Computer Science as a Discipline

This chapter uses narrative, illustrations, and review questions to introduce computer science and technology concepts.

Having surveyed the relationships of computer science with other disciplines, it remains to answer the basic questions: What is the central core of the subject? What is it that distinguishes it from the separate subjects with which it is related? What is the linking thread which gathers these disparate branches into a single discipline? My answer to these questions is simple—it is the art of programming a computer. It is the art of designing efficient and elegant methods of getting a computer to solve problems, theoretical or practical, small or large, simple or complex. It is the art of translating this design into an effective and accurate computer program.

C.A.R. Hoare
Essays in Computing Science

Computer Science is the first engineering discipline in which the complexity of the objects created is limited solely by the skill of the creator, and not by the strength of raw materials.

Brian K. Reid
Communications of the ACM, October 1987

In theory, there is no difference between theory and practice. But, in practice, there is.

Jan van de Snepscheut

Most traditional sciences, such as biology and chemistry, trace their origins back hundreds or even thousands of years. Since electronic computers were not introduced until the 1940s, computer science is clearly a much newer field of study. Nonetheless, computers and related technology have made an astounding impact on our society in this short period of time.

This chapter presents an overview of computer science as a discipline, emphasizing its common themes (hardware, software, and theory) as well as its diversity. As you will learn, computer science comprises much more than just the study of computers. It encompasses all aspects of computation, from the design and analysis of algorithms to the construction of computers for carrying out those algorithms. The jobs performed by computer scientists are as varied as computer science itself, and this chapter examines
the roles played by computational theorists, programmers, systems architects, and microchip designers, among others. To further your understanding of this diverse field, we delve into specific computer science subfields, providing details on topics such as systems architecture, operating systems, networks, and artificial intelligence. We conclude with a discussion of ethical conduct within the computing profession.

**Computer “Science”**

Although government and industry began embracing computer technology as early as the 1940s, the scientific community was slower to acknowledge computer science’s importance. Most colleges and universities did not even recognize computer science as a separate field of study until the 1970s or 80s. Many people still argue that computer science is not a science in the same sense that biology and chemistry are. In fact, computer science has a lot in common with the natural sciences, but it is also closely related to other fields, such as engineering and mathematics. This interdisciplinary nature has made computer science a difficult field to classify. Biology can be neatly described as the study of life, whereas chemistry can be defined as the study of chemicals and their interactions. Is computer science, as its name suggests, simply the study of computers?

Although computers are the most visible feature of computer science, it would be inaccurate to characterize the field as dealing only with machinery. A more inclusive definition would be that computer science is the study of *computation*. The term computation denotes more than just machinery—it encompasses all aspects of problem solving. This includes the design and analysis algorithms, the formalization of algorithms as programs, and the development of computational devices for executing those programs. The computer science field also addresses more theoretical questions, such as those surrounding the power and limitations of algorithms and computational models.

Whether this combination of problem solving, engineering, and theory constitutes a “science” is a matter of interpretation. Some define “science” as a rigorous approach to understanding complex phenomena and solving problems. The process developed by the scientific community for examining observations and events, commonly referred to as the scientific method, can be summarized as an algorithm (Figure 10.1).

The Scientific Method:

1. Formulate a hypothesis that explains an observed behavior.
2. Design an experiment to test your hypothesis.
3. Conduct the experiment.
4. Analyze the results of the experiment — if the results do not support the hypothesis, revise and repeat.

![Figure 10.1](image)

Using this definition, we can classify many activities performed by computer scientists as “science.” The design and analysis of algorithms for solving problems involves a rigorous approach that is based on the scientific method. In particular, verifying the behavior of an algorithm requires forming hypotheses, testing those hypotheses (either through experimentation or mathematical analysis), and then revising the algorithm in response to the results. For example, to design an effective algorithm for accessing records in a large database, you would need to form hypotheses regarding the database’s behavior under various conditions. Then, you would test those hypotheses, either by implementing the algorithm as a computer program and executing it on actual data or by performing rigorous mathematical analysis. Finally, if the result failed to validate your hypotheses, you would modify the algorithm and retest. The development of complex hardware and software systems adheres to this same rigorous, experimental approach—when applied to projects involving millions of transistors or thousands of lines of code, the scientific method is simply implemented on a much larger scale.
Artificial Science

What sets computer science apart from the natural sciences is the type of items being studied. All natural sciences, as well as many social sciences, are concerned with examining complex, naturally occurring phenomena. These sciences work within the confines of innate laws that define the way matter behaves, the way chemicals react, the way life evolves, and even the way people interact. Experiments in physics, chemistry, biology, and psychology strive to understand natural occurrences and extract the underlying laws that control behavior. By contrast, the systems that computer scientists investigate are largely artificial. Programs, computers, and computational models are designed and constructed by people. When a computer scientist analyzes the behavior of a program executing on a particular computer, the phenomena being studied are defined within the artificial world of that machine. Likewise, by devising new computers or other computational models, computer scientists can create their own artificial laws to govern how systems behave. This distinction was effectively summarized by artificial intelligence pioneer Herbert Simon (1916–2001), who coined the phrase “artificial science” to distinguish computer science from the “natural sciences.”

European institutions often bypass the question of whether computer science is a science by calling the field “Informatics” or “Information Systems.” However, these names are limiting in that they denote an emphasis on information processing, rather than on the machinery used to process that information. Donald Knuth (1938–), a noted computer scientist and author of The Art of Computer Programming, has proposed the alternative name “Algorithmics,” which highlights the central role of algorithms in computation. The name “Algorithmics” also suggests a mathematical approach to computing, reflecting the fact that many early researchers in what would eventually be called computer science were mathematicians.

It should be noted that computer science is linked to several other disciplines, each of which approaches computing from a slightly different perspective. For instance, computer engineering is an offshoot of electrical engineering focused on applying scientific theory and engineering principles to the development of new computing technology. The fields of information technology and information systems management approach computing from a business perspective, concentrating on the effective use of information and computer technology in supporting government and commercial organizations. Although the historical backgrounds of these fields are very different, the divisions between the disciplines are not always distinct. A computer scientist conducting research in circuit design might use methods similar to those of a computer engineer, and an information technology manager organizing a network might use methods similar to those of a computer scientist.

Computer Science Themes

Since computation encompasses so many different types of activities, the research conducted by computer scientists is often difficult to classify. However, there are three recurring themes that define the discipline: hardware, software, and theory (Figure 10.2). Some computer scientists focus on hardware,
designing and building computers and related infrastructure. Others concern themselves with the process of composing software and encoding algorithms that can be executed on computers. Still others attack computation from a theoretical standpoint, striving to understand the foundations of the discipline. Although it is possible to find computer scientists whose expertise is exclusive to one of these areas, most activities involve a combination of all three.

Hardware
As we saw in Chapter 1, the term hardware refers to the physical components of a computer and its supporting devices. The design of most modern computers is based on the von Neumann architecture, which consists of a central processing unit (CPU), memory, and input/output devices (see Chapter 6 for additional information regarding the von Neumann architecture). In combination, these three components form a general-purpose machine that can store and execute programs for accomplishing various tasks.

Although most computers are built according to this basic architecture, ongoing research-and-development projects aim to improve the design and organization of hardware. For example, computer scientists involved in circuit design and microchip manufacturing must draw on expertise from chemistry, particle physics, and mechanical engineering to squeeze millions of transistors onto dime-sized microchips. Systems architects, on the other hand, study different ways of connecting components to increase computational throughput (i.e., the amount of work that can be completed in a given time). One subfield of systems architecture is parallel processing, in which a computer’s computational burden is split across multiple CPUs. As the Internet has grown in popularity, many hardware specialists have turned their attention to networking, creating new ways for separate computers to share information and work together.

Software
Whereas hardware consists of machinery for performing computations, software refers to the programs that execute on computers. Most people in the computer industry work on the software side, holding positions such as programmer, systems analyst, or software engineer. In fact, despite the recent economic recession, the software industry remains one of the fastest growing facets of the U.S. economy.

Software can be divided into three basic categories according to each application’s purpose and intended audience. Systems software includes programs that directly control the execution of hardware components. For example, an operating system includes systems software for managing files and directories, linking programs with hardware devices such as printers and scanners, and processing user inputs such as keystrokes and mouse clicks. The term development software refers to programs that are used as tools in the development of other programs. Microsoft .NET and Metrowerks CodeWarrior are examples of development environments for creating, testing, and executing programs using a variety of languages (such as C++ and Java). The applications software category encompasses all other programs, which perform a wide variety of complex tasks. Examples include Web browsers such as Internet Explorer and Netscape Navigator, word processors such as Word and WordPerfect, presentation tools such as PowerPoint and FrameMaker, editors such as NotePad and emacs, and games such as Solitaire and Doom.

Many diverse jobs in computer science are related to the design, development, testing, and maintenance of software. Some software experts’ work is more theoretical, focusing on the formulation and analysis of algorithms. While examining general approaches to problems, these computer scientists strive to understand the complexity of those problems and their solutions. Such “software people” may never write a program in their lives! Another subcategory of the software industry involves developing and extending programming languages so that problems can be solved more easily and efficiently. But perhaps the most common type of software expert is the programmer, who designs algorithms and encodes them into a particular programming language for execution on a computer.
Theory

Just as certain computer scientists study algorithms but never write a program, other experts may never even touch a computer. This is because computer science is concerned with the study of computational methods, in addition to the more practical fields of hardware and software. Theoretical computer scientists, whose work is closely related to mathematics and formal logic, examine different models of computation and strive to understand the capabilities of algorithms and computers. It may surprise you to learn that some of computer science’s key theories were formulated years before the development of electronic computers. For example, in 1930, Alan Turing (1912–1954) (Figure 10.3) designed an abstract computational machine now known as a Turing machine.

![Figure 10.3](image)

Alan Turing.

A Turing machine is a very simple processing unit that can read and write data on a potentially infinite tape. The tape, which we can think of as being divided into cells, serves as the Turing machine’s memory. The processor has the ability to read the data at a particular cell on the tape, write new data to that cell, move in either direction, and distinguish between a finite number of states (distinct situations that alter the behavior of the machine). A Turing machine is programmable in that you can specify a finite set of rules describing the processor’s behavior. However, these rules are limited by the machine’s capabilities (reading, writing, and moving right or left) and are specific to the data stored on the tape. For instance, the example Turing machine in Figure 10.4 is programmed to recognize an even number of “a”s written on the tape. If the tape contains an even number of “a”s, the machine writes the value “Y” to the final cell, but if the tape contains an odd number of “a”s, the machine writes “N” to that cell.

1. Initially, the processor is positioned at the left end of the sequence in state 0.
2. Following the processor instructions, the processor moves right, alternating between state 0 (on odd-numbered cells) and state 1 (on even-numbered cells).
3. If the processor reaches the end of the sequence (marked by a space) in state 0, then there were an even number of a’s — write “Y” in the cell and HALT.
4. If the processor reaches the end of the sequence (marked by a space) in state 1, then there were an odd number of a’s — write “N” in the cell and HALT.

```plaintext
a a a a a
processor state = 0
```

**Figure 10.4** Turing machine to recognize an even number of a’s on the tape.
Although Turing machines might seem simplistic, analysis has proven them to be as powerful as today’s computers. That is, any computation that can be programmed and performed using a modern computer can also be programmed and performed using a Turing machine. Of course, the equivalent Turing machine program might be much more complex and involve a much larger number of steps; however, the Turing machine would eventually complete the task. The advantage of a simple model such as the Turing machine is that it provides a manageable tool for studying computation. In fact, Turing was able to use the Turing machine model of computation to prove a rather astounding result: there exist problems whose solutions cannot be computed. In other words, he was able to verify that certain problems are not solvable using algorithms or any computing device. The most famous example is referred to as the Halting Problem, which Turing demonstrated to be noncomputable in 1930. In programming terminology, the Halting Problem states that you can’t write a program that determines in all cases whether another program will terminate. As you will see in Chapter 13, conditional repetition refers to code that executes repeatedly as long as some condition holds. For example, the JavaScript code below will prompt the user for her age, then will keep prompting until she enters a positive number.

```javascript
age = prompt("Please enter your age", "");
age = parseFloat(age);

while (age < 0) {
age = prompt("You can't have a negative age! Try again:", "");
age = parseFloat(age);
}
```

The ability to include conditional repetition in programs is a powerful algorithmic tool, and most real-world algorithms involve some form of repetition. However, when you introduce conditional repetition into a program, it is possible that the condition will hold indefinitely, resulting in a program that never terminates. This phenomenon is called an infinite loop. Using the Turing machine as his model of computation, Turing was able to prove the impossibility of writing a program (algorithm) that is guaranteed to recognize every program (algorithm) containing an infinite loop. In general, theoretical computer science attempts to understand computation at a level that is independent of any particular machine architecture. This enables the discipline to provide a strong foundation for developing new algorithms and computers.

**Subfields of Computer Science**

In his classic description of the field, *Computer Science: The Discipline*, Peter J. Denning identifies 12 major subfields of computer science (Figure 10.5). Each subfield constitutes a unique viewpoint and approach to computation, many of which have close ties to other disciplines, such as physics, psychology, and biology. However, the common themes of computer science—hardware, software, and theory—influence every subfield.

The following sections examine computer science’s five most visible subfields: algorithms and data structures, architecture, operating systems, and networks, software engineering, and artificial intelligence and robotics. Although it is impossible to include a complete description of these subfields in such a short space, these sections summarize the key ideas that define the disciplines and provide a representative example from each.

**Algorithms and Data Structures**

The subfield of algorithms and data structures involves developing, analyzing, and implementing algorithms for solving problems. Since algorithms are fundamental to all computer science (see Chapter 8), researchers in this subfield can approach the topic from various perspectives. A theoretical computer scientist might be concerned with analyzing the characteristics, efficiency, and limitations of various algorithms. This type of work is relevant, because examining which types of algorithms are best suited to certain tasks can lead to more effective problem solving. Since programs are simply implementations of