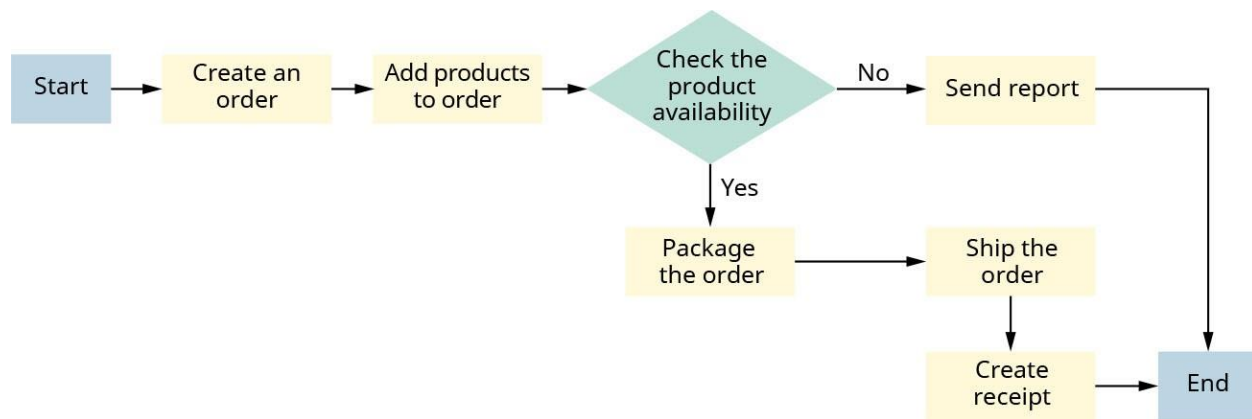


Figure 2.12 This flowchart outlines the business process model for creating an order, checking availability, shipping, and payments. (attribution: Copyright Rice University, OpenStax, under CC BY 4.0 license)

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.

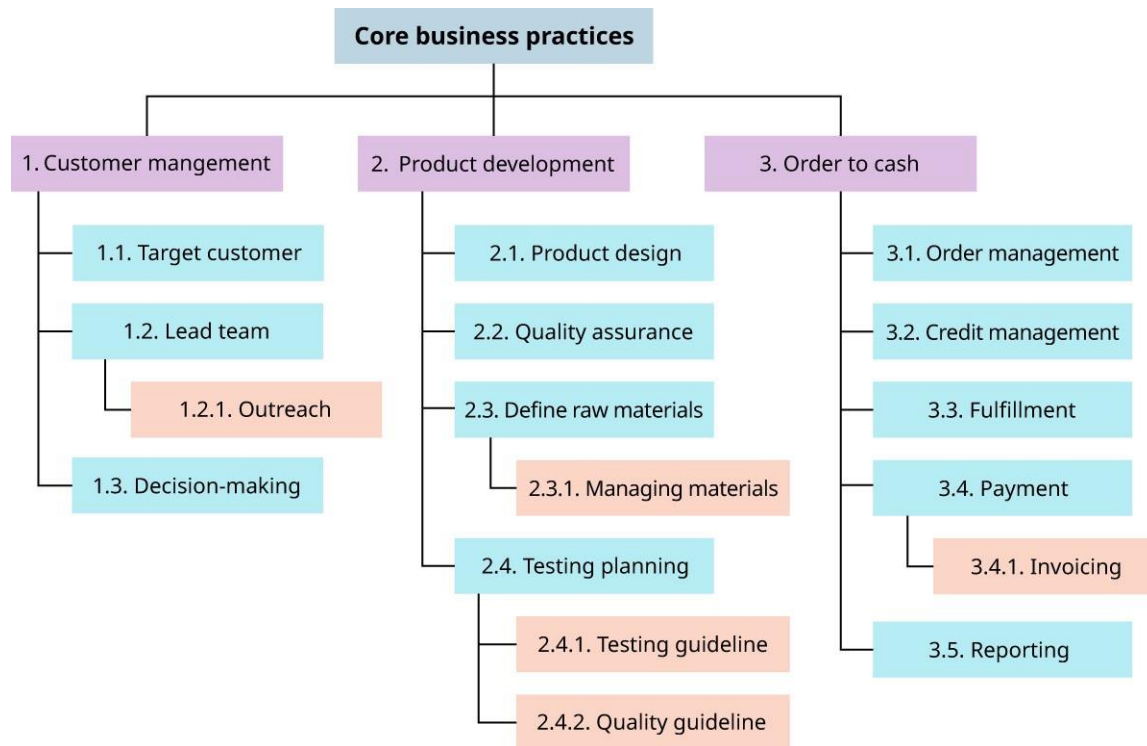


Figure 2.13 The trading business process hierarchy includes activities such as customer management, product development, quality control, order fulfillment, and customer payments. (attribution: Copyright Rice University, OpenStax, under CC BY 4.0 license) At the top of the hierarchy, there are three 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.

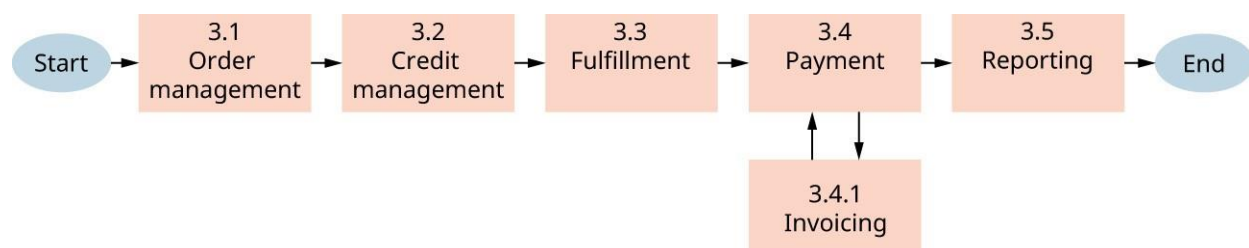


Figure 2.14 The order to cash business process flows from order management through product fulfillment and payment. (attribution: Copyright Rice University, OpenStax, under CC BY 4.0 license)

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.

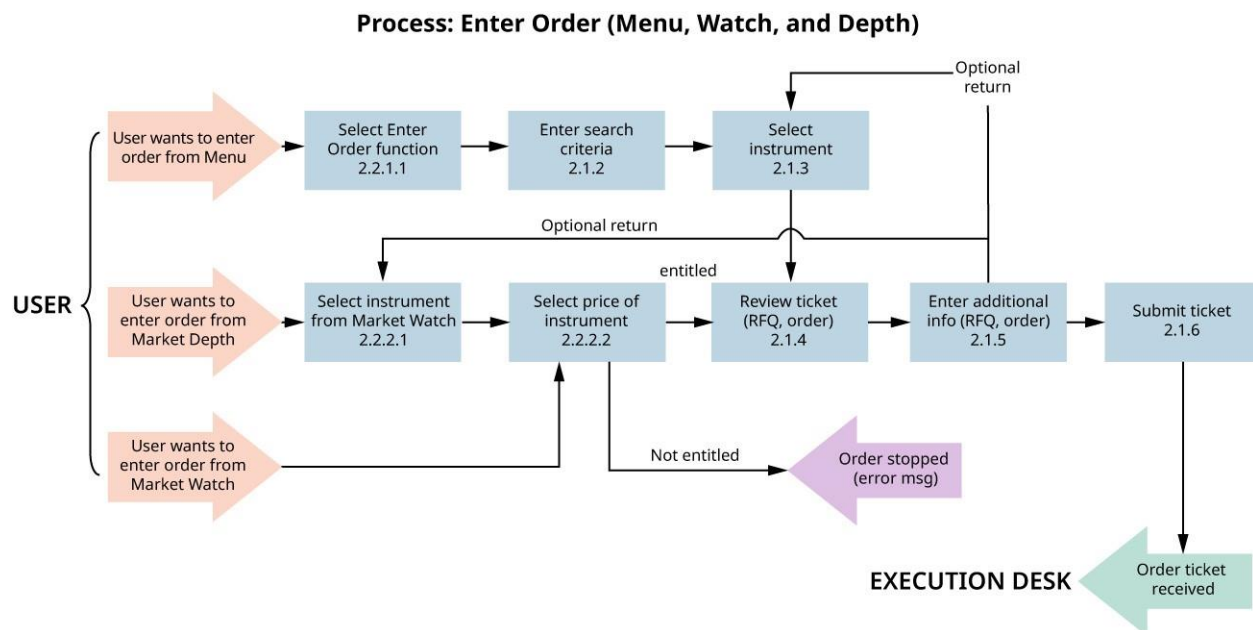


Figure 2.15 This sample trading business model represents a process map diagram for the enter order process. (attribution: Copyright Rice University, OpenStax, under CC BY 4.0 license)

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 systems⁴ 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.

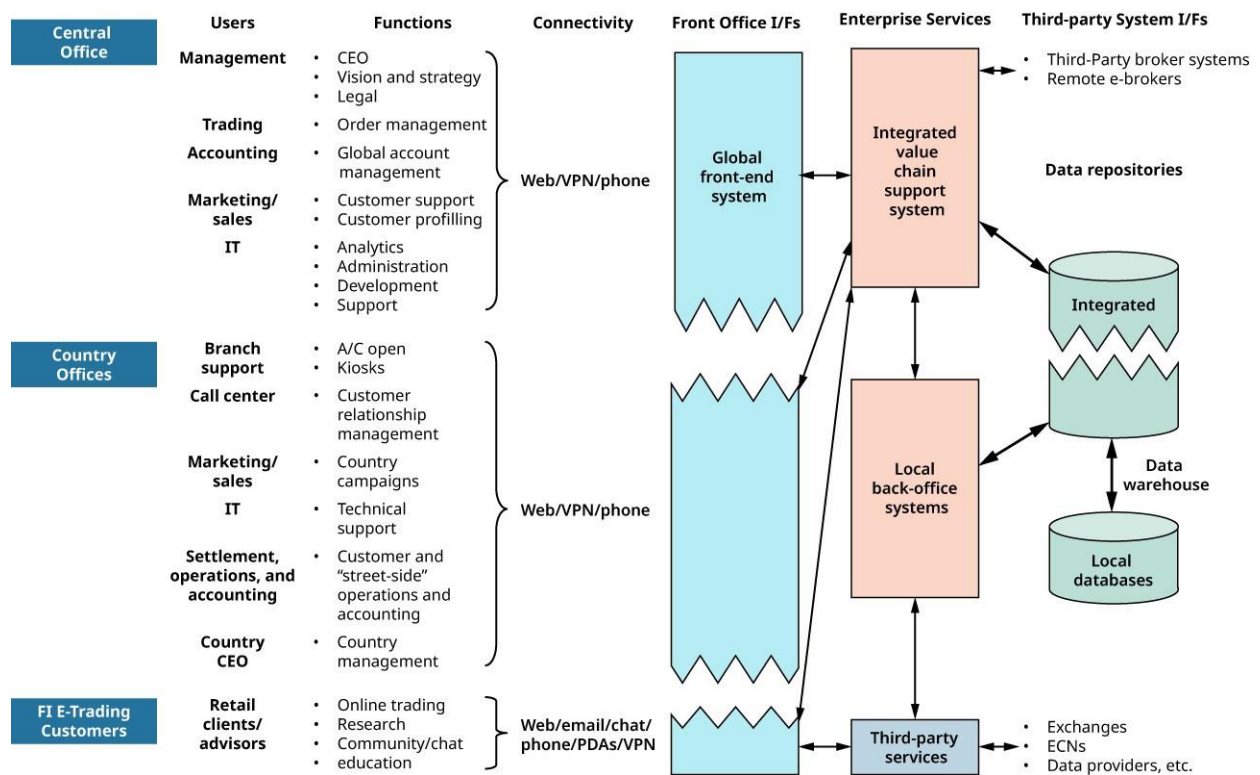


Figure 2.16 A conceptual trading application architecture moves through various levels, including users, functions, enterprise services, and third-party systems. (attribution: Copyright Rice University, OpenStax, under CC BY 4.0 license)

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.

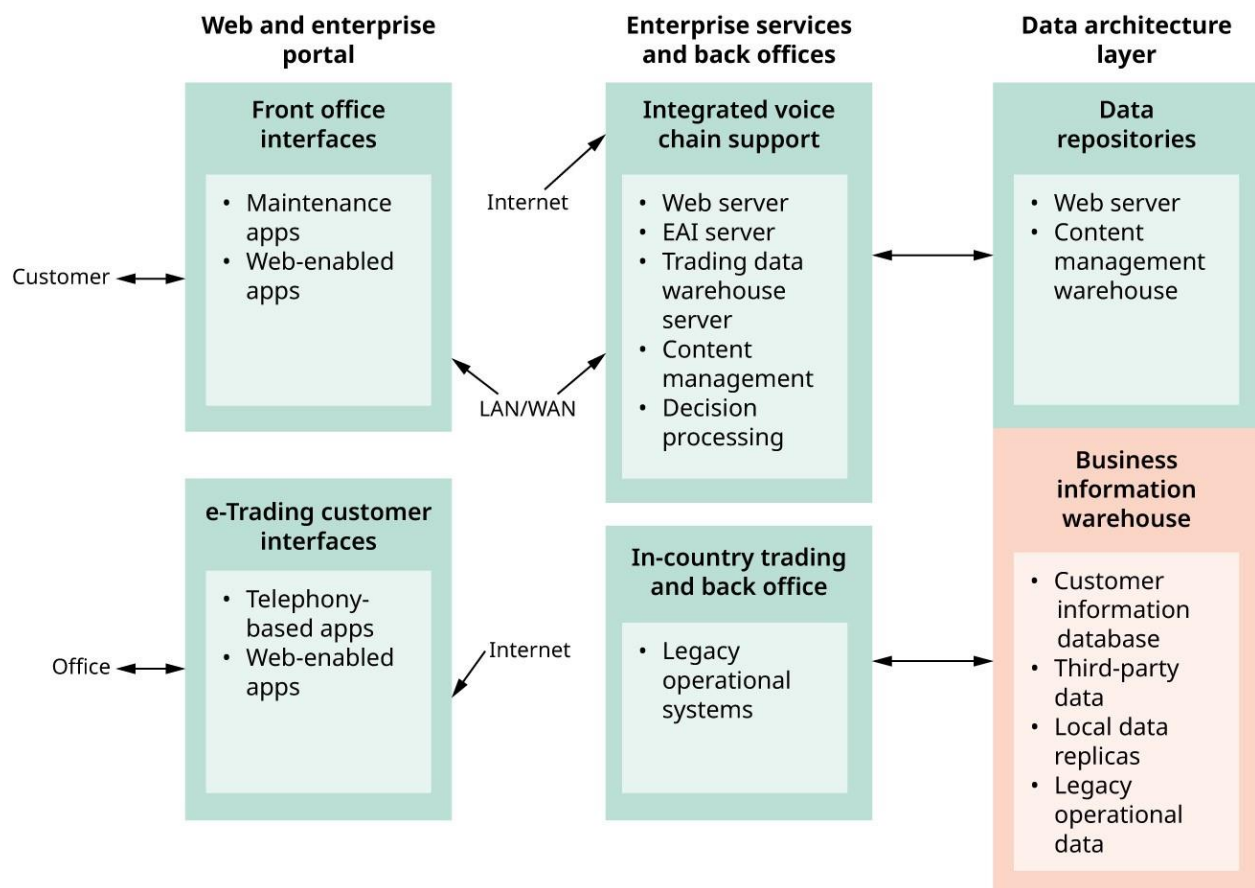


Figure 2.17 The logical trading application architecture illustrates each function block for the processing of customer orders.

(attribution: Copyright Rice University, OpenStax, under CC BY 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.

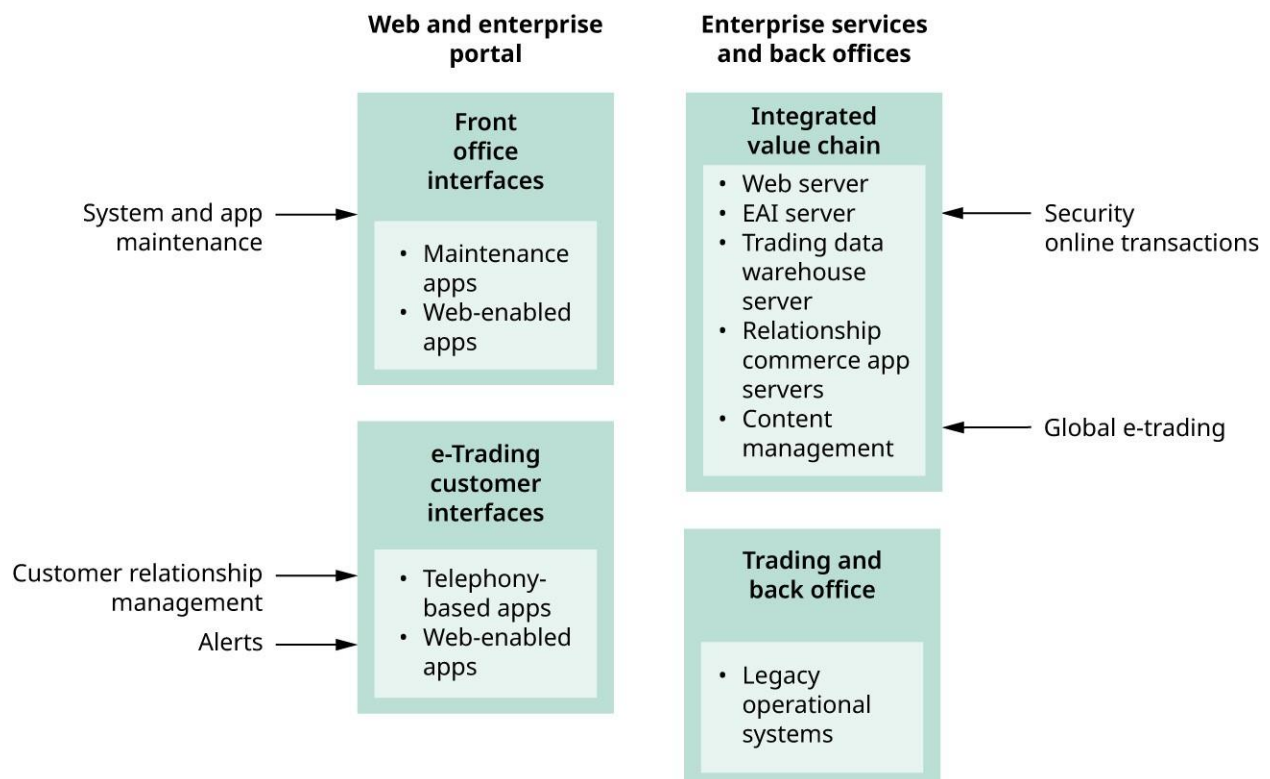


Figure 2.18 This logical trading application architecture includes function callouts that identify the technologies used for this process. (attribution: Copyright Rice University, OpenStax, under CC BY 4.0 license)

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](#).

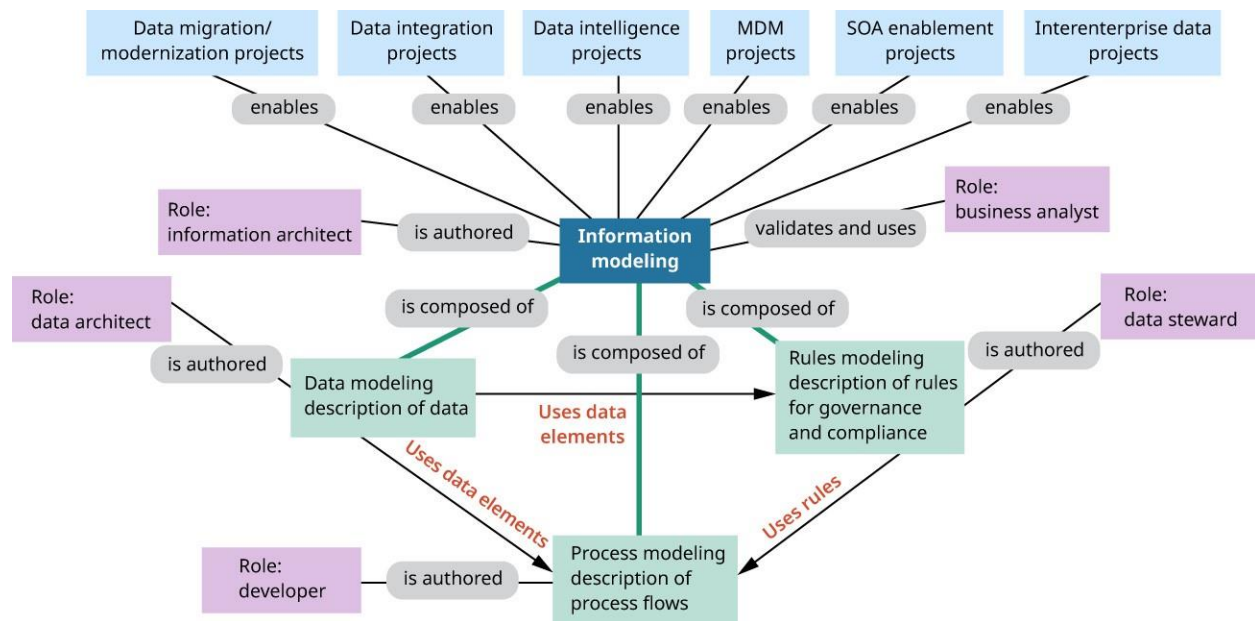


Figure 2.19 Examining and learning from data is integral to an organization's success, as outlined in this role of information modeling figure. (attribution: Copyright Rice University, OpenStax, under CC BY 4.0 license)

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.

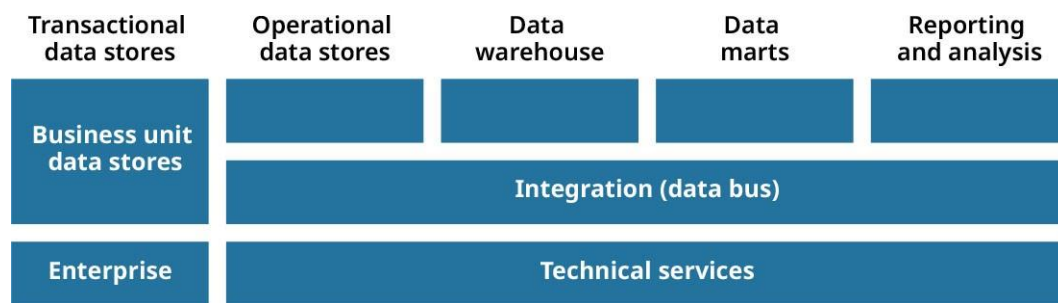


Figure 2.20 The enterprise conceptual data architecture for a fictitious insurance company highlights the different ways in which data is handled and managed. (attribution: Copyright Rice University, OpenStax, under CC BY 4.0 license)

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.

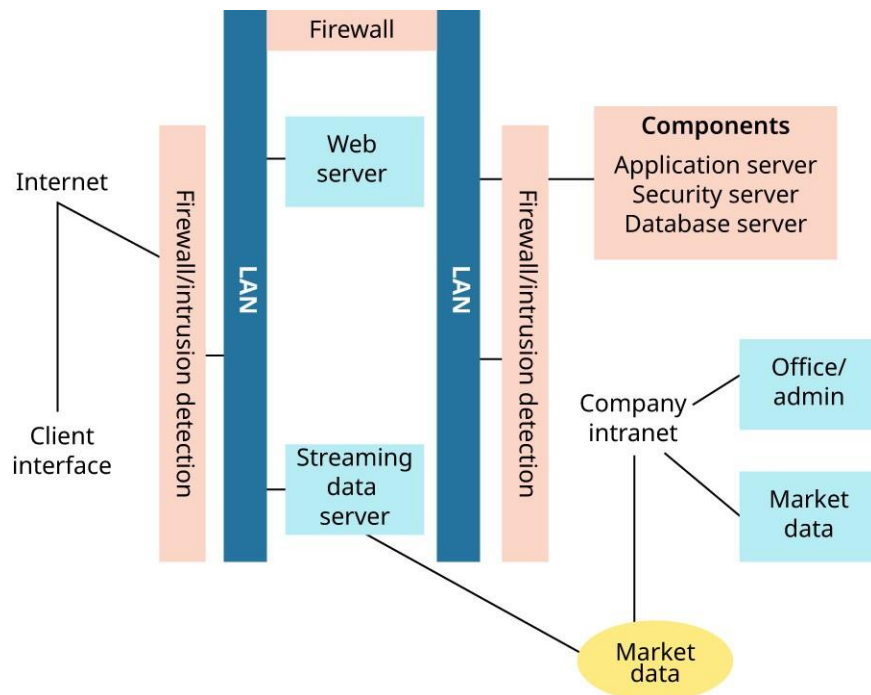


Figure 2.21 The physical trading application architecture highlights the use of hardware to meet the organization's goals. (attribution: Copyright Rice University, OpenStax, under CC BY 4.0 license)

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.

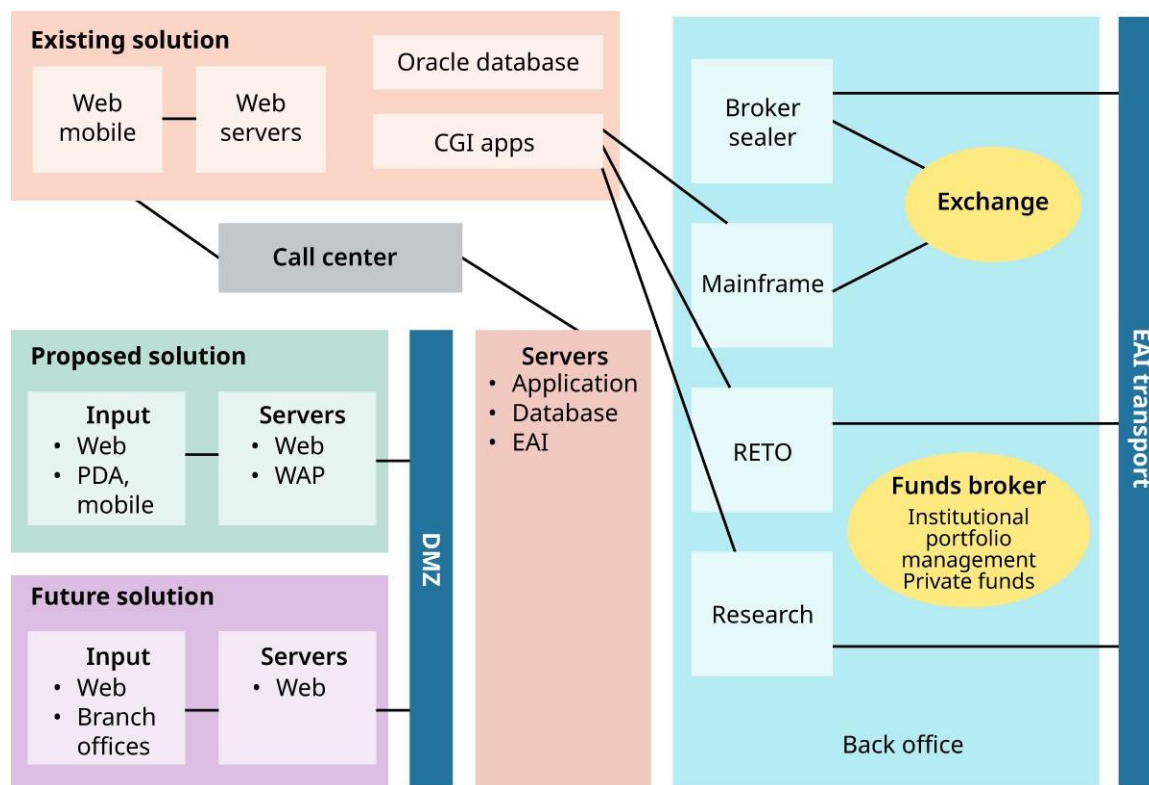


Figure 2.22 This alternative physical trading application architecture outlines possible changes from the existing web solution. (attribution: Copyright Rice University, OpenStax, under CC BY 4.0 license)

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.

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

Figure 2.23 TOGAF architectural dimensions include various levels of abstraction.
(attribution: Copyright Rice University, OpenStax, under CC BY 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 systemlevel 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](#).

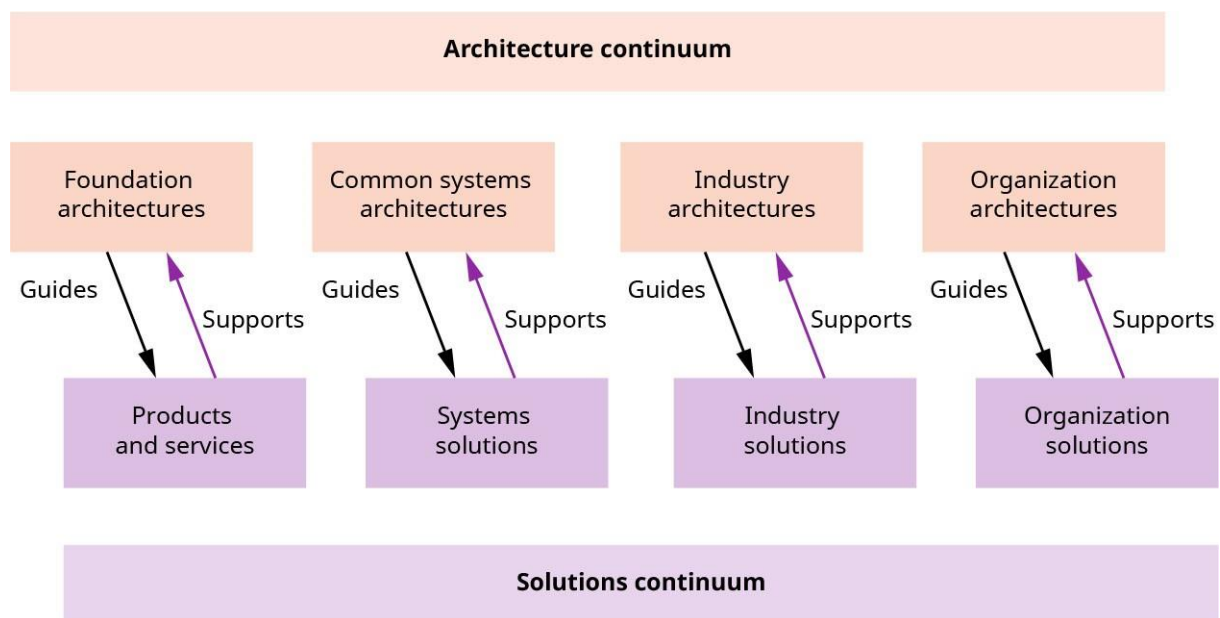


Figure 2.24 This TOGAF solutions continuum illustrates how each architecture guides and supports the others. (attribution: Copyright Rice University, OpenStax, under CC BY 4.0 license)

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.

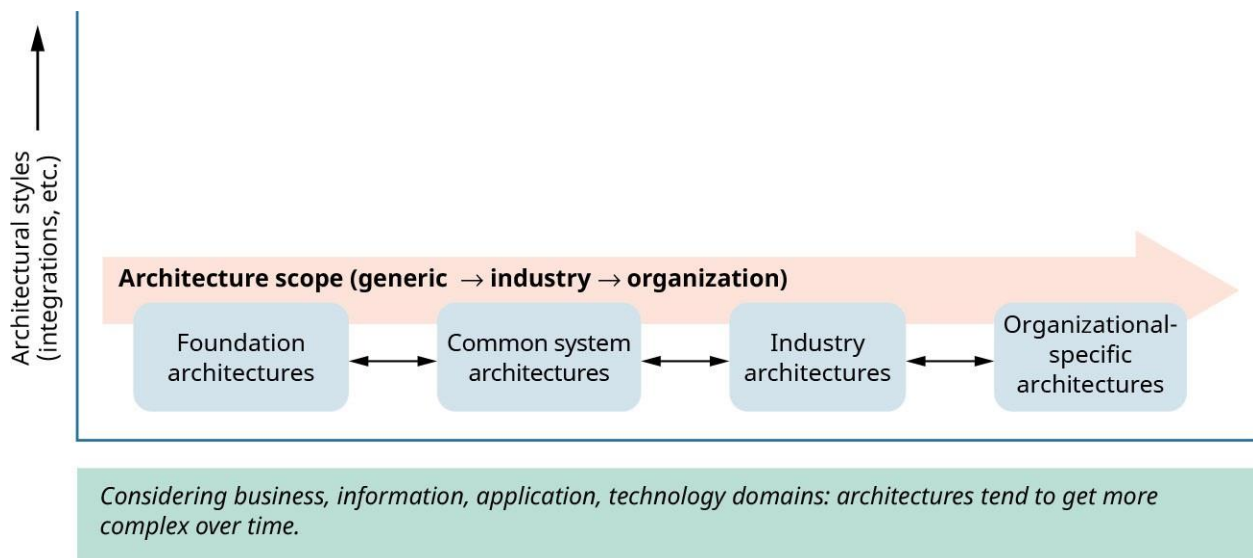


Figure 2.25 The TOGAF architecture continuum can suggest extensions but may only be able to focus on one aspect at a time. (attribution: Copyright Rice University, OpenStax, under CC BY 4.0 license)

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](#).

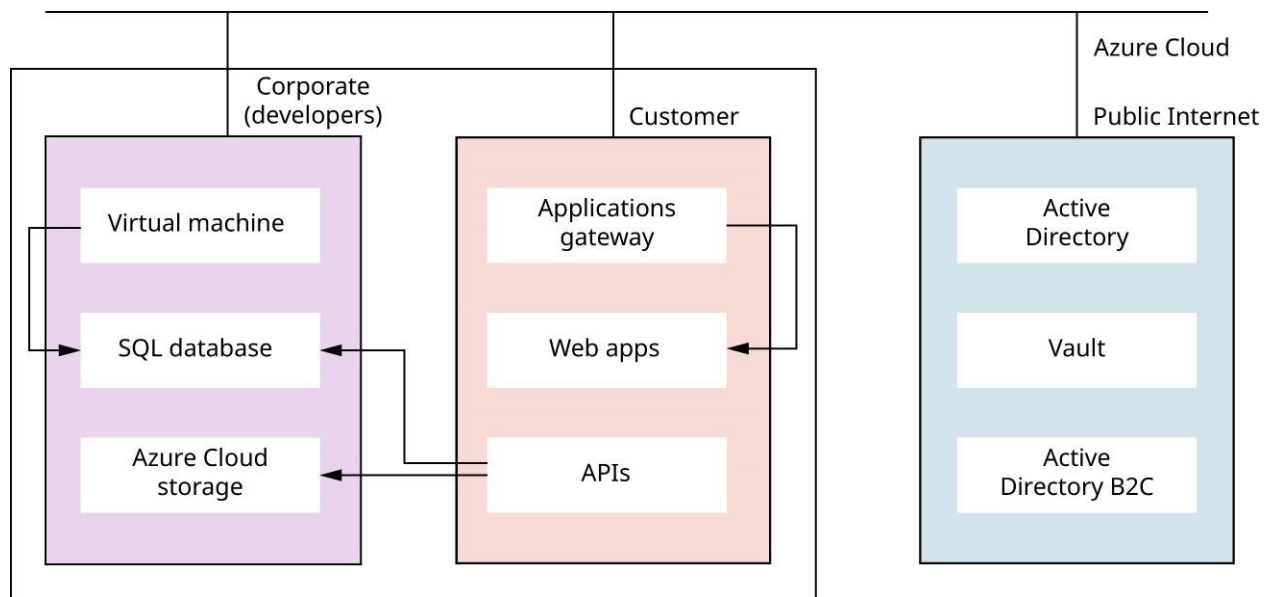


Figure 2.26 The logical application architecture of Microsoft Azure-hosted web applications allows for responsive web and mobile solutions for users. (attribution: Copyright Rice University, OpenStax, under CC BY 4.0 license)

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 commercialgrade 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 solutions at different

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](#).

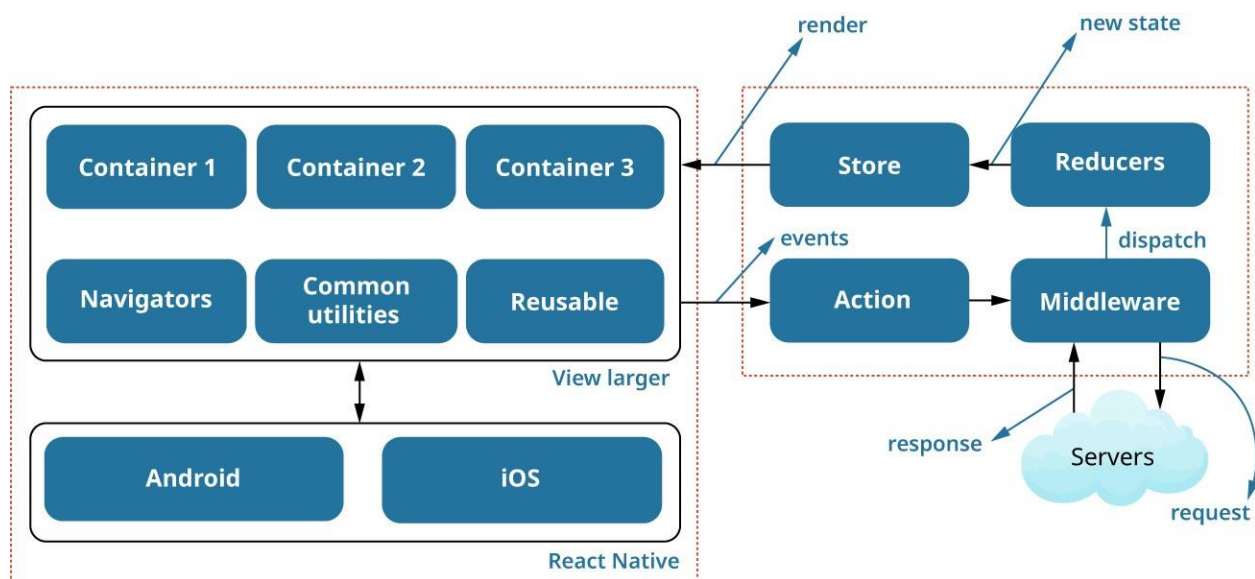


Figure 2.27 The logical application architecture of React Native mobile web apps shows the back-end processes that allow both

Android and iOS customers to use the same application. (attribution: Copyright Rice University, OpenStax, under CC BY 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.

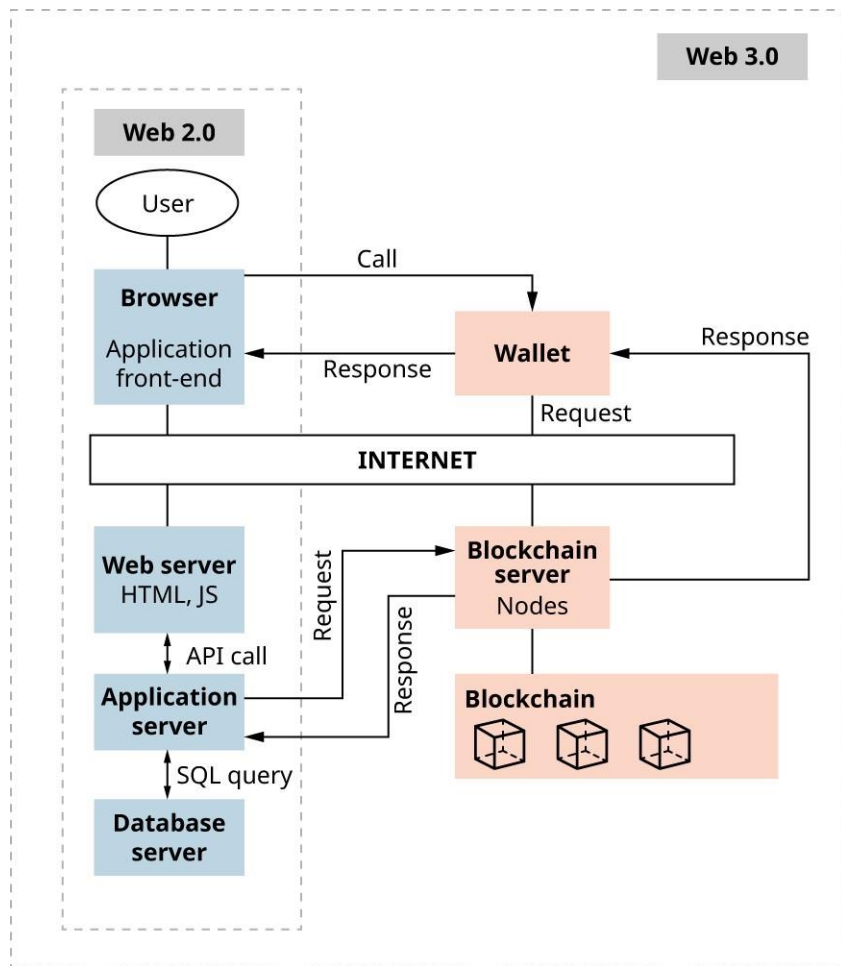


Figure 2.28 The flowchart show the logical application architecture of a Web 2.0 and Web 3.0 Website. (attribution: Copyright Rice

University, OpenStax, under CC BY 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, Consensus, 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](#).

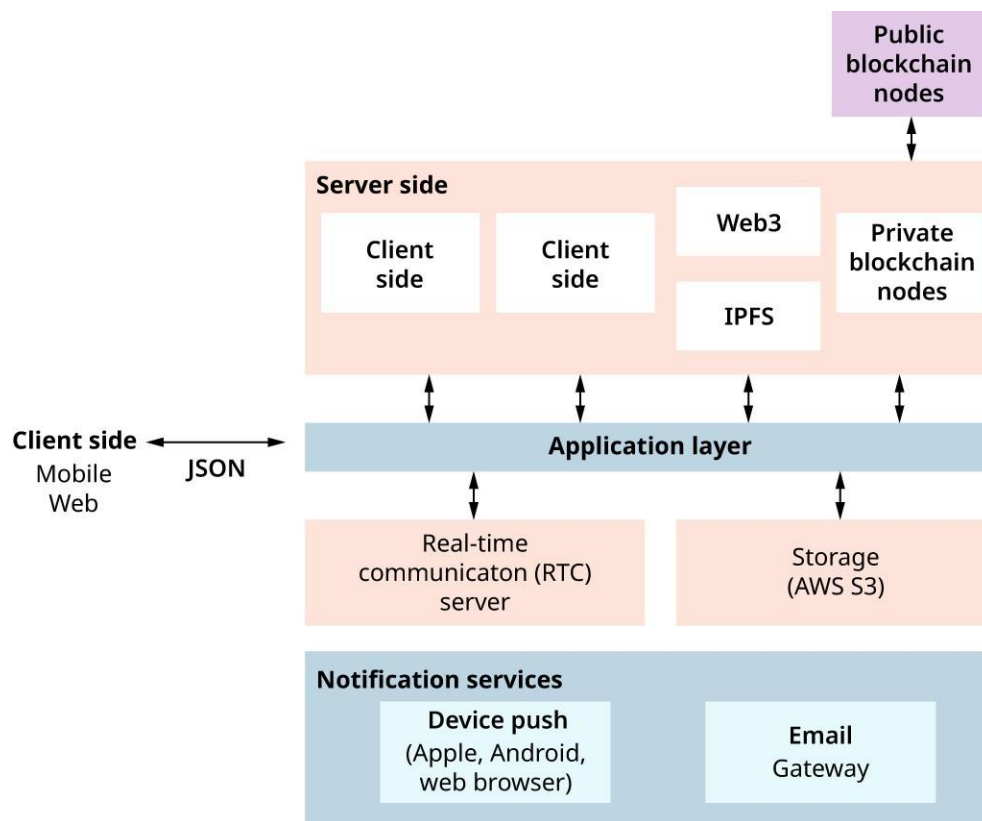


Figure 2.29 The cloud-native application architecture view of a digital enterprise shows both client-side and server-side processes through each layer. (attribution: Copyright Rice University, OpenStax, under CC BY 4.0 license)

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.

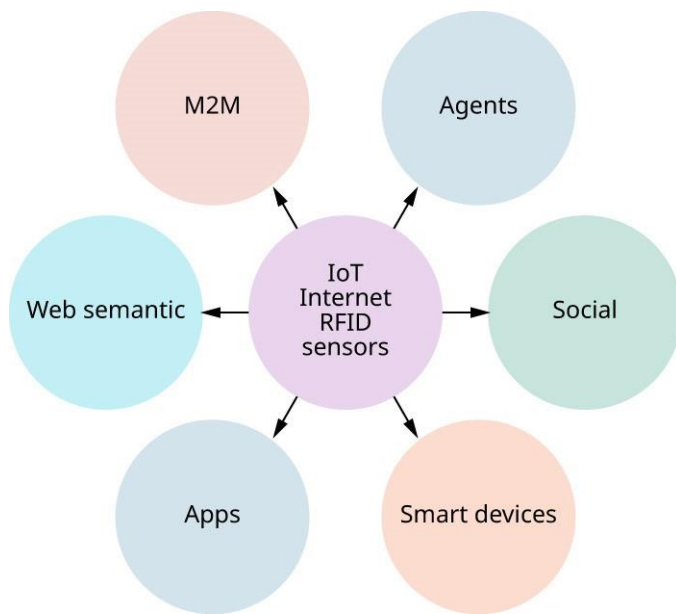


Figure 2.30 IoT devices use sensors, applications, and connectivity to interact, collect, and exchange data. (attribution: Copyright

Rice University, OpenStax, under CC BY 4.0 license)

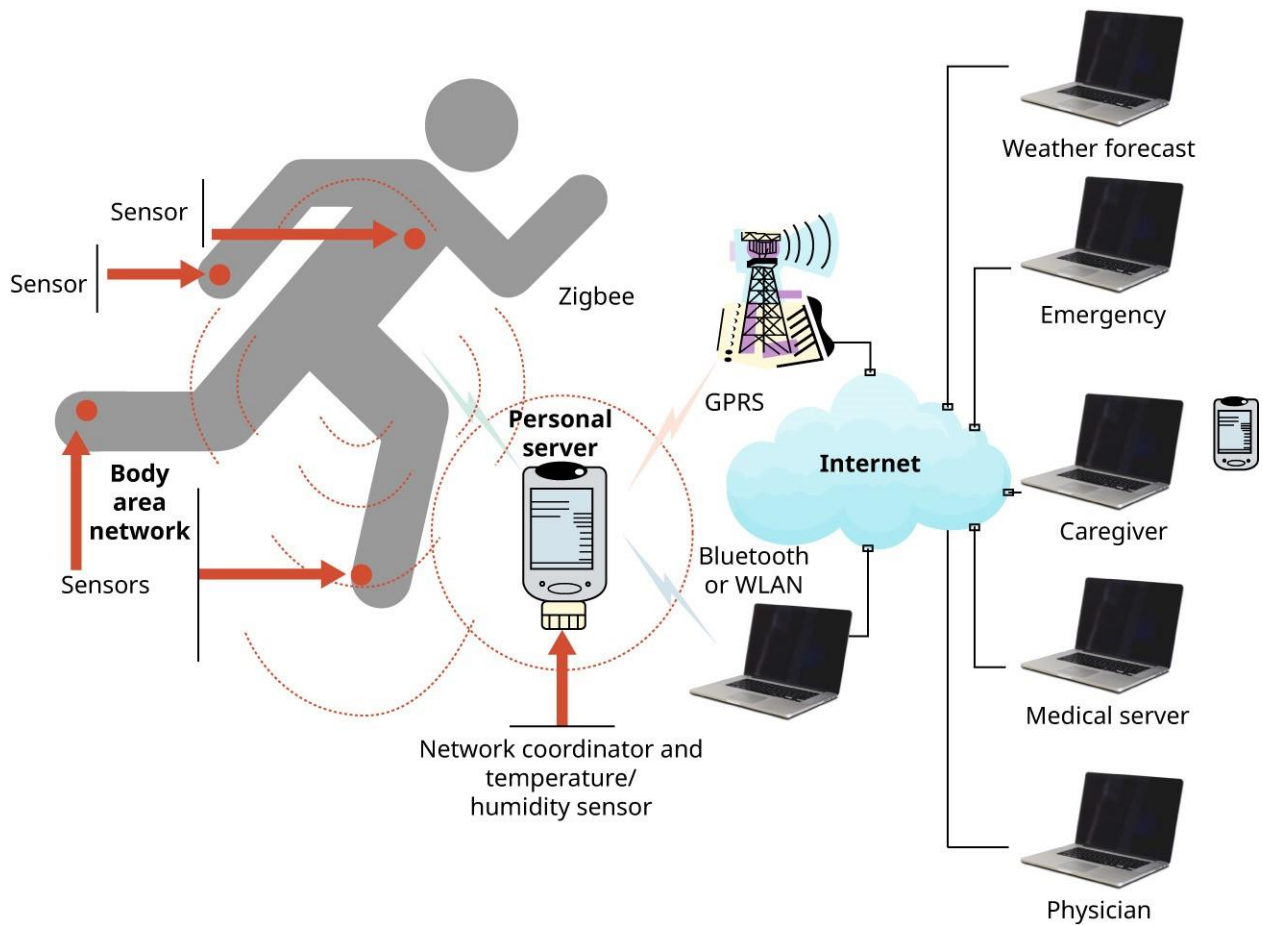


Figure 2.31 This diagram depicts an architecture of a body area network. (credit: modification of "Body Area Network" by

"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 longterm 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.