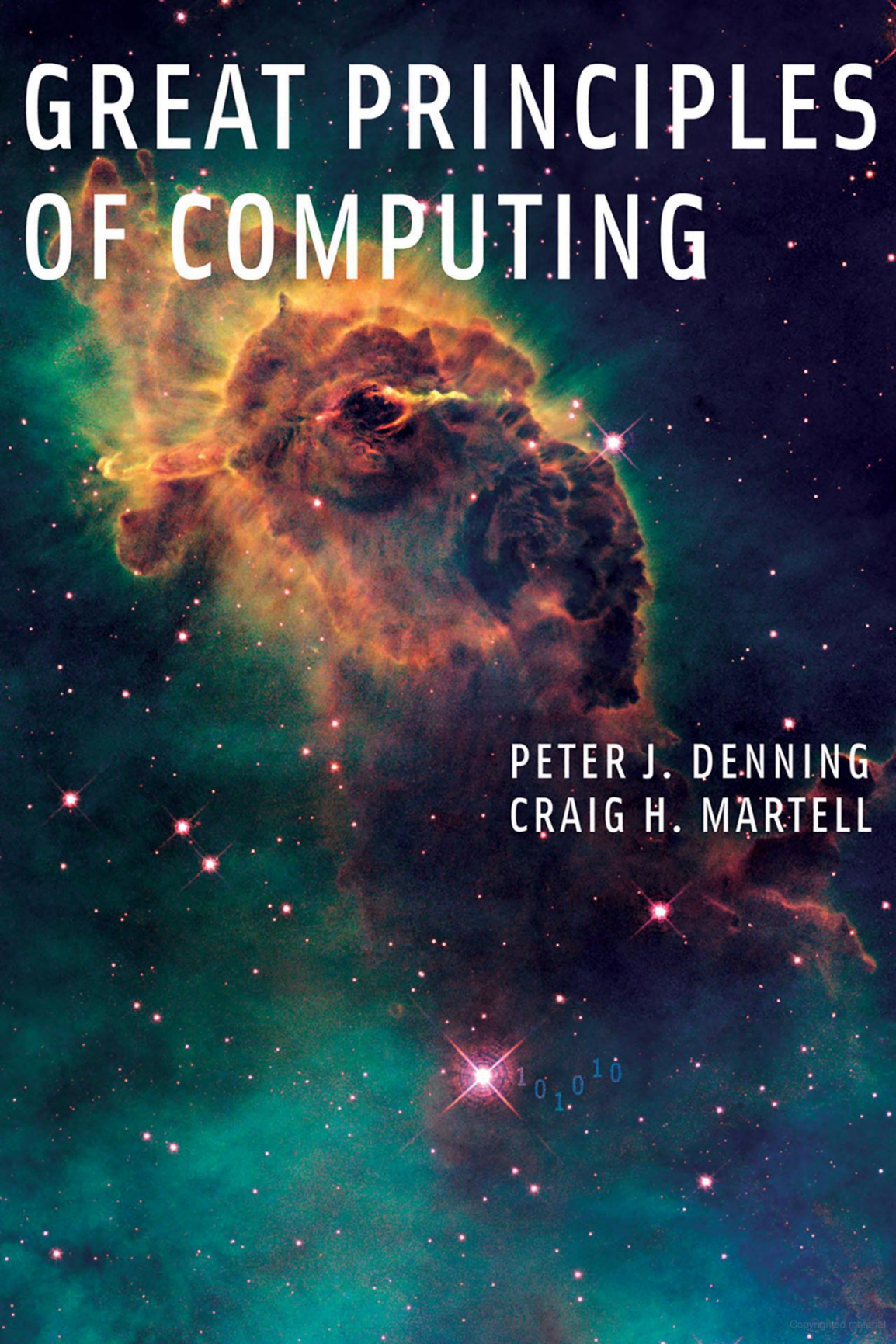
GREAT PRINCIPLES OF COMPUTING



PETER J. DENNING
CRAIG H. MARTELL

Copyrighted material

© 2015 Massachusetts Institute of Technology

All rights reserved. No part of this book may be reproduced in any form by any electronic or mechanical means (including photocopying, recording, or information storage and retrieval) without permission in writing from the publisher.

MIT Press books may be purchased at special quantity discounts for business or sales promotional use. For information, please email special_sales@mitpress.mit.edu.

This book was set in Stone Sans Std and Stone Serif Std by Toppan Best-set Premedia Limited, Hong Kong. Printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Denning, Peter J., 1942–
Great principles of computing / Peter J. Denning and Craig H. Martell.
pages cm
Includes bibliographical references and index.
ISBN 978-0-262-52712-5 (pbk. : alk. paper)
1. Computer science. I. Martell, Craig H., 1965– II. Title.
QA76.D3483 2015
004—dc23

2014022598

10 9 8 7 6 5 4 3 2 1

Copyrighted material

Contents

Copyrighted material

Copyrighted material

Foreword

Peter Denning and Craig Martell have taken on a monumental topic: identifying and elucidating principles that shape and inform the process of coercing computers to do what we *want* them to do and struggling with the difference between what they *actually* do (that is, what we told them) and what we want them to do. Bugs (errors) are examples of the difference. Bugs are usually a result of inadvertently programming the machine to do something we did not intend. But errors are not the only source of bugs. A bug also arises when an unexpected behavior emerges from the execution of a program in a system. Networks of computers with their myriad variations in software and interactions are often the source of emergent behaviors. We sometimes speak of a *network effect* in which a trend becomes a predominant behavior that reinforces the emergence of some feature that might not have been envisioned or even intended. This can happen when an application is put to use in ways that were not anticipated. *Spam* and *phishing* are examples of emergent behaviors with email in large networks.

Such effects challenge our ability to understand, anticipate, and analyze complex behaviors arising in large-scale software systems in large-scale networks. Even if each component operates within its design parameters, the system as a whole can give indeterminate results because of unpredictable interactions among components.

Complex emergent behaviors also arise simply because computing machines are finite. Digitized information always contains small errors of representation. Tiny errors can accumulate into catastrophes over billions of computational steps. A very concrete example of unanticipated results arises from floating point arithmetic with finite precision. Round-off errors and other artifacts of handling very large or very small values can lead to catastrophic results, as William Kahan eloquently demonstrated in his paper on this topic in a symposium on numerical computation in 2005.[1]

Copyrighted material

These effects teach us that the business of getting computers to do things on our behalf is both a nontrivial and deeply intellectual exercise. This book aims to provide insight into some fundamental principles that can orient our approach to computing in its most general sense.

The authors organize their analysis into eleven chapters, each of which plays an important role in the panoply of activities we associate with computing. I think of these as foci for marshaling and organizing resources in aid of computational outcomes. By *computational* I mean to suggest achieving particular objectives through the use of computers and their software. This is intentionally unspecific. Making a computer game work is a computational objective as much as getting a complex, distributed, networked financial exchange system to work. Despite the disparity of computational objectives, designers are aided by definite principles for managing and marshaling resources—information representations, communications, computing elements, programs, memory, modeling, analysis, and so forth. I read that the book's overall intent is to codify the principles that facilitate achievement of these objectives. This effort is broad in its scope and depth.

One of the things that makes computing so interesting is the utter generality of binary representations. We can choose to make the bits mean anything we wish. We can manipulate these bits in myriad ways and choose to interpret the results in equally diverse fashion. Just as we convert algebraic word problems into equations that we manipulate according to straightforward mathematical rules to find answers compatible with the original equations, so also we write programs to manipulate bits following rules that lead to a chosen interpretation of the resulting bits. Large-scale simulations, big data, and complex visual renderings all share the property that they help us to understand and interpret the bits we manipulate.

One of the reasons I have been a strong proponent of teaching programming in middle school and high school (and perhaps even earlier) is the discipline it imposes on organizing thoughts to problem solving. One has to analyze the problem, break it into manageable parts, figure out what has to happen for the program to solve the problem (that is, produce the desired result), then work through the task of writing the program, utilizing preexisting libraries, if applicable, compiling and running the program, and verifying that it produces the desired result and nothing else. The last discipline, which we might call a combination of debugging and verification, is a skill that is applicable to more than programming. Although I am not an advocate of making everyone into a programmer, I think it is valuable for people to learn the skills that are applicable to successful programming because these skills are broadly applicable to many other problems.

Copyrighted material

Programming skills can be put to work dealing with complex system design and analysis. Here I think we reach a very important area that Denning and Martell emphasize in their chapter on design. Good design has many useful properties. I am reminded of the remark that "neatness is its own reward" because you can find things you put away when you need them. Good design is its own reward because it facilitates understanding of complexity and ability to evolve and revise the design to achieve new objectives. In design of the Internet we took a lesson from its predecessor, the ARPANET, which could not scale up in size. We envisioned the functionality of the system in *layers* and standardized the interfaces between the layers. The result was that while keeping these interfaces stable, we were able to allow for enormous flexibility in the implementation and reimplementation of the layers between the interfaces. The Internet Protocol is a good example. Designers of applications knew nothing about how Internet protocol packets were carried—the protocol did not specify. Nor did the protocol itself depend on what information packet payloads carried—the meanings of bits in the payload were opaque. One consequence of this design decision has been that the Internet Protocol has been layered on top of every new communication system designed since the early 1970s. Another consequence is that new applications have been placed in the Internet without changing the networks because the Internet Protocol carried their packets to software at the edges of the Internet. Only the transmitting and receiving hosts needed to know what the payload bits carried in packets meant. The routers that move Internet Protocol packets do not depend on the content of the packet payloads.

The role of design cannot be overemphasized in dealing with computing. Whether it is the hardware, the operating system, the application, the data, file and directory structures, the choice of language(s), it all comes down to thinking about design and how the ensemble, the system, will work. Sometimes one hears the term *system engineering* too infrequently. I am a systems person and take some pride in thinking along architectural lines. How do all the pieces go together? What should be a *piece?* How does the design facilitate adaptation to new requirements? Is the design maintainable? How hard is it to teach someone else how the design works?

An interesting test of a good design is to see whether someone who is confronted with the system de novo can figure out how to make it do something new without destroying its previously designed capabilities. In some ways this is a fairly powerful test of one's understanding of the program or system and its organization. You may not need to know everything about the system, but you have to know enough to be reasonably sure your

Copyrighted material

change has no unintended consequences. This is the meaning of a clean design—it can be revised with a reasonable sense and likelihood of safety. I am glad that this book is so strong on design and emphasizes the role of architecture, and not just algorithms, in design.

There is a great deal more to be said about computing principles, but that is the point of the book that follows. Keeping these principles in mind should make the task of designing computing systems a lot more manageable. Read on!

Vint Cerf
Woodhurst, VA
April 2014

Note

1. W. M. Kahan, "How futile are mindless assessments of roundoff in floating-point computations: Why should we care? What should we do? (Extract)," in *Proceedings of the Householder Symposium XVI on Numerical Linear Algebra*, p. 17, 2005.

Copyrighted material

From Machines to Universal Digitization

The computing *machine* was the center of attention in the early years of the computing field (the 1940s through the 1960s). Computation was seen as the action of machines performing complicated calculations, solving equations, cracking codes, analyzing data, and managing business processes. The leaders in those days defined computer science as the study of phenomena surrounding computers.

Over the years, however, this definition made less and less sense. The computational science movement of the 1980s maintained that computation was a new way of doing science, alongside traditional theory and experiment. They used the term "computational thinking" for a mental practice of inquiry and problem solving, not as a way to build computers. A decade later, scientists in several fields started finding natural information processes in their fields. These included biology (DNA translation), physics (quantum information[4]), cognitive science (brain processes), vision (image recognition), and economics (information flows). The emphasis of computing shifted from machines to information processes, both artificial and natural.

Today, with the digitization of nearly everything, computation has entered everyday life with new ways to solve problems, new forms of art, music, motion pictures, social networking, cloud computing, commerce, and new approaches to learning. Computational metaphors are part and parcel of everyday language with expressions like "My software reacts that way," and "My brain crashed and had to be rebooted."

In response to these changes universities have been designing new principles-based approaches to the teaching of computing. The University of Washington, one of the first at this, developed a course and book on fluency with information technology, now widely used in high schools and colleges to help students learn and apply basic computational principles.[5] The Educational Testing Foundation partnered with the National Science Foundation to develop a new Advanced Placement curriculum based on computing principles.[6] Many people now use the term "computational thinking" to refer to the use of computational principles in many fields and in everyday life, not just in computational science.[7]

As it has matured, the computing field has attracted many followers in other fields. We know of sixteen books that reached out to explain aspects of computing for interested nonspecialists.[8] Most of the books focus on individual parts such as information, programming, algorithms, automation, privacy, and the "guts" of the Internet. We wrote this book to examine

Copyrighted material

the field as a whole and offer an account of how all the parts fit together. Readers will find a coherent set of principles behind all these parts.

In our own experience teaching graduate students who are transitioning into computer science, we have found that a principles framework is easier for beginners than a technology framework. Describing the field in terms of technology ideas was a good approach in the early days when the core technologies were few. In 1989 the Association for Computing Machinery listed nine core technologies. In 2005, however, ACM listed about fourteen, and in 2013 about eighteen. The six-category principles framework does not redefine the core knowledge of computing, but it does provide a new way of looking at the field and reducing its apparent complexity.

Origins and Aims

We are often asked about the origins of the six categories of principles. Author Peter Denning started this project in the 1990s at George Mason University. He collected a list of potential principle statements from many colleagues. He discovered seven natural clusters and named them communication, computation, recollection, coordination, evaluation, design, and automation.[9] When we put this book together, we realized that automation is not a category for manipulating matter and energy; it is instead the focus of the computing domain of artificial intelligence. In this book we deleted automation from the set of categories and included it among the computing domains.

The six categories do not divide the computing knowledge space into separate slices. They are like windows of a hexagonal kiosk. Each window sees the inside space in a distinctive way; but the same thing can be seen in more than one window. The Internet, for example, is sometimes seen as means for data communication, sometimes as means of coordination, and sometimes as a means for recollection.

This set of categories satisfied our goal to have a framework with a manageable number of categories. Although the list of computing technologies will continue to grow, and the set of computing domains will enlarge, the number of categories is likely to remain stable for a long time.

This book is a holistic view of computer science, focusing on the deepest, most pervasive, principles, "cosmic" principles.[10] It presents computing as a deeply scientific field whose principles affect every other field as well as business and industry.

We designed this book for all who use computing science to accomplish their objectives. Scientifically educated readers can learn about the

Copyrighted material

principles of computing spanning the whole field from algorithms to systems. A person inside the computing field can find overviews of less familiar parts of this giant field, such as a programmer who wants to learn about parallel computing. The members of a "computer science for us" class in a college or university can find help to understand how computing technologies affect them, such as how networking and the Internet enable social networks. Budding scientists, engineers, and business entrepreneurs might find here a *Popular Science*–type approach to the whole of computer science.

Acknowledgments

Peter thanks the many people he met on his long journey with the principles of computing, which began when, at the age of eleven, his father gave him a remarkable book about the principles of machines, *How It Works*, published in 1911.[11] His high school math teacher and science club advisor, Ralph Money, encouraged him in 1960 to direct his energies toward computers, the machines of the future. When he became a student at MIT Project Mac in 1964, his mentors Jack Dennis, Robert Fano, Jerry Saltzer, Fernando Corbato, and Allan Scherr broadened his interests to include the fundamental principles behind all computing. His second published paper in 1967, on the working-set principle for storage management, came with the help of major inspirations from Les Belady, Walter Kosinski, Brian Randell, Peter Neumann, and Dick Karp. In 1969 he led a task force to design a core course on operating systems principles, where his teammates Jack Dennis, Butler Lampson, Nico Habermann, Dick Muntz, and Dennis Tsichritzis helped identify the principles of operating systems, with insights from Bruce Arden, Bernie Galler, Saul Rosen, and Sam Conte. In the following years Roger Needham and Maurice Wilkes provided numerous additional insights into the principles of operating systems. He joined with Ed Coffman to write a book on operating systems theory in 1973.

In 1975 Jeff Buzen drew him into his new field of operational analysis, an investigation of the fundamental principles for performance evaluation of computing systems. Erol Gelenbe, Ken Sevcik, Dick Muntz, Leonard Kleinrock, Yon Bard, Martin Reiser, and Mani Chandy all contributed to his understanding during that time.

In 1985 the ACM Education Board asked him to lead a project to identify the core principles of computing as a discipline, for use in designing the 1991 ACM/IEEE curriculum recommendations. He is grateful to the team for deepening his understanding of computing principles: Douglas Comer, David Gries, Michael Mulder, Allen Tucker, Joe Turner, and Paul Young.

Copyrighted material

In the mid-1990s he began to gather all computing principles under a single umbrella. Jim Gray enjoined him to look for "cosmic" principles—ones so deep and vast that they would be true in all parts of the universe at all times. He designed a capstone course called "The Core of Information Technology" and launched it in 1998 with help from his George Mason colleagues Daniel Menascé, Mark Pullen, Bob Hazen, and Jim Trefil.

In 2002 the Education Board of ACM asked him to set up a great principles task force to advise it on how a great principles framework might inform the design of future curricula. What a fabulous team came together to help: Robert Aiken, Gordon Bell, Fred Brooks, Fran Berman, Jeff Buzen, Boots Cassel, Vint Cerf, Fernando Corbato, Ed Feigenbaum, John Gorgone, Jim Gray, David Gries, David Harel, Juris Hartmanis, Lilian Israel, Anita Jones, Mitch Kapor, Alan Kay, Leonard Kleinrock, Richard LeBlanc, Peter Neumann, Paul Rosenbloom, Russ Schackelford, Mike Stonebraker, Andy Tanenbaum, Allen Tucker, and Moshe Vardi.

Rick Snodgrass, a kindred spirit with his pursuit of "ergalics," gave us much sage advice on what makes science in computing. Vint Cerf and Rob Beverly gave many helpful suggestions about the sections on networking.

Peter is most grateful to his wife Dorothy Denning for her constant insistence on clear logical flow and compelling grounding and for her unfailing encouragement for him to take the time needed for writing. He is grateful as well to his daughters, Anne Denning Schultz and Diana Denning LaVolpe, for their faith and support.

Craig Martell was attracted to co-authoring this book because he has the sometimes unfortunate characteristic of always wanting to parameterize the world. Computing as a field presents a particularly fascinating challenge in this regard in that it is equal parts science, mathematics, and engineering. He is constantly fascinated that these machines even work!

Craig would like to thank Mitch Marcus, with whom he co-taught "Information Technology and its Impact on Society" at the University of Pennsylvania. Working through the syllabus for that course began the process that led to his contribution here. He would also like to thank his co-author, Peter Denning; he made the writing process fun, as well as productive. Finally, he would like to thank Pranav Anand, Mark Gondree, Joel Young, Rob Beverly, and Mathias Kölsch, *sine qua taedium*.

Craig's contribution to this book would not have been possible without the patience and multifaceted support of his wife, Chaliya, and the adoring smile of his daughter, Katie. He is convinced that he is, in fact, the luckiest man in the world.

Copyrighted material

1 Computing

Computer science studies phenomena surrounding computers.
—Newell, Simon, and Perlis

Computer science is no more about computers than astronomy is about telescopes.
—Edsger W. Dijkstra

Computing is integral to science, not just as a tool for analyzing data but also as a method 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 (1934), Alonzo Church (1936), Emil Post (1936), and Alan Turing (1936), who saw the importance of automatic computation. They laid the mathematical foundations to answer the question, "what is computation?" and discussed schemes for implementing computations. Their seemingly different schemes were quickly found to be equivalent, as a computation in any one could be realized in any other. It is all the more remarkable, then, that their models all led to the same conclusion that certain functions of practical interest, such as whether a computational algorithm terminates, cannot be answered computationally.

In the time that these men wrote, the terms "computation" and "computers" were already in common use. 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 all named their systems with acronyms ending in "-AC," meaning automatic computer or some close variant—names such as ENIAC, UNIVAC, EDVAC, and EDSAC.

At the start of World War II the militaries of the United States and the United Kingdom became interested in applying automatic computers to

Copyrighted material

Copyrighted image

Figure 1.2
Alan Turing (1912–1954) saw computation as the evaluation of mathematical functions. In 1936 he invented an abstract machine, known now as the Turing machine, to model function evaluation. His machine consisted of a finite state control unit traversing an infinitely long tape with symbols written in each square; on each move the machine reads a single symbol, possibly overwrites with another symbol, moves one square left or right, and enters a new control state. Turing showed how to build a Universal Machine that could imitate any other given its description. He argued that any function that might be called computational could be implemented by one of his machines. He also demonstrated that there are noncomputable functions, such a deciding whether a machine halts rather than going into an infinite loop. (Source: Wikipedia Creative Commons)

Copyrighted material

Copyrighted image

Figure 1.3
After Babbage's failure to build a working Analytical Engine, no one tried to design a general computing machine for the next 80 years. Then, in the late 1930s, the militaries of the United States and United Kingdom sought electronic machines to calculate ballistic firing tables and to crack ciphers. In 1944 the US Army commissioned the ENIAC at University of Pennsylvania under the leadership of John Mauchly and J. Presper Eckert. Its first programmers were Kay McNulty, Betty Jennings, Betty Snyder, Marlyn Wescoff, Fran Bilas, and Ruth Lichterman. The picture shows Jennings (left) and Bilas operating the ENIAC's main control panel. At the time, computers were people and computing was their profession; the electronic machines were called automatic or electronic computers. Programming consisted of wiring plugboards. (Source: University of Pennsylvania)

processes, and deserved to be called sciences even if qualified by the term "artificial."

Computing's Paradigm

For three decades after 1962 traditional scientists often 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, Simon's protests to the contrary, they understood science as a way of dealing with the natural world, and computers looked suspiciously artificial.

Copyrighted material

A B

Figure 1.4
Pioneers (A) John Backus (1924–2007) and (B) Grace Hopper (1902–1992) designed higher-level programming languages that could be automatically translated into machine code by a compiler. In 1957 Backus led a team that developed FORTRAN, a language well suited for numerical computations. In 1959 Hopper led a team that developed COBOL, a language well suited for business records and calculations. Both languages are still used today. With these inventions, the ENIAC picture of programmers plugging wires died, and computing became accessible to many people via easy-to-use languages. Many thousands of programming languages have since been invented. (Source: Wikipedia Creative Commons)

The founders of the field came from all three paradigms.[1] Some thought computing was a branch of applied mathematics, some a branch of electrical engineering, and some a branch of computational-oriented science. During the 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 had largely been met, and computing was 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

Copyrighted material

A B C

Figure 1.5
(A) Allen Newell (1927–1992), (B) Alan Perlis (1922–1990), and (C) Herb Simon (1916–2001) saw computing as a science of phenomena surrounding computers. In 1967 they argued that computer science was a necessary science that studied everything computational, from computing machines, software, intelligence, information, design of systems, graphics, algorithms for solving problems in other fields, and much more. Simon went further and argued that studies of phenomena surrounding man-made artifacts—sciences of the artificial—were just as much science as traditional sciences. (Source: Wikipedia Creative Commons)

challenge" problems in science and engineering. The resulting "computational science" movement involved scientists from many countries and culminated in the US Congress adopting the High-Performance Computing and Communications (HPCC) Act of 1991 to support research on a host of grand challenge problems.

Today, there seems to be an agreement that computing *exemplifies* science and engineering and that neither science nor engineering *characterizes* computing. What then does characterize computing? What is the paradigm of computing?

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. The first, led by Newell, Perlis, and Simon (1967), argued that computing was unique among all the sciences in its study of information processes. Simon called computing a science of the artificial (1969), implicitly agreeing with the common belief that computations are not natural processes. 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 useful information processes. In the early 1970s,

Copyrighted material

A B

Figure 1.6
(A) Donald Knuth (b. 1938) and (B) Edsger Dijkstra (1930–2002) considered programming to be at the heart of computing. Around 1970 they argued that the processes of designing and analyzing algorithms are at the center of everything computer scientists do. To them, a master programmer was a master computer scientist. Unfortunately, this noble view was lost by the late 1990s; governments defined programmers as low-level coders. (Source: Wikipedia Creative Commons)

computing pioneers Edsger Dijkstra and Donald Knuth took strong stands favoring algorithms 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, and because public understanding of a programmer became so narrow (a coder).

The third wave came as a result of the Computer Science and Engineering Research Study (COSERS), led by Bruce Arden (1983) and funded by the National Science Foundation in the 1970s. It defined computation as the automation of information processes in engineering, science, and business. Its catchphrase was "computing is the automation of information processes." Although its final report successfully 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 focus of attention. The computational science movement

Copyrighted material


Copyrighted image

Figure 1.9
Nobel Laureates (A) Ken Wilson (1936–2013), a physicist, and (B) David Baltimore (b. 1938), a biologist, were at the forefront of computational science, which held that computing was a new way of thinking and discovery in science. In the mid-1980s Wilson popularized the notion of "grand challenge" problems in science that could be solved by computational methods, and he advocated highly parallel supercomputers to do the job. In the 1990s Baltimore popularized the notion that biology had become the study of information processes embedded into cells and all life processes. Computer scientists were at first reluctant to be involved but have since embraced computational science and have started a science renaissance in computing. (Source: Wikipedia Creative Commons)

scientists would have described the field by naming its core technologies: algorithms, data structures, numerical methods, programming languages, operating systems, networks, databases, graphics, artificial intelligence, and software engineering. This is a deeply technological interpretation of the field. The principles interpretation used here emphasizes the fundamental laws that empower and constrain the technologies.

The principles of computing fall into six categories: communication, computation, coordination, recollection, evaluation, and design (Denning 2003, Denning and Martell 2004)[2] (see figure 1.10.). These categories are all concerned with manipulating matter and energy to produce intended computations. Table 1.1 defines and illustrates them, and notes which chapters of this book focus on them.

Copyrighted material

Table 1.1
Great Principles of Computing

Category	Focus	Examples	Focal Chapters
Communication	Reliably moving information between locations	Minimal-length codes, error-correcting codes, compression of files, cryptography.	3. Information 11. Networking
Computation	What can and cannot be computed	Classifying complexity of problems in terms of the number of computational steps to achieve a solution. Characterizing problems that have no algorithmic solution.	4. Machines 5. Programming 6. Computation
Recollection	Representing, storing, and retrieving information from media	All storage systems are hierarchical. No storage system can offer equal access time to all objects. The locality principle: all computations favor subsets of their data objects for extended intervals.	7. Memory 11. Networking
Coordination	Effectively using many autonomous computing agents	Protocols that lead the parties to have the same knowledge, eliminate conditions that cause indeterminate results, or synchronize. Choice uncertainty principle.	2. Domains 8. Parallelism 9. Queueing
Evaluation	Measuring whether systems produce intended computations	Predicting system throughput and response time with queueing network models, designing experiments to test algorithms and systems.	9. Queueing 10. Design
Design	Structuring software systems for reliability and dependability	Complex systems can be decomposed into interacting modules and virtual machines. Modules can be stratified corresponding to their time scales of events that manipulate objects.	10. Design

Copyrighted material

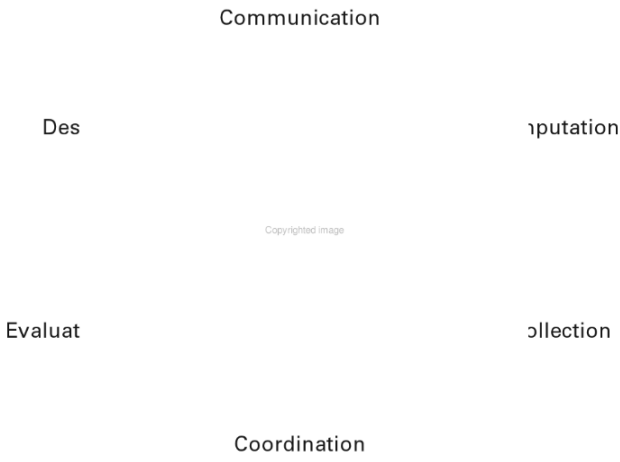


Figure 1.10
Each category of principles is a perspective on computing: a window into the computing knowledge space. (There is no significance to ordering of the category names around the sides of the hexagon.) The categories are not mutually exclusive. For example, the Internet can be viewed from the perspectives of a communication system, a coordination system, or a storage system. Most computing technologies use combinations of principles from all six categories; each category has its own weight in the mixture, but they are all there. These categories also represent mental perspectives people develop about computing. Some people see computing as computation, others as data, networked coordination, or automated systems. The framework can broaden people's perspectives about what computing really is.

There is more to computing than a set of principles and the core technologies that build on them. Computing professionals do the daily work as members of communities that specialize in many computing domains (see figure 1.11). In addition to their knowledge of computing principles, computing professionals are expected to be competent in four core practices: programming, systems thinking, modeling, and computational thinking. A practice is a skill set embodied through continuous practice and interactions with customers. A practitioner's skill can be rated as beginner, advanced beginner, competent, proficient, or expert. A beginning programmer, for example, would be focused on language syntax, getting programs to compile, and finding bugs; an expert programmer would be able to build large systems, solve complex systems problems, and mentor junior programmers. Principles and practices are in constant interaction. People put computing principles to work through skilled action; new principles are occasionally discovered from common practices people have developed.

Copyrighted material

Copyrighted image

Figure 1.11
Computing as a whole depends on both principles and practices. The core technologies are pervasive tools used by practitioners to carry out their work in numerous computing domains. This book concentrates on the principles and their uses in several key domains, leaving core technologies and practices to other books. The principles are either *mechanics*—laws and recurrences—or *design wisdom*—accumulated knowledge about what works or does not work—to build computing systems that are dependable, reliable, usable, safe, and secure.

The communities in which computing people and their customers gather are called *computing domains*. There are dozens of domains. ACM (the Association for Computing Machinery) recognizes no less than 42 domains of professional interest to its members (Denning 2001, 2011), and there are many more under the heading of "computing applications." The next chapter examines four domains of high contemporary interest—security, artificial intelligence, cloud computing, and big data.

A great many of the computing domains interact with other fields outside of computing. In an analysis of how computing interacts with the three great domains of science—physical, life, and social sciences—Paul

Copyrighted material

Rosenbloom found two kinds of interactions: *implementation* and *influence* (Rosenbloom 2004, Denning and Rosenbloom 2009, Rosenbloom 2012). Implementation means that something from one domain is used to create or build something in the other. Influence means that something in one domain affects the behavior of something in the other. Implementation and influence can be single- or bidirectional. Rosenbloom built the chart of table 1.2 to demonstrate the rich set of interactions between computing and all of science in all these dimensions. He included a column for computing in his chart. He did this simply because computing is constantly implementing and influencing itself through the interactions among the many computing domains. There can be no question about the pervasive influence of computing throughout science.

Where Does Computing Fit in Science?

Because computing is so pervasive in its influence in science, and because no other scientific field is directly concerned with information, Rosenbloom came to the conclusion that computing qualifies as the fourth scientific domain.

What is so special about computing's approach to information? Information traditionally means facts that add to knowledge when communicated. It is an old concept, studied for centuries in philosophy, mathematics, business, the humanities, and the sciences. Science is concerned with discovering facts, fitting them together into models, using the models to make predictions, and turning validated predictive models into technologies. Scientists record all they have learned in the structure called the "scientific body of knowledge" for future use. Information has always played a strong role in the sciences.

Computing differs in two ways from the other sciences in its approach to information. First, computing emphasizes the *transformation* of information, not simply its discovery, classification, storage, and communication. Algorithms not only read information structures, they modify them. Moreover, humans constantly modify information structures—such as in the web—with transformations for which we yet have no computational models. Purely analytic methods cannot help us understand the dynamics of these information structures. The experimental methods of science are needed to make progress.

The second difference is that the structures of computing are not just descriptive, they are *generative*. An algorithm is not just a description of a method for solving a problem, it causes a machine to solve the problem.

Copyrighted material

response times of complicated networks of servers when many parallel jobs compete for their services.

Chapter 10 is concerned with design, how to plan and organize computing systems that are dependable, reliable, usable, safe, and secure. Chapter 11, a case study of the Internet, is about how we mobilize other principles to build a vast, reliable data communication network of links and hosts.

We have included a bibliography at the end of the book. The bibliography contains selected items that have inspired us; they are not meant to be historically complete summaries of literature. If you find someone's name in the text, you will also find at least one bibliographical item by that person.

Conclusions

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

The great principles framework reveals a rich set of principles on which all computation is based. These principles support many computing domains and a large number of domains within the physical, life, and social sciences.

Computing is not a subset of the physical, life, or social 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. The computing sciences may well be the fourth great domain of science.

Acknowledgment

This chapter is adapted from "Great Principles of Computing," by Peter J. Denning, *American Scientist 98* (Sep–Oct 2010), 369–372. Republished in *Best Writings on Mathematics 2010,* ed. Mircea Pitici, Princeton University Press (2011). Reused here with permission from *American Scientist* magazine.

Copyrighted material

2 Domains

Biology is an information science.
—David Baltimore

Computation is a third way of doing science, besides theory and experiment.
—Kenneth Wilson

Science and applications of science are bound together as the fruit of the tree which bears it.
—Louis Pasteur

The action of computing comes from people, not principles. Computing people have organized into numerous communities of practice, which we call *computing domains*. Each domain is centered on a technology or an application of technology. For example, the security domain is centered on security technologies and the privacy domain on applications of security technologies to safeguard personal information. The members of these domains share similar concerns, skill sets, methods, and interactions with other communities. They are empowered and constrained by computing principles. The great principles framework would be incomplete without the computing domains (Rosenbloom 2012) (see figure 2.1).

Numerical aerodynamic simulation is an example of a domain. Computer scientists have long collaborated with aeronautics engineers on the design of aircraft. Starting in the 1980s, aircraft companies turned to numerical simulation to design wings and fuselages for efficient, nonturbulent air flows. The traditional methods of wind tunnels and test flights were no longer practical for the size and complexity of aircraft. With new algorithms running on massively parallel supercomputers, engineers were able to design new aircraft that would fly safely on the first flight. The Boeing 777 was the first aircraft completely designed numerically. The teams of

Copyrighted material

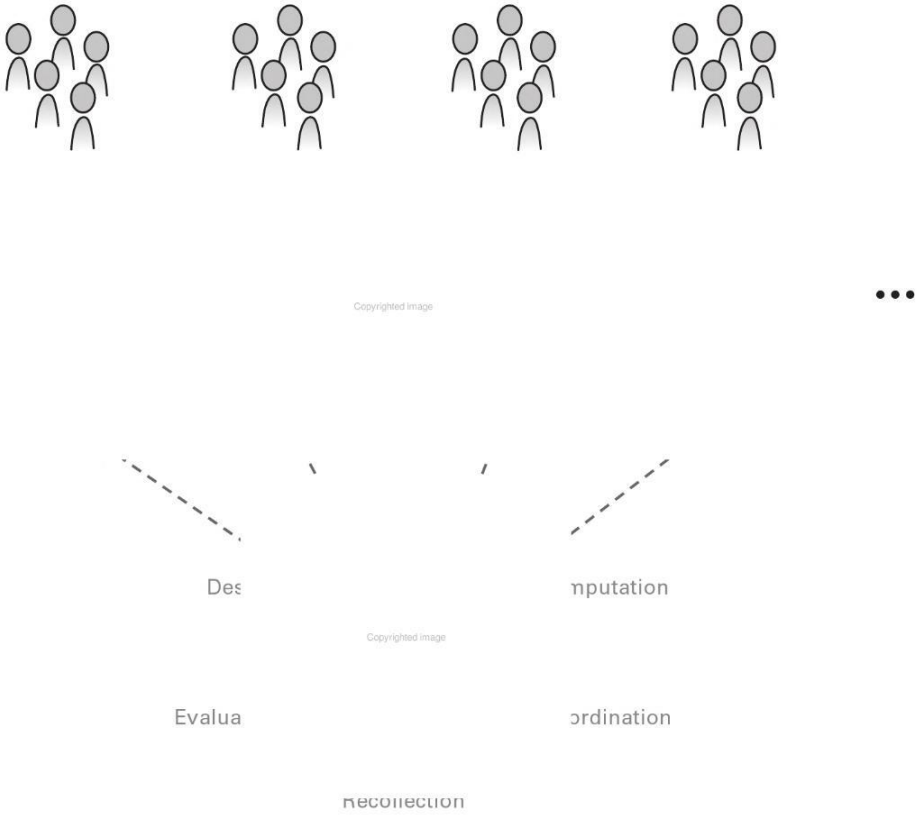


Figure 2.1
The six categories of the great principles framework (*bottom*) are all concerned with managing matter and energy to produce intended computations. In contrast, computing domains (*top*) are communities of practice; their people mobilize computing principles to support solutions to their problems, breakdowns, and interests (*dashed arrows*). The domains also feature strong interactions between computing and other fields. Their work adds principles to computing and to their own fields.

Copyrighted material

aeronautics and computing people developed a new field, computational fluid dynamics, which computed the complex movements of flowing air. They designed computational methods based on 3D grids to solve equations from fluid dynamics in the regions of space around an airframe. They exploited a category of fast *multigrid algorithms*, which solved very large airframes in minimal time on hypercube-connected parallel processors (Chan and Saad 1986, Denning 1987). These teams also developed new methods of refining grids dynamically to add precision in zones of rapid change of air pressure or speed. Some of these methods were recognized as new principles of computing. As a result, computational methods became a permanent part of fluid dynamics.

Computing domains are numerous. The Association for Computing Machinery (ACM) recognizes 42 domains of direct professional interest to its members, and there are dozens of additional application areas and collaborations with other fields (Denning and Frailey 2011). In this chapter we examine four computing domains—security, artificial intelligence, cloud computing, and big data—within a framework that analyzes four factors:

- Who is involved in the domain
- What domain problems, concerns, and interests are taken care of in the domain
- What computing principles are mobilized for the domain
- How domains have generated new principles for computing as well as the other participating fields

This kind of analysis can reveal other principles that could improve a design. It can help other domain participants understand the advantages and limitations of what computing offers them. It could also expose connections between technologies, which might be exploited for future innovations.

Before turning to the examples, it is worthwhile to take a closer look at the relationship between the computing domains and the great principles framework. This understanding will help with the analysis of the domains.

Domains and Principles

There are two basic, useful strategies for representing a field's body of knowledge. One enumerates the domains of the field, the other its principles. These different interpretations of the same knowledge space create different possibilities for actions. For this chapter, we use the term *domain* to mean a technology domain, namely a domain centered on a particular technology.

Copyrighted material

Educators use the term *body of knowledge* (BOK) to mean an organized description of the knowledge of a field. Educators often work with a BOK to design curricula that cover the essential knowledge of their field. The ACM offered its first computing BOK in 1968. It provided updates in 1989, 2001, and 2013. ACM listed nine core domains in 1989 (Denning et al. 1989), fourteen in 2001 (ACM Education Board 2001), and eighteen in 2013 (ACM Education Board 2013). They are core domains because all the other domains depend on their technologies in some way.

A principles framework, as in this book, is orthogonal to a domain-oriented framework. The same principle may appear in several domains, and a particular domain relies on several principles. The set of active principles (those used in at least one technology) evolves much more slowly than the technologies.

Although the two styles of framework are different, they are strongly connected. To visualize the connection, imagine a two-dimensional matrix. The rows are the topics from a domain-oriented framework, and the columns are the categories of principles. The interior of the matrix is the *knowledge space* of the field (see figure 2.2).

With this picture, we can say that the technology-oriented BOK enumerates the knowledge by rows of the matrix, whereas the principles-oriented BOK enumerates by columns. They see the same knowledge—from different perspectives and interpretations.

Imagine someone who wants to enumerate all the principles involved with a technology. That person can analyze the technology domain for its principles in each of the six categories—which corresponds to reading the principles from the row of the matrix (see figure 2.3). That is what we will do with the four example domains in the following sections.

The principles framework opens new inquiries. For example, someone could enumerate all the technologies that employ a particular principle or category of principles (see figure 2.4).

Security

Security as a domain has a long, rich history in computer science. Even in the earliest days, when batch processing was the norm, users were concerned about data entrusted to the machine. Was the machine in a physically secure place? Was the memory cleared before a new job was loaded? Could an operator's mistake or hardware failure lose data?

With the first multiprogrammed, time-sharing systems around 1960, operating system designers got heavily involved in information protection.

Copyrighted material

Copyrighted image

Figure 2.4
The technologies of coordination can be identified by reading the knowledge matrix down the coordination column. Almost all computing domains, including the six illustrated here, employ coordination principles.

freedoms be lost (Garfinkel 2001). In 1999 there were palpable fears of network collapse from the "Y2K" problem caused by encoding years with two digits instead of four. After that many people slowly awoke to the vulnerabilities of information networks and to the challenges of securing them. Experts in many countries began predicting devastating cyber attacks that could ruin economies and even endanger civilization (Schneier 2004, 2008, Clark 2012).

The people, problems, and computing principles of the security domain are displayed in table 2.1.

Artificial Intelligence

The idea of machines performing human intellectual tasks dates back five centuries. Blaise Pascal built a mechanical calculator in 1642. Charles Babbage proposed the Difference Engine in 1823 to calculate navigation and other arithmetic function tables automatically. In the late 1800s the "mechanical Turk" was a convincing hoax appearing to be an expert chess-playing automaton (Standage 2003). Indeed, many of the ideas that have

Copyrighted material

Table 2.1
Security Domain

Who	Members	Operating system designers, network engineers, cyber operators, military defense, law enforcement, forensics investigators, homeland security, public policy officials, diplomats, privacy advocates
What	Breakdowns, problems, concerns	Controlled sharing, memory protection, file protection, access control, information flow, trusted systems, secret communications, authentication, signatures, key distribution, preventing inference through data correlation
Computing principles	Communication	Cryptography, secrecy, authentication
	Computation	One-way functions, cryptographic complexity, hashing, formal verification
	Recollection	Access control, error confinement, information flow, multilevel secure storage, reference monitors
	Coordination	Key distribution, zero knowledge proofs, authentication protocols, signature protocols
	Evaluation	Performance and throughput of protocols, criteria for secure systems
	Design	Open design, least privilege, fail-safe defaults, psychological acceptability, end-to-end designs, layered functions, virtual machines
Principles from other fields		Information assurance practices, intrusion detection, biometric ID, forensic rules of evidence, inference from statistical databases

become the basis of artificial intelligence (AI) predated most of computer science (Russell and Norvig 2010).

In 1956 John McCarthy organized a workshop at Dartmouth with help from Claude Shannon and Nathaniel Rochester. Their workshop gave birth to the field of artificial intelligence. Their founding vision was that "every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it." This appeared plausible because so many intelligent tasks appeared to be following algorithms, and the brain itself appeared to be an electrical network capable of executing algorithms. Herbert Simon predicted that by 1967 a computer would be world chess champion, a computer would discover and prove an important new mathematical theorem, and many theories in psychology would be embodied in computer programs. His first dream was achieved 30 years late, and the other two have yet to be achieved.

Copyrighted material

Alan Turing (1950) crystalized many of the seeds of modern AI: the Turing test, machine learning, and even the idea that we might "grow" an intelligent machine through stages of development, like a child. Turing realized that "intelligence" is so ill defined that he could make no progress with the question of when a machine might be intelligent. His imitation game (the Turing test) asks not whether a machine *has* intelligence but instead whether it *behaves* intelligently. He predicted that by the year 2000 machines would be able to fool 70 percent of determined human interrogators for at least five minutes. That dream also has yet to be achieved.

Turing's behavioral focus was adopted into the founding dream of AI. By the 1970s, however, it became the brunt of sharp criticism. Many AI projects set out to design "expert systems," which would perform as well as human experts in many domains such as medical diagnosis. Hubert Dreyfus (1972, 1992) maintained that expert behavior was beyond the capability of rule-based machines. He was initially ridiculed, but time seems to have proved him right. Only a handful of expert systems worked competently, and none approached genuine experts. John Searle (1984) attacked the notion that conventional computing machines are capable of intelligence; he described a rule-based machine that might appear to carry on conversations in Chinese but did not embody any sort of understanding of Chinese. He attacked "strong AI"—the notion that the mind is a product of machine behavior—and favored "weak AI"—that simulations might imitate a behavior without any resemblance to the way a brain generates the behavior. Terry Winograd and Fernando Flores (1987) argued that AI was based on philosophical assumptions that could not explain or lead to intelligence.

By the mid-1980s it was clear to many that the initial dreams of AI were not going to be achieved any time soon. The research funding agencies began to withhold funds and to demand deliverable results. Many researchers did a lot of soul searching about the weaknesses of their field. AI pioneer Raj Reddy called that dark time the period of "AI winter."

A new field of AI emerged from that introspection. The focus shifted from trying to model the way the human mind works to simply building systems that could take over human cognitive work. Automated cognition systems need not work the same way the human mind works; they do not even need to mimic a human solving a problem. The field simultaneously adopted a strong emphasis on experimental methods to validate whether proposed automations were useful, reliable, and safe (Russell and Norvig 2010, Nilsson 2010). Recent publicity-garnering triumphs include the IBM Deep Blue chess program beating World Chess Champion Garry Kasparov in 1997, Google's driverless car in 2010, and IBM's Watson computer

Copyrighted material

winning the TV game Jeopardy in 2011. The methods used in these programs were highly effective but did not resemble human thought or brain processes. Moreover, the methods were specialized to the single purpose and did not generalize.

Many researchers in computer science, cognitive science, medical science, and psychology continue to study how the brain works and how it generates a mind. The fascination with the singularity (Kurzweil 2005) and the Brain Activity Map Project (announced in 2013) are signs that this line of inquiry maintains its allure.

The reborn AI field has been so successful that it has raised a new set of concerns. In *Race against the Machine*, Erik Brynjolfsson and Andrew McAfee (2012) document how waves of automation are edging out knowledge-work jobs, just as mechanical automation in the previous century had edged out many manual labor jobs. Examples of knowledge automation abound: call centers, voice menu systems, online purchasing, online banking, government services, publishing, news distribution, music publishing, advertising, surveillance, terrorist hunting, and much more. The authors worry that we are sliding toward a society with too few jobs to sustain the population of workers and insufficient resources for public agencies to serve the jobless.

The people, problems, and computing principles of the artificial intelligence domain are displayed in table 2.2.

Cloud Computing

Cloud computing is a modern buzzphrase that hides a rich tradition of information sharing and distributed computing. It refers to networks of computing devices that give economies of scale by hiding the locations of servers and data stores. The term "cloud" came into use in the late 1990s, probably from a practice of showing "the network" as a cloud in technical and marketing presentations.

The idea of building systems that could share computing power among many users cheaply was embodied into MIT Project MAC in the mid-1960s. MAC was an acronym for "multiple-access computer" and sometimes for "man and computer." Project MAC built Multics, a powerful multiplexed system that distributed the expense of memory, disk, and CPU over a large community so that the cost of computing for any one user would be very small. J. C. R. Licklider, the visionary who supplied the initial inspiration, thought that computing power could be supplied like a utility: anyone could plug a terminal into a wall-socket (Licklider 1960).

The ARPANET, which started operation in late 1969, supported the utility ambition. It was designed for resource sharing—users anywhere in the

Copyrighted material

Table 2.2
Artificial Intelligence Domain

Who	Members	AI experts, AI practitioners, artificial lifers, planners, singularity followers, chess players, Jeopardy enthusiasts, Bayesian learners, machine learners, biologically inspired designers, cognitive scientists, human factors designers, psychologists, economists, law enforcers, roboticists
What	Breakdowns, problems, concerns	Automation of cognitive tasks, design and experimental evaluation of heuristics, evolutionary computing, genetic computing, neural computing, pattern recognition, automatic classification, speech recognition, natural language translation, artificial brains, superhuman intelligence, autonomous systems such as drones and cars
Computing principles	Communication	Noisy-channel model
	Computation	Heuristic algorithms, classification, Bayesian inference, machine learning, searching large state spaces, models of intelligence
	Recollection	Memory models, sparse distributed memory, neural network retrieval, locality learning algorithms
	Coordination	Training protocols, coordination theory
	Evaluation	Experimental methods for evaluating heuristics; precision, recall, accuracy
	Design	Storage of large data sets for use in supervised and unsupervised experiments
Principles from other fields		Brains generate minds, speech act theory, linguistics, neuroscience, statistical inference

network could connect with any host and use its services. No one had to replicate a shared service. The ARPANET designers soon realized that shared services would be sought by name rather than location and that location-independent addressing would be the only way to hide the many addressing conventions of local networks containing the services. Vint Cerf and Bob Kahn invented the TCP/IP protocols (1974) to exchange messages between any computers in the Internet knowing only their IP addresses but not their physical locations. The ARPANET standardized on the TCP/IP protocol in 1983.

In 1984, the ARPANET adopted the Domain Naming System (DNS), an online database that mapped symbolic host names to their IP addresses; for example "gmu.edu" maps to "129.174.1.38." This added another level of

Copyrighted material

gene data into genome maps. Tony Chan and Yousef Saad (1986) demonstrated that one of the first parallel architectures, the hypercube, was optimal for a large class of numerical algorithms, called multigrid algorithms, used in solving mathematical models applied to very large data spaces. Jeffrey Dean and Sanjay Ghemawat (2008) designed MapReduce, a new method for mobilizing thousands of parallel processors to solve very large data-processing problems.

Large data sets have always been a concern for businesses. They store data on customers, inventory, manufacturing, and accounting—everything companies need to operate while big and international. IBM became wealthy in the data-processing markets in the 1930s, many years before electronic computers, selling business machines such as card sorters and retrievers. In the 1950s IBM joined a growing number of computer companies that offered electronic data processing. IBM generated considerable publicity in 1956 when it introduced the first hard disk storage system, RAMAC 305. IBM claimed businesses could move warehouses of file cabinets on to a single disk and process the data with amazing speed. As data stores grew, the designers focused on methods to organize the data for fast access and easy maintenance. The two chief competing methods were the Integrated Data System (Bachman 1973) and the Relational Database System (Codd 1970, 1990). The IDS was simple, fast, and pragmatic in its approach to managing large sets of files while hiding the file structure and location on the disks. The RDS was based on the mathematical theory of sets; it had a clean conceptual model but took many years to perfect and achieve the kinds of efficiency seen in IDS. Starting in the 1970s, there has been an active community of researchers who meet annually to discuss issues in very large databases (VLDB).

Beginning in the 1950s, computing researchers helped librarians to organize data for fast retrieval of documents. Libraries were early users of these information retrieval systems. They developed search systems that could deal with fuzzy queries such as "find documents about information retrieval" but without necessarily containing the text string "information retrieval." Today's Internet is a large unstructured store in which keyword retrieval is very fast but imprecise and information retrieval is difficult (Dreyfus 2001).

The Gartner Group defined the modern "big data" domain in terms of four V's: problems with large *volumes* of data, which arrive at high *velocity*, are in a large *variety* of formats, and whose *veracity* is important to decisions based on it. As of 2014, data science courses, centers, and curricula were popping up at universities and research labs. The people involved are from

Copyrighted material

Table 2.4
Big Data Domain

Who	Members	Business, government agencies, enterprise designers, scientific data collectors, statisticians, large systems modelers
What	Breakdowns, problems, concerns	Finding correlated items in very large data sets, computational complexity, privacy issues, inference, forensics of data recovery, information retrieval
Computing principles	Communication	Reliable transmissions from thousands of sensors to repositories, detecting if data have been corrupted, altered, or lost, detecting if data has been placed in illegal jurisdictions
	Computation	Computational complexity of algorithms for data analysis
	Recollection	Storage, replication, error control, testing whether data still exist, testing for physical location of data, forensics data recovery from very large stores
	Coordination	Map-Reduce computing
	Evaluation	Predicting completion times of large searches and analyses in very large networks
	Design	Replication of data, indexing data, structuring for optimal retrieval
Principles from other fields		Natural language processing, statistic inference, mood inference, crowdsourcing, forensics practices

many disciplines including analysts from operations research and statistics, architectures from computer science and information systems, and visualizers from modeling and simulation. The associated "data science" domain is concerned with the scientific basis for analysis and processing of very large data sets.

The people, problems, and computing principles of the big data domain are summarized in table 2.4.

Conclusion

The great principles framework is a useful way to identify bundles of principles making up a technology. It is also useful to identify computing principles that underpin computing domains, in which people from computing and other fields interact to solve persistent problems of concern in their communities.

Copyrighted material

Copyrighted material