1

Structure and Organization of Computing*

Peter J. Denning
Naval Postgraduate School in Monterey

Computing is integral to science—not just as a tool for analyzing data but also as an agent of thought and discovery.

It has not always been this way. Computing is a relatively young discipline. It started as an academic field of study in the 1930s with a cluster of remarkable papers by Kurt Gödel, Alonzo Church, Emil Post, and Alan Turing. The papers laid the mathematical foundations that would answer the question, "what is computation?" and discussed schemes for its implementation. These men saw the importance of automatic computation and sought its precise mathematical foundation. The various schemes they each proposed for implementing computation were quickly found to be equivalent, as a computation in any one could be realized in any other. It is all the more remarkable that their models all led to the same conclusion that certain functions of practical interest—such as whether a computational algorithm (a method of evaluating a function) will ever come to completion instead of being stuck in an infinite loop—cannot be answered computationally.

In the time that these men wrote, the terms "computation" and "computers" were already in common use but with different connotations from today. Computation was taken to be the mechanical steps followed to evaluate mathematical functions. Computers were people who did computations. In recognition of the social changes they were ushering in, the designers of the first digital computer projects

* An earlier version of this chapter, without the section on technology view of the field, was published in *American Scientist 98* (September–October 2010), pp. 198–202. It was reprinted in *Best Writings on Mathematics 2011* (M. Pitici, ed.), Princeton University Press (2011).

Copyright is held by the author.

all named their systems with acronyms ending in "-AC," meaning automatic computer or something similar—resulting in names such as ENIAC, UNIVAC, and EDSAC.

At the start of World War II, the militaries of the United States and the United Kingdom became interested in applying computation to the calculation of ballistic and navigation tables and the cracking of ciphers. They commissioned projects to design and build electronic digital computers. Only one of the projects completed before the war was over. That was the top-secret project at Bletchley Park in England, which cracked the German Enigma cipher using methods designed by Alan Turing.

Many people involved in those projects went on to start computer companies in the early 1950s. The universities began offering programs of study in the new field in the late 1950s. The field and the industry have grown steadily into a modern behemoth whose Internet data centers are said to consume almost 3% of the world's electricity.

During its youth, computing was an enigma to the established fields of science and engineering. At first, it looked like only the technology applications of math, electrical engineering, or science, depending on the observer. However, over the years, computing seemed to provide an unending stream of new insights, and it defied many early predictions by resisting absorption back into the fields of its roots. By 1980, computing had mastered algorithms, data structures, numerical methods, programming languages, operating systems, networks, databases, graphics, artificial intelligence, and software engineering. Its great technology achievements—the chip, the personal computer, and the Internet—brought it into many lives. These advances stimulated more new subfields, including network science, web science, mobile computing, enterprise computing, cooperative work, cyberspace protection, user-interface design, and information visualization. The resulting commercial applications have spawned new research challenges in social networks, endlessly evolving computation, music, video, digital photography, vision, massive multiplayer online games, user-generated content, and much more.

The name of the field changed several times to keep up with the flux. In the 1940s, it was called *automatic computation*, and in the 1950s, *information processing*. In the 1960s, as it moved into academia, it acquired the name *computer science* in the United States and *informatics* in Europe. By the 1980s, the computing field comprised a complex of related fields including computer science, informatics, computational science, computer engineering, software engineering, information systems, and information technology. By 1990, the term *computing* became the standard for referring to this core group.

1.1 Computing Paradigm

Traditional scientists frequently questioned the name *computer science*. They could easily see an engineering paradigm (design and implementation of systems) and a mathematics paradigm (proofs of theorems) but they could not see much of a science paradigm (experimental verification of hypotheses). Moreover, they understood science as a way of dealing with the natural world, and computers looked suspiciously artificial.

The word "paradigm" for our purposes means a belief system and its associated practices, defining how a field sees the world and approaches the solutions of problems. This is the sense that Thomas Kuhn used in his famous book, *The Structure of Scientific Revolutions* (1962). Paradigms can contain subparadigms: thus, engineering divides into electrical, mechanical, chemical, civil, etc., and science divides into physical, life, and social sciences, which further divide into separate fields of science. Table 1.1 outlines the three paradigms that combined to make the early computing field.

The founders of the field came from all three paradigms. Some thought computing was a branch of applied mathematics, some a branch of electrical engineering, and some a branch of computational-oriented science. During its first four decades, the field focused primarily on engineering: The challenges of building reliable computers, networks, and complex software were daunting and occupied almost everyone's attention. By the 1980s, these challenges largely had been met and computing was

Downloaded by [Peter Denning] at 10:45 13 October 2014

TABLE 1.1 Subparadigms Embedded in Computing

	Math	Science	Engineering
1. Initiation	Characterize objects of study (definition).	Observe a possible recurrence or pattern of phenomena (hypothesis).	Create statements about desired system actions and responses (requirements).
2. Conceptualization	Hypothesize possible relationships among objects (theorem).	Construct a model that explains the observation and enables predictions (model).	Create formal statements of system functions and interactions (specifications).
3. Realization	Deduce which relationships are true (proof).	Perform experiments and collect data (validate).	Design and implement prototypes (design).
4. Evaluation	Interpret results.	Interpret results.	Test the prototypes.
5. Action	Act on results (apply).	Act on results (predict).	Act on results (build).

Downloaded by [Peter Denning] at 10:45 13 October 2014

spreading rapidly into all fields, with the help of networks, supercomputers, and personal computers. During the 1980s, computers had become powerful enough that science visionaries could see how to use them to tackle the hardest, "grand challenge" problems in science and engineering. The resulting "computational science" movement involved scientists from all countries and culminated in the US Congress's adopting the High Performance Computing and Communications (HPCC) act of 1991 to support research on a host of large computational problems.

Today, there is agreement that computing *exemplifies* science and engineering and that neither science nor engineering *characterizes* computing. Then what does? What is computing's paradigm?

The leaders of the field struggled with the paradigm question ever since the beginning. Along the way, there were three waves of attempts to unify views. Newell et al. (1967) led the first one. They argued that computing was unique among all the sciences in its study of information processes. Simon (1996), a Nobel laureate in Economics, went so far as to call computing a science of the artificial. Amarel (1971) endorsed this basic idea and added an emphasis on interactions with other fields. A catchphrase of this wave was that "computing is the study of phenomena surrounding computers."

The second wave focused on programming, the art of designing algorithms that produced information processes. In the early 1970s, computing pioneers Edsger Dijkstra and Donald Knuth took strong stands favoring algorithm analysis as the unifying theme. A catchphrase of this wave was "computer science equals programming." In recent times, this view has foundered because the field has expanded well beyond programming, whereas public understanding of a programmer has narrowed to just those who write code.

The third wave came as a result of the NSF-funded Computer Science and Engineering Research Study (COSERS), led by Bruce Arden in the late 1970s. Its catchphrase was "computing is the automation of information processes." Although its final report successfully exposed the science of computing and explained many esoteric aspects to the layperson, its central view did not catch on.

An important aspect of all three definitions was the positioning of the computer as the object of attention. The computational science movement of the 1980s began to step away from that notion, adopting the view that computing is not only a tool for science but also a new method of thought and discovery in science. The process of dissociating from the computer as the focal center came to completion in the late 1990s when leaders of the field of biology—epitomized by Nobel laureate David Baltimore (2001) and echoing cognitive scientist Douglas Hofstadter (1985)—said that biology had become an information science and DNA translation is a natural information process. Many computer scientists have joined biologists in research to understand the nature of DNA information processes and to discover what algorithms might govern them.

Take a moment to savor this distinction that biology makes. First, some information processes are natural. Second, we do not know whether all natural information processes are produced by algorithms.

The second statement challenges the traditional view that algorithms (and programming) are at the heart of computing. Information processes may be more fundamental than algorithms.

Scientists in other fields have come to similar conclusions. They include physicists working with quantum computation and quantum cryptography, chemists working with materials, economists working with economic systems, cognitive scientists working with brain processes, and social scientists working with networks. All have said that they discovered information processes in their disciplines' deep structures. Stephen Wolfram (2002), a physicist and creator of the software program *Mathematica*, went further, arguing that information processes underlie every natural process in the universe.

All this leads us to the modern catchphrase: "Computing is the study of information processes, natural and artificial." The computer is a tool in these studies but is not the object of study. Dijkstra once said: "Computing is no more about computers than astronomy is about telescopes."

The term *computational thinking* has become popular to refer to the mode of thought that accompanies design and discovery done with computation (Wing 2006). This term was originally called *algorithmic thinking* in the Newell et al. (1960) and was widely used in the 1980s as part of the rationale for computational science. To think computationally is to interpret a problem as an information process and then seek to discover an algorithmic solution. It is a very powerful paradigm that has led to several Nobel Prizes.

All this suggests that computing has developed a paradigm all its own (Denning and Freeman 2009). Computing is no longer just about algorithms, data structures, numerical methods, programming languages, operating systems, networks, databases, graphics, artificial intelligence, and software engineering, as it was prior to 1990. It now also includes exciting new subjects including Internet, web science, mobile computing, cyberspace protection, user-interface design, and information visualization. The resulting commercial applications have spawned new research challenges in social networking, endlessly evolving computation, music, video, digital photography, vision, massive multiplayer online games, user-generated content, and much more.

The computing paradigm places a strong emphasis on the scientific (experimental) method to understand computations. Heuristic algorithms, distributed data, fused data, digital forensics, distributed networks, social networks, and automated robotic systems, to name a few, are often too complex for mathematical analysis but yield to the scientific method. These scientific approaches reveal that *discovery* is as important as *construction* or *design*. Discovery and design are closely linked: the behavior of many large designed systems (such as the web) is discovered by observation; we design simulations to imitate discovered information processes. Moreover, computing has developed search tools that are helping make scientific discoveries in many fields.

The central focus of the computing paradigm can be summarized as information processes—natural or constructed processes that transform information. They can be discrete or continuous.

Table 1.2 summarizes the computing paradigm with this focus. While it contains echoes of engineering, science, and mathematics, it is distinctively different because of its central focus on information processes (Denning and Freeman 2009). It allows engineering and science to be present together without having to choose.

There is an interesting distinction between computational expressions and the normal language of engineering, science, and mathematics. Engineers, scientists, and mathematicians endeavor to position themselves as outside observers of the objects or systems they build or study. Outside observers are purely representational. Thus, traditional blueprints, scientific models, and mathematical models are not executable. (However, when combined with computational systems, they give automatic fabricators, simulators of models, and mathematical software libraries.) Computational expressions are not constrained to be outside the systems they represent. The possibility of self-reference makes for very powerful computational schemes based on recursive designs and executions and also for very powerful limitations on computing, such as the noncomputability of halting problems. Self-reference is common in natural information processes; the cell, for example, contains its own blueprint.

Downloaded by [Peter Denning] at 10:45 13 October 2014

TABLE 1.2 The Computing Paradigm

	Computing
1. Initiation	Determine if the system to be built (or observed) can be represented by information processes, either finite (terminating) or infinite (continuing interactive).
2. Conceptualization	Design (or discover) a computational model (e.g., an algorithm or a set of computational agents) that generates the system's behaviors.
3. Realization	Implement designed processes in a medium capable of executing its instructions. Design simulations and models of discovered processes. Observe behaviors of information processes.
4. Evaluation	Test the implementation for logical correctness, consistency with hypotheses, performance constraints, and meeting original goals. Evolve the realization as needed.
5. Action	Put the results to action in the world. Monitor for continued evaluation.

Downloaded by [Peter Denning] at 10:45 13 October 2014

1.2 Two Views of Computing

Part of a scientific paradigm is a description of the knowledge of the field, often referred to as the "body of knowledge." Within the computing paradigm, two descriptions of the computing body of knowledge have grown up. They might be called a technology interpretation and a principles interpretation.

Before 1990, most computing scientists would have given a technological interpretation, describing the field in terms of its component technologies. After 1990, the increasingly important science aspect began to emphasize the fundamental principles that empower and constrain the technologies.

In reality, these two interpretations are complementary. They both see the same body of knowledge, but in different ways. The technological view reflects the way the field has evolved around categories of technology; many of these categories reflect technical specialties and career paths. The science view reflects a deeper look at timeless principles and an experimental outlook on modeling and validation in computing.

These two views are discussed in Sections 1.3 and 1.4.

1.3 View 1: Technologies of Computing

Over the years, the ACM and Institute of Electrical and Electronics Engineers Computer Society (IEEECS) collaborated on a computing body of knowledge and curriculum recommendations for computer science departments. The milestones of this process give a nice picture of the technological development of the field.

1.3.1 First Milestone: Curriculum 68

In the mid-1960s, the ACM (with help from people in IEEECS) undertook the task to define curriculum recommendations for schools that wished to offer degrees in the new field of computer science (ACM 1968). Their report said that the field consisted of three main parts:

Information structures and processes
Information processing systems
Methodologies

The methodologies included design approaches for software and applications. The core material was mostly the mathematical underpinnings for the parts listed earlier:

Algorithms
Programming
Data structures
Discrete math
Logic circuits
Sequential machines
Parsing
Numerical methods

Many computer science departments adopted these recommendations.

1.3.2 Second Milestone: Computing as a Discipline

The ACM and IEEECS formally joined forces in 1987 to defend computing curricula from a bastardized view that "CS = programming." Around Donald Knuth and Edsger Dijkstra (1970) started making strong and eloquent cases for formal methods of software design, analysis, and construction. They said "we are all programmers" trying to employ powerful intellectual tools to tame complexity and enable correct and dependable software. Although computer scientists understood a programmer as highly skilled expert at these things, the public view of programmers was narrowing to low-level coders, who occasionally caused trouble by hacking into other people's systems.

The committee laid out a model of the computing field that emphasized its breadth, showing that it is much richer than simply programming (Denning et al. 1989). Table 1.3 depicts the 9 × 3 matrix model of the computing field offered by the committee. Theory, abstraction, and design were used in the report for the mathematics, science, and engineering paradigms, respectively. The report gave details about what ideas and technologies fit into each of the 27 boxes in the matrix. It became the basis for a major ACM/IEEE curriculum revision in 1991.

Although this effort had a strong internal influence on the curriculum, it had little external influence on the perception that "CS=programming." In fact, that perception was alive and well in the early 2000s when enrollments declined by over 50%.

1.3.3 Third Milestone: Information Technology Profession

In 1998, the ACM launched an "IT profession" initiative, based on a widely held perception that the field had evolved from a discipline to a profession (Denning 1998, Denning 2001, Holmes 2000). The initiative responded to three trends: the growing interest in the industry for professional standards (especially in safety-critical systems), organized professional bodies representing various specialties,

TABLE 1.3 Matrix Model of Computing Discipline, 1989

Topic Area	Theory	Abstraction	Design
1. Algorithms and data structures			
2. Programming languages			
3. Architecture			
4. Operating systems and networks			
5. Software engineering			
6. Databases and information retrieval			
7. Artificial intelligence and robotics			
8. Graphics			
9. Human–computer interaction			

Downloaded by [Peter Denning] at 10:45 13 October 2014

TABLE 1.4 The Profession of Information Technology

IT-Core Disciplines	IT-Intensive Disciplines	IT-Supportive Occupations
Artificial intelligence	Aerospace engineering	Computer technician
Computer science	Bioinformatics	Help desk technician
Computer engineering	Cognitive science	Network engineer
Computational science	Cryptography	Professional IT trainer
Database engineering	Digital library science	Security specialist
Graphics	E-commerce	System administrator
Human–computer interaction	Economics	Web services designer
Network engineering	Genetic engineering	Web identity designer
Operating systems	Information science	Database administrator
Performance engineering	Information systems	
Robotics	Public policy and privacy	
Scientific computing	Quantum computing	
Software architecture	Instructional design	
Software engineering	Knowledge engineering	
System security	Management information systems	
	Material science	
	Multimedia design	
	Telecommunications	

Downloaded by [Peter Denning] at 10:45 13 October 2014

and a university movement to establish degree programs in information technology. The ACM leadership concluded that the computing field met the basic criteria for a profession and that it was time for ACM to configure itself accordingly.

Table 1.4 is an inventory ACM made of the organized groups in the field. They saw IT professionals as a much larger and more diverse group than computer scientists and engineers, with at least 42 organized affinity groups in three categories. The first category comprises the major technical areas of IT and spans the intellectual core of the field. The second category comprises other well-established fields that are intensive users of IT; they draw heavily on IT and often make novel contributions to computing. The third category comprises areas of skill and practice necessary to keep and support the IT infrastructures that everyone uses. Allen Tucker and Peter Wegner (1996) also noted the dramatic growth and professionalization of the field and its growing influence on many other fields.

Unfortunately, the talk about "profession" led to a new round of terminological confusion. A profession is a social structure that includes many disciplines, but it is not a discipline in its own right. IT is not a field of research; the core disciplines (left column) and partner disciplines (middle column) attend to the research. To what does the term "computing field" refer in this context?

A decade later, it was clear that this interpretation of the field did not match what had actually evolved (Denning and Freeman 2009). The popular label IT did not reconcile the three parts of the computing field under a single umbrella unique to computing. IT now connotes technological infrastructure and its financial and commercial applications, but not the core technical aspects of computing.

1.3.4 Fourth Milestone: Computing Curriculum 2001

The ACM and IEEECS Education Boards were more cautious than ACM leadership in embracing an IT profession when they undertook a curriculum review and revision in 1999. They focused on the core specialties (first column in Table 1.2) and identified the computing discipline with these six academic specialties:

EE—Electrical engineering
CE—Computer engineering
CS—Computer science

SWE—Software engineering
IS—Information systems
IT—Information technology

It was understood that students interested in hardware would enroll in an EE or CE program; students interested in software in a CE, CS, or SWE program; and students interested in organizational and enterprise aspects would enroll in IS or IT programs. Here, the term "IT" is far from what the IT profession initiative envisioned—it refers simply to a set of degree programs that focus on organizational applications of computing technology.

The CC2001 committee organized the body of knowledge into 14 main categories, as follows:

Algorithms and complexity
Architecture and organization
Computational science
Discrete structures
Graphics and visual computing
Human–computer interaction
Information management
Intelligent systems
Net-centric computing
Operating systems
Programming fundamentals
Programming languages
Social and professional issues
Software engineering

There were a total of 130 subcategories. The body of knowledge had 50% more categories than a decade before!

1.3.5 Fifth Milestone: Computing Curriculum 2013

The ACM and IEEECS again collaborated on a ten-year review of the computing curriculum. They learned that the field had grown from 14 to 18 knowledge areas since the 2001 review:

Algorithms and complexity
Architecture and organization
Computational science
Discrete structure
Graphics and visual computing
Human–computer interaction
Information assurance and security
Information management
Intelligent systems
Networking and communications
Operating systems
Platform-based development
Parallel and distributed computing
Programming languages
Software development fundamentals
Software engineering
Systems fundamentals
Social and professional issues

Downloaded by [Peter Denning] at 10:45 13 October 2014

The committee was concerned about the pressure to increase the size of the computer science core. They calculated that the 2001 curriculum recommended 280 core hours and an update in 2008 increased that to 290. The core hours to cover the list earlier would be 305. They divided the core into two parts. Tier 1, the "must have" knowledge, and Tier 2, the "good to have" knowledge. They recommended that individual departments choose at least 80% of the Tier 2 courses, for a total of 276 hours, leaving plenty of time for electives in a student's specialization area.

When ACM issued Curriculum 68, most of us believed that every computer scientist should know the entire core. Today, that is very difficult, even for seasoned computer scientists, since the field has grown so much since 1968.

1.4 View 2: Great Principles of Computing

The idea of organizing the computing body of knowledge around the field's fundamental principles is not new. Many of the field's pioneers were deeply concerned about why computing seemed like a new field, not a subset of other fields like mathematics, engineering, or science. They spent considerable effort to explain what they were doing in terms of the fundamental principles they worked with. Prominent examples are Turing's paper (1937); the essays of Newell et al. (1967); Simon's book (1996); and Arden's COSERS report (1971, 1983). In subsets of the field, thinkers ferreted out the fundamental principles. Examples are Coffman and Denning (1973) on operating systems, Kleinrock (1975) on queueing systems, Hillis (1999) on the nature of computing machines, and Harel (2003) on algorithms and limits of computing.

This viewpoint, however, stayed in the background. I think the reason was simply that for many years, we were concerned with the engineering problems of constructing computers and networks that worked reliably. Most computer scientists were occupied solving engineering problems. The ones most interested in fundamental principles were the ones interested in theory. By 1990, we had succeeded beyond our wildest dreams with the engineering. However, our descriptions of the field looked like combinations of engineering and mathematics. Many outsiders wondered what the word "science" was doing in our title.

When the computational science movement began in the 1980s, many computer scientists felt like they were being excluded. Computational scientists, for their part, did not realize that computer scientists were interested in science. A growing number of us became interested in articulating the science side of computing. It was not easy, because many scientists agreed with Herb Simon (1996), that we are at best a science of the artificial, but not a real science. Real sciences, in their opinions, dealt with naturally occurring processes.

But by 1990, prominent scientists were claiming to have discovered natural information processes, such as in biology, quantum physics, economics, and chemistry. This gave new momentum to our efforts to articulate a science-oriented view of computing (Denning 2005, Denning 2007).

Inspired by the great principles work of James Trefil and Robert Hazen (1996) for science, my colleagues and I have developed the Great Principles of Computing framework to accomplish this goal (Denning 2003, Denning and Martell 2004). Computing principles fall into seven categories: computation, communication, coordination, recollection, automation, evaluation, and design (Table 1.5).

Each category is a perspective on computing: a window into the computing knowledge space. The categories are not mutually exclusive. For example, the Internet can be seen as a communication system, a coordination system, or a storage system. We have found that most computing technologies use principles from all seven categories. Each category has its own weight in the mixture, but they are all there.

In addition to the principles, which are relatively static, we need to take account of the dynamics of interactions between computing and other fields. Scientific phenomena can affect each other in one of two ways: implementation and influence. A combination of existing things implements a phenomenon by generating its behaviors. Thus, digital hardware physically implements computation, artificial intelligence implements aspects of human thought, a compiler implements a high-level language with machine code, hydrogen and oxygen implement water, and complex combinations of amino acids implement life.

Downloaded by [Peter Denning] at 10:45 13 October 2014

TABLE 1.5 Great Principles of Computing

Category	Focus	Examples
Computation	What can and cannot be computed	Classifying complexity of problems in terms of the number of computational steps to achieve a solution. Is P=NP? Quantum computation.
Communication	Reliably moving information between locations	Information measured as entropy. Compression of files, error-correcting codes, cryptography.
Coordination	Achieving unity of operation from many autonomous computing agents	Protocols that eliminate conditions that cause indeterminate results. Choice uncertainty: cannot choose between two near simultaneous signals within a deadline. Protocols that lead the parties to common beliefs about each other's system.
Recollection	Representing, storing, and retrieving information from media	All storage systems are hierarchical, but no storage system can offer equal access time to all objects. Locality principle: all computations favor subsets of their data objects in any time interval. Because of locality, no storage system can offer equal access time to all objects.
Automation	Discovering algorithms for information processes	Most heuristic algorithms can be formulated as searches over enormous data spaces. Human memory and inference are statistical phenomena described by Bayes Rule. Many human cognitive processes can be modeled as information processes.
Evaluation	Predicting performance of complex systems	Most computational systems can be modeled as networks of servers whose fast solutions yield close approximations of real throughput and response time.
Design	Structuring software systems for reliability and dependability	Complex systems can be decomposed into interacting modules and virtual machines following the principles of information hiding and least privilege. Modules can be stratified by layers corresponding to time scales of events that manipulate objects.

Downloaded by [Peter Denning] at 10:45 13 October 2014

Influence occurs when two phenomena interact with each other. Atoms arise from the interactions among the forces generated by protons, neutrons, and electrons. Galaxies interact via gravitational waves. Humans interact with speech, touch, and computers. Interactions exist across domains as well as within domains. For example, computation influences physical action (electronic controls), life processes (DNA translation), and social processes (games with outputs). Table 1.6 illustrates interactions between computing and each of the physical, life, and social sciences as well as within computing itself. There can be no question about the pervasiveness of computing in all fields of science.

1.5 Relation between the Views

The technology and the principles views discussed earlier are two different interpretations of the same knowledge space. They are alternatives for expressing the computing body of knowledge.

The same principle may appear in several technologies, and a particular technology likely relies on several principles. The set of active principles (those used in at least one technology) evolves much more slowly than the technologies.

TABLE 1.6 Examples of Computing Interacting with Other Domains

	Physical	Social	Life	Computing
Implemented by	Mechanical, optical, electronic, quantum, and chemical computing	Wizard of Oz, mechanical robots, human cognition, games with inputs and outputs	Genomic, neural, immunological, DNA transcription, evolutionary computing	Compilers, OS, emulation, reflection, abstractions, procedures, architectures, languages
Implements	Modeling, simulation, databases, data systems, quantum cryptography	Artificial intelligence, cognitive modeling, autonomic systems	Artificial life, biomimetics, systems biology	
Influenced by	Sensors, scanners, computer vision, optical character recognition, localization	Learning, programming, user modeling, authorization, speech understanding	Eye, gesture, expression, and movement tracking, biosensors	Networking, security, parallel computing, distributed systems, grids
Influences	Locomotion, fabrication, manipulation, open-loop control	Screens, printers, graphics, speech generation, network science	Bioeffectors, haptics, sensory immersion	
Bidirectional influence	Robots, closed-loop control	Human–computer interaction, games	Brain–computer interfaces	

Downloaded by [Peter Denning] at 10:45 13 October 2014

While the two styles of framework are different, they are strongly connected. To see the connection, imagine a 2D matrix. The rows name technologies, and the columns name categories of principles. The interior of the matrix is the knowledge space of the field.

Imagine someone who wants to enumerate all the principles involved with a technology. If the matrix is already filled in, the answer is simply to read the principles from the row of the matrix. Otherwise, fill it in by analyzing the technology for principles in each of the seven categories. In the figure later, we see that the security topic draws principles from all seven categories.

	Computation	Communication	Coordination	Recollection	Automation	Evaluation	Design
Security	O(.) of encryption functions	Secrecy authentication covert channels	Key distr protocol zero knowl proof	Confinement partitioning for MLS reference monitor	Instrusion detection biometric id	Protocol perform under various loads	End-to-end layered functions virtual machines

Within the principles framework, someone can enumerate all the technologies that employ a particular principle. In the example later, we see that the coordination category contributes principles to all the technologies listed.

	Computation	Communication	Coordination	Recollection	Automation	Evaluation	Design	
Architecture			Hardware handshake					
Internet			TCP and IP protocols					
Security			Key distr protocol zero knowl proof					
Virtual memory			Page fault interrupt					
Database			Locking protocol					
Programming language			Semaphores monitors					

1.6 What Are Information Processes?

There is a potential difficulty with defining computation in terms of information. Information seems to have no settled definition. Claude Shannon the father of information theory, in 1948 defined information as the expected number of yes–no questions one must ask to decide which message was sent by a source. This definition describes the inherent information of a source before any code is applied; all codes for the course contain the same information. Shannon purposely skirted the issue of the meaning of bit patterns, which seems to be important to defining information. In sifting through many published definitions, Paolo Rocchi (2010) concluded that definitions of information necessarily involve an objective component, signs and their referents or, in other words, symbols and what they stand for, and a subjective component, meanings. How can we base a scientific definition of information on something with such an essential subjective component?

Biologists have a similar problem with "life." Life scientist Robert Hazen (2007) notes that biologists have no precise definition of life, but they do have a list of seven criteria for when an entity is living. The observable affects of life, such as chemistry, energy, and reproduction, are sufficient to ground the science of biology. In the same way, we can ground a science of information on the observable affects (signs and referents) without a precise definition of meaning.

A representation is a pattern of symbols that stands for something. The association between a representation and what it stands for can be recorded as a link in a table or database or as a memory in people's brains. There are two important aspects of representations: *syntax* and *stuff*. Syntax is the rules for constructing patterns; it allows us to distinguish patterns that stand for something from patterns that do not. Stuff is measurable physical states of the world that hold representations, usually in media or signals. Put these two together and we can build machines that can detect when a valid pattern is present.

A representation that stands for a method of evaluating a function is called an algorithm. A representation that stands for values is called data. When implemented by a machine, an algorithm controls the transformation of an input data representation to an output data representation. The distinction between the algorithm and the data representations is pretty weak; the executable code output by a compiler looks like data to the compiler and algorithm to the person running the code.

Even this simple notion of representation has deep consequences. For example, as Gregory Chaitin (2006) has shown, there is no algorithm for finding the shortest possible representation of something.

Downloaded by [Peter Denning] at 10:45 13 October 2014

Some scientists leave open the question of whether an observed information process is actually controlled by an algorithm. DNA translation can be called an information process; if someone discovers a controlling algorithm, it could be also called a computation.

Some mathematicians define computation separate from implementation. They do this by treating computations as logical orderings of strings in abstract languages and are able to determine the logical limits of computation. However, to answer questions about running time of observable computations, they have to introduce costs representing the time or energy of storing, retrieving, or converting representations. Many real-world problems require exponential-time computations as a consequence of these implementable representations. I still prefer to deal with implementable representations because they are the basis of a scientific approach to computation.

These notions of representation are sufficient to give us the definitions we need for computing. An information process is a sequence of representations. (In the physical world, it is a continuously evolving, changing representation.) A computation is an information process in which the transitions from one element of the sequence to the next are controlled by a representation. (In the continuous world, we would say that each infinitesimal time and space step is controlled by a representation.)

Downloaded by [Peter Denning] at 10:45 13 October 2014

1.7 Where Computing Stands

Computing as a field has come to exemplify good science as well as engineering. The science is essential to the advancement of the field because many systems are so complex that experimental methods are the only way to make discoveries and understand limits. Computing is now seen as a broad field that studies information processes, natural and artificial.

This definition is broad enough to accommodate three issues that have nagged computing scientists for many years: Continuous information processes (such as signals in communication systems or analog computers), interactive processes (such as ongoing web services), and natural processes (such as DNA translation) all seemed like computation but did not fit the traditional algorithmic definitions.

The great principles framework reveals a rich set of rules on which all computation is based. These principles interact with the domains of the physical, life, and social sciences, as well as with computing technology itself.

Computing is not a subset of other sciences. None of those domains is fundamentally concerned with the nature of information processes and their transformations. Yet this knowledge is now essential in all the other domains of science. Computer scientist Paul Rosenbloom (2012) of the University of Southern California argued that computing is a new great domain of science. He is on to something.

References

Amarel, S. Computer science: A conceptual framework for curriculum planning. *ACM Communications* 14(6): 391–401 (June 1971).

Arden, B. W. (ed.) The computer science and engineering research study. *ACM Communications* 19(12): 670–673 (December 1971).

Arden, B. W. (ed.) *What Can Be Automated: Computer Science and Engineering Research Study* (*COSERS*). MIT Press, Cambridge, MA (1983).

Atchison, W. F. et al. (eds.) ACM Curriculum 68. *ACM Communications* 11: 151–197 (March 1968).

Baltimore, D. How biology became an information science. In *The Invisible Future* (P. Denning, ed.), McGraw-Hill, New York (2001), pp. 43–56.

Chaitin, G. *Meta Math! The Quest for Omega*. Vintage, New York (2006).

Coffman, E. G. and P. Denning. *Operating Systems Theory*. Prentice-Hall, Englewood Cliffs, NJ (1973).

Denning, P. Computing the profession. *Educom Review* 33: 26–30, 46–59 (1998). http://net.educause.edu/ir/library/html/erm/erm98/erm9862.html (accessed on October 15, 2013).

Denning, P. Who are we? *ACM Communications* 44(2): 15–19 (February 2001).

Denning, P. Great principles of computing. *ACM Communications* 46(11): 15–20 (November 2003).
Denning, P. Is computer science science? *ACM Communications* 48(4): 27–31 (April 2005).
Denning, P. Computing is a natural science. *ACM Communications* 50(7): 15–18 (July 2007).
Denning, P., D. E. Comer, D. Gries, M. C. Mulder, A. Tucker, A. J. Turner, and P. R. Young. Computing as a discipline. *ACM Communications* 32(1): 9–23 (January 1989).
Denning, P. and P. Freeman. Computing's paradigm. *ACM Communications* 52(12): 28–30 (December 2009).
Denning, P. and C. Martell. Great principles of computing web site (2004). http://greatprinciples.org.
Denning, P. and P. Rosenbloom. The fourth great domain of science. *ACM Communications* 52(9): 27–29 (September 2009).
Forsythe, G., B. A. Galler, J. Hartmanis, A. J. Perlis, and J. F. Traub. Computer science and mathematics. *ACM SIGCSE Bulletin* 2 (1970). http://doi.acm.org/10.1145/873661.873662 (accessed on October 15, 2013).
Harel, D. *Computers, Ltd*. Oxford University Press, Oxford, U.K. (2003).
Hazen, R. *Genesis: The Scientific Quest for Life's Origins*. Joseph Henry Press, Washington, DC (2007).
Hillis, D. *The Pattern on the Stone*. Basic Books, New York (1999).
Hofstadter, D. *Metamagical Themas: Questing for the Essence of Mind and Pattern*. Basic Books, New York (1985). See his essay on "The Genetic Code: Arbitrary?"
Holmes, N. Fashioning a foundation for the computing profession. *IEEE Computer* 33: 97–98 (July 2000).
Kleinrock, L. *Queueing Systems*. John Wiley & Sons, New York (1975).
Newell, A., A. J. Perlis, and H. A. Simon. Computer science. *Science* 157(3795): 1373–1374 (September 1967).
Rocchi, P. *Logic of Analog and Digital Machines*. Nova Publishers, New York (2010).
Rosenbloom, P. S. A new framework for computer science and engineering. *IEEE Computer* 31–36: 23–28 (November 2004).
Rosenbloom, P. S. *On Computing: The Fourth Great Scientific Domain*. MIT Press (2012).
Shannon, C. and W. Weaver. *The Mathematical Theory of Communication*. University of Illinois Press, Champaign, IL (1948). First edition 1949. Shannon's original paper is available on the Web: http://cm.bell-labs.com/cm/ms/what/shannonday/paper.html (accessed on October 15, 2013).
Simon, H. *The Sciences of the Artificial*. MIT Press, Cambridge, MA (1st edn. 1969, 3rd edn. 1996).
Trefil, J. and R. Hazen. *Science: An Integrated Approach*. John Wiley & Sons, Chichester, U.K. (1st edn. 1996, 6th edn. 2009).
Tucker, A. and P. Wegner. Computer science and engineering: The discipline and its impact. Chapter 1, in *Handbook of Computer Science and Engineering*, A. Tucker (ed.), pp. 1–15. CRC Press, Boca Raton, FL (1996).
Turing, A. M. On computable numbers with an application to the Entscheidungsproblem, *Proceedings of the London Mathematical Society*, Series 2, Vol. 42, pp. 230–265, London, U.K. (1937).
Wing, J. Computational thinking. *ACM Communications* 49(3): 33–35 (March 2006).
Wolfram, S. *A New Kind of Science*. Wolfram Media, Champaign, IL (2002).

Downloaded by [Peter Denning] at 10:45 13 October 2014