Tedre M., Denning P.J. (2017) *Shifting Identities in Computing: From a Useful Tool to a New Method and Theory of Science.* In: Werthner H., van Harmelen F. (eds) Informatics in the Future. Springer, Cham

Chapter 1
Shifting Identities in Computing: From a Useful Tool to a New Method and Theory of Science

Matti Tedre and Peter J. Denning

Abstract Following a number of technological and theoretical breakthroughs in the 1930s, researchers in the nascent field of automatic computing started to develop a disciplinary identity independent from computing's progenitor fields, mainly electrical engineering and mathematical logic. As the technology matured in the next four decades, computing emerged as a field of great value to all of science and engineering. Computing's identity as an academic discipline was the subject of many spirited debates about areas of study, methods, curricula, and relations with other fields. Debates over the name of the field and its relations with older academic departments occupied many hours and journal pages. Yet, over time computing revolutionized practices, then principles, of science and engineering. Almost every field—not just science and engineering, but also humanities—embraced computing and developed its own computational branch. Computing triumphed over all the doubts and became the most important player in science today.

1.1 Introduction

Computing has come to pervade every sector of life. Everyday citizens want to know if it is really true that advanced automation will take their jobs, and whether some outcomes of computing research, such as artificial intelligence, are dangerous. Educators want to know what their curriculum should say about computing or what is meant by popular terms like "computational thinking". Researchers want to

M. Tedre (✉)
Department of Computer and Systems Sciences (DSV), Stockholm University, Borgarfjordsgatan 8, 164 07 Kista, Sweden

School of Computing, University of Eastern Finland, Joensuu, Finland
e-mail: matti.tedre@uef.fi

P.J. Denning
Department of Computer Science, Code CS, Cebrowski Institute, Naval Postgraduate School, 1411 Cunningham Rd GE-314, Monterey, CA 93943, USA
e-mail: pjd@nps.edu

H. Werthner, F. van Harmelen (eds.), *Informatics in the Future*,
DOI 10.1007/978-3-319-55735-9_1

know whether computing can help solve problems in their fields and whether computer scientists who join their teams are peers or just programmers. Students want to know what they would study in computing and what kinds of jobs might be available to them. To give a context for any answer to these concerns, we propose to examine what is computing as a discipline and where it comes from.

Some of us might think that computing as an academic discipline began in the 1990s with the World Wide Web, or in the 1980s with personal computers and networks, or in the 1970s with the microchip and Internet, or in the 1960s with time shared operating systems. Others point to the automatic computing projects started in the 1940s by the military as the beginning, as well as the advances in mathematical logic in the 1930s. It is not easy to pinpoint an exact beginning for computing and computer science. In fact, computing in the sense of calculating numbers has been a human concern for thousands of years.

For millennia, merchants, engineers, and scientists have relied on mechanical instruments to calculate numbers. Tools like the abacus have been with us for as long as historians can see. Mathematicians have produced many clever methods (today called algorithms) for calculating numbers. Some methods eventually were embodied into machines. In an explosion of interest in the 1600s, Pascal invented a simple machine for calculating sums and differences, and Napier invented the logarithm for multiplying numbers fast. The Pascal machine was the first in a long line of machines that became ever better at helping merchants, leading up to marvels such as the Marchant calculator in the 1920s, which with many gears and levers could add, subtract, multiply, and divide. The Napier logarithm initiated a long line of slide rules that were the most popular computing instrument among engineers and scientists for the next 300 years—until 1972 when the Hewlett Packard pocket digital calculator made the slide rule obsolete.

In the middle of these ancient currents, Charles Babbage proposed to the British government in the 1820s that he could build a machine—the Difference Engine—that would calculate numerical tables error-free. He noted that many shipwrecks were the result of errors in hand-calculated navigation tables. Alas, his machine demanded greater precision in lathing the gears and levers than the technology of his day could achieve. Dissatisfied by his inability to produce a working machine, the government cut off his funding in the 1830s. In his frustration with the obstacles in getting the Difference Engine finished, he started designing the Analytical Engine, a new machine with fewer parts that would be programmable and more versatile than the Difference Engine. He collaborated with Lady Ada Lovelace, Lord Byron's daughter, who designed algorithms for the Analytical Engine. She became known to history as a "programmer" of the first general-purpose computer, albeit the machine was a hypothetical one. However, Babbage was unable to get much funding for the machine and the project died with him in 1871.

Development of all kinds of computing instruments continued. In the early 1900s the US government developed an interest in machines for fire control of weapons. It sponsored projects including Vannevar Bush's Differential Analyzer at MIT in the late 1920s for solving differential equations. In the 1930s governments in Germany, UK, and US developed sophisticated electronics for radar and

networks of radars. Some of their engineers became interested in whether the electronics could be used to build machines that computed numbers, resurrecting Babbage's dream. Around 1940, the US Army commissioned a project at University of Pennsylvania to build an automatic computer, ENIAC, that would calculate ballistic tables for artillery. In those days the term "computer" named individuals whose profession was calculating numbers, and the "automatic computer" replaced slow error prone human computers. In 1945 a small group of computer designers led by electrical engineers Eckert and Mauchly and mathematician von Neumann, proposed a design for an electronic stored program computer. The first of these was working by the end of the 1940s and the computer industry was born in the 1950s. By then it had been 80 years since Babbage dreamed of a programmable computer.

One of the great ironies of the computer is that, owing to the complexity of computer systems, it is extremely difficult to ensure that a machine will, even in theory, compute its numbers correctly. Thus the machines Babbage dreamed about turned out to be prone to errors, like the human processes they were to replace. The biggest challenges in computing today continue to be the design of reliable and dependable computer hardware and software.

By 1960 computer science had become a common name for the new field of automatic computation, and academic departments were formed to gather faculty and educate students. In the early years universities resisted the formation of the new departments because computing looked like a technology field without a pedigree. There were endless arguments and debates over the years about whether computing is a legitimate field of science and engineering, and if so, what is its subject matter. Our purpose here is to explore some of these debates and show what identity has emerged in the 80 years since the first electronic digital computers were built. What has emerged is a strong, principled field, which some argue has the status of a new domain of science alongside the traditional domains of physical, life, and social sciences. Let us tell you this story.

1.2 The Birth of a Discipline

A major impetus to computing's emergence as a discipline was given in the late 1940s and early 1950s by a change in computing's status in universities. Many academic pioneers, who were not called computer scientists at the time because there was no such field yet, played important roles in technical innovation in computing. They included, for example, Bush at MIT, Atanasoff and Berry at Iowa State University, Aiken at Harvard, and Kilburn and Williams at University of Manchester (Aspray 2000). But after the World War II, hardware research and development moved quickly to private companies' laboratories; academic computing people were faced with increasing demands to train future computer programmers and engineers. Some universities started to offer courses and degrees in computing in the late 1950s. Still, computing pioneers had to justify their work as a new field that overlapped greatly with mathematics and electrical engineering and

to withstand pressure from academic administrators who wanted computer science to be a branch in one of the existing departments.

Computing's place in the academic world was uncertain at the beginning. In its early days, computers were seen as tools for numerical calculation. A community of mathematicians called numerical analysts devised algorithms for mathematical functions that would not succumb to round-off errors from the machine's use of finite registers to represent real numbers. Engineers worked on making the machinery faster, smaller, more reliable, and cheaper. The job of software design was often left to the numerical analysts, who designed the bulk of software used for scientific and engineering computing. Some business schools also entered the fray with groups that designed software of use in companies such as accounting and tabulating systems.

For nearly four decades after the first electronic computers were built, the people involved were almost all primarily occupied with getting the technology of computers and networks to work well. Despite astounding progress with the technology, the academic field of computing remained an enigma to outsiders in established fields of science and engineering: depending on the observer, it looked much like applied mathematics, electrical engineering, industrial or business applications, or applied science (Denning and Martell 2015). From this diversity arose a perennial question: who should own computer science? The School of Science, which housed Mathematics, the School of Engineering, or even the School of Business? Even more, should computer science be a division of an existing department or a new department?

It is no surprise that the early discussions about organizing the new field were filled with debates about what to name the field (Tedre 2014). Some name suggestions (of which some were less serious than the others) emphasized the field's theoretical elements: Turology, comptology, algorithmics, hypology, and computing science. Others emphasized computing's technical aspects: Computer science, computerology, technetronics, and computics. Some names, like datalogy and informatics, called attention to the "material" that computers process. Others, like intellectronics, bionics, autonomics, cybernetics, and synnoetics, called attention to the field's interdisciplinary and societal nature. Yet others, such as Turingineering, attempted to combine theory and practice into one. Although names create strong impressions of a field's research agenda—its driving questions, research outputs, methodology, valid interpretations of results, and place among other disciplines—names never capture the richness of any field and many fields have evolved well beyond what their names suggest (Knuth 1985).

The intensity of interest in computing and of the debates about computing strongly motivated computing people to form societies and professional organizations early. In 1946 the American Institute for Electrical Engineers (AIEE) founded a society for computing professionals. The next year 78 people convened at Columbia University in New York to found the Eastern Association for Computing Machinery, today known as the ACM. In 1951 the Institute of Radio Engineers (IRE) started another professional group for computing, which after mergers between AIEE and IRE, became the IEEE Computer Society (Tedre 2014).

A division between engineering oriented members of AIEE, IRE, and IEEE and the more mathematically oriented members of ACM emerged early: the ACM focussed more on theoretical computer science and applications, while the engineering associations focussed more on standards, hardware, and technological issues. In addition, a variety of communities of different sizes and foci emerged at a rapid succession and in different countries, each providing to different groups of professionals and academics (Ensmenger 2010).

In the 1950s there was a broad realization of computing's value to science, engineering, business, and various other sectors (Akera 2007). Industry looked to academia to provide computing education for graduates who would be qualified for the rapidly-growing computing sector. Yet, despite generous support from private companies, universities were slow to start computing education (Ensmenger 2010). Many academic computing people, who worked in different departments, found themselves in weak positions: computing lacked independent student and staff quotas, faculty billets, budgets, computing centers, leverage in university politics, and representation in national or international boards (Tedre 2014). There were few directed grants for computing, and research funding agencies such as the National Science Foundation in the US had no computing-specific research programs. In short, computing had a very weak identity. And yet there was a growing desire for independence (Tedre 2014).

1.3 The Quest for Independence

Arguing for independence was a dilemma in its own right. Computing's original cornerstone ideas originated from insights by mathematicians, logicians, scientists, and electrical engineers. Mathematicians and scientists described numerical methods that solved differential equations in small, discrete steps, thus showing that computing opened new doors for scientific discovery based on simulation of mathematical models. Logicians contributed the ideas of representing data and algorithms as strings of symbols in languages; the idea of a universal system that could compute anything any other system of computation could; and the idea that data could be reinterpreted from representing numbers to representing algorithms. Electrical engineers contributed circuits that performed basic arithmetic and that sequenced the operations of an encoded algorithm; they provided means to combine these many circuits into full-blown computers; they discovered that binary representations were easy to generate and led to the most fault tolerant circuits; and they discovered how to use clocks to avoid the devastations of race conditions.

Because these insights originated in the expertise of different fields, arguing that computing was a new field was a tough challenge. Academic advocates of computer science found themselves in the unenviable position of arguing that while it shared aspects with mathematics, logic, science, and engineering, computing was not reducible to those fields. They also had to argue that computing was not merely technology—at the time technology departments were not well regarded in many

academic institutions. The most well known example of an argument that tried to walk this tightrope came from Newell, Perlis, and Simon in 1967 who said that computer science studies phenomena surrounding computers (Newell et al. 1967). There were many critics of this argument, mostly around the belief that only studies of natural phenomena can be science, but computers were artifacts. This critique so irritated Herb Simon, a Nobel Laureate in Economics, that he wrote a book *Sciences of the Artificial* (Simon 1969), which demonstrated that many other established sciences already accepted human constructs as part of their phenomena.

For these reasons, early computing departments started in places of safe refuge in their universities. Those who were most interested in building and studying hardware did so within electrical engineering departments, sometimes forming a computing division of their departments and sometimes adding "computer" to their titles, as in the Electrical and Computer Engineering (ECE) departments that sprang up in the late 1960s. Those who were most interested in computational methods did so within a mathematics context. For example, Purdue founded the first computer science department in 1962 in the Division of Mathematical Sciences in its School of Science. Each university placed computer science in whatever existing school (engineering or science) would protect and nurture it. Within 20 years there were over a 100 Ph.D.-granting computer science departments in the United States and Canada. Yet, the founding of those departments was an intensely political process in the universities (Tedre 2014).

The process of placing new computer science departments in hospitable environments of their universities led, not surprisingly, to an intensification of the debates over the roles of the three roots (engineering, science, and mathematics) in the identity of the field. Some founders such as McCluskey at Princeton, Aiken at Harvard, and Wilkes at the University of Cambridge, believed that the primary work of the field was constructing hardware and software, which they regarded as an inherently engineering task. Some founders, such as Newell, Perlis, and Simon mentioned earlier (Newell et al. 1967), and Forsythe at Stanford (Forsythe 1968), argued that computer scientists should develop an empirical approach. Some, such as McCarthy (1962) and Hoare (1969), advocated a thoroughly reductionist view of computing that was like theoretical physics, where computing was reduced to axiomatic, mathematical, and purely deductive science.

In 1989 a committee of the ACM and IEEE, led by Peter Denning, sought to reconcile these three views by integrating them into a "theory–abstraction–design" model (Denning et al. 1989). The three historical roots of mathematics, science, and engineering contributed important paradigms to the way computing was done, and from their blend emerged the unique field of computing. They recommended using the term "computing" for the field rather than "computer science and engineering", and the name stuck. The Europeans had been using the term "informatics" in the same way and did not change from their established usage. The ACM/IEEE unified view helped, for its part, to pull the field back from two controversies—software engineering and computational science—that nearly split the field in the 1980s and 1990s. We will discuss these controversies shortly. About 20 years later, Denning and Freeman noted that the controversies had settled and they articulated a unique

paradigm for computing that had emerged from the blend of the three root paradigms (Denning and Freeman 2009).

Paradoxically, even though academics tried to distance themselves from computing technology, it was the exponential advance of computing technology, captured by the statement of Moore's law, that opened the space of possibilities for computing to be accepted under its own identity. Every 10 years, chip makers accomplished the amazing feat of increasing the speeds of their chips by a factor of 30 with no increase of size or cost. That meant that computing kept expanding its reach as algorithm designers and engineers invented ingenious ways to automate tasks that only a decade before seemed impossible. On top of this, more and more things were converted into digital representations, meaning that computers came to be able to manipulate almost any information in the world. By the late 1990s, it was clear that computing is indeed a unique phenomenon and requires an independent discipline that aims to understand and harness that phenomenon.

1.4 Search for Disciplinary Identity

In search of the field's disciplinary identity, computing pioneers started to investigate their field's foundations. To start with, all academic fields of science and engineering like to define what they study with a pithy phrase. For example, physics studies the nature and properties of matter and energy. Chemistry studies the nature of substances and their interactions and bonding. Biology is the study of living organisms. What does computing study?

Some earliest proposals—especially that computer science studies computers—were rejected by many scientists who did not see computers as a natural phenomenon and who thought that technology developments in digital electronics did not merit an academic department. Similar, the argument that computer science is the study of phenomena surrounding computers (Newell et al. 1967), although widely quoted among people in computing, did not sell well with other academic departments.

A new argument that "computer science is the study of algorithms" began to emerge in the late 1960s. The idea was that programs are actualizations of algorithms to control machines. Even though the machines and executable programs are physical artifacts, algorithms are abstract mathematical objects that map inputs to outputs, and they can be analyzed formally, using the tools of mathematics. A strong impetus for this argument came from Donald Knuth, whose books *The Art of Computer Programming* developed rigorous analysis of algorithms and became very popular and influential. Another influence was Edsger Dijkstra, who coined the term "structured programming" for his methods of organizing programs (Dijkstra 1972). These pioneers articulated a vision of a programming as an elite, if not noble, calling that required great skill in mathematics, logic, proof, and design to formulate, analyze, and demonstrate great algorithms. They helped to put the algorithm at the center of computing.

Because pragmatic algorithm design most often involves programming, the algorithm-centric movement became known—right or wrong—to many as the "CS = programming" movement. Programming was seen to be the central activity of computer science, and the notion of mathematically oriented elite programmer became the educational goal of many computer science departments. But there were problems with this algorithms-oriented interpretation of the field, too. One was that it often ignored a large segment of the field that was involved in the architecture of computing machines and systems, such as instruction set design, operating systems, databases, graphics, and artificial intelligence. These specialties shared an engineering tradition whereby system builders and software developers constantly evaluate trade-offs in their search for systems that work. The idea of a trade-off did not fit with the idea that algorithms should have precise specifications and be provably correct.

Another problem with the algorithms-oriented interpretation was that, at least in the US, government labor departments did not understand it or agree with it. When they finally added the category "programmer" to their official lists of occupations, they defined a programmer as a fairly low level coder, someone who translates an algorithm design into a working machine code. The official public definition was a small subset of the noble view of programmers held by many pioneers of the field. Computer scientists argued against the programmer-as-coder view for years to no avail, meanwhile continuing to use the term programmer in the way that they wished it to be understood. This produced a large gap of misunderstanding between the general public and working computer scientists, which contributed to an identity crisis that strengthened around 1980 and took another two decades to resolve. That identity crisis surfaced in many ways and it fueled three crucial debates about computing—about experimental computer science, software engineering, and computational science.

Conflicts around experimental computer science became apparent in the late 1970s, and, for various reasons, they manifested first with issues with computing workforce (Feldman and Sutherland 1979). Many faculty members in computer science departments with systems expertise—such as computer architects, operating systems engineers, and graphics experts—were receiving lucrative offers from industry laboratories to join them. This created a "brain drain" that depleted systems faculty. It compounded the problem that many departments were strongly under the mathematics influence and looked to prolific publishing in theoretically oriented journals as a primary measure of academic impact. Systems developers, whose work involved a lot of experimental design and development, published fewer papers and their colleagues did not regard their design solutions and software as legitimate forms of publication. For many systems developers, who faced a hard time gaining tenure, the choice was easy when industry labs offered them positions at twice the salary.

This imbalance troubled many leaders of ACM, IEEE, and industry. They looked for the computer science departments to educate people in computing and prepare them for the workforce, but industry sought much more systems emphasis than the mathematically inclined departments were offering. Jerry Feldman of the

University of Rochester and his team published a report documenting what they called the "experimental computer science crisis." They called for help from the government, especially the US National Science Foundation (NSF) (Feldman and Sutherland 1979). Over the next 2 years the assembly of CS department chairs issued its own "Snowbird Report" on the crisis (Denning et al. 1981), and the ACM executive committee highlighted the crisis and discussed the nature of experimental computer science (Denning 1981; McCracken et al. 1979). The groundswell of community support around these reports led the NSF to create a program called CER (coordinated experimental research) and to fund a proposal to build the CSNET (the ARPANET-inspired network among all CS departments and research labs).

Although the crisis was readily acknowledged and actions taken, there were recurring issues with experimental computer science itself: What does experimental computer science, strictly speaking, mean? Although the mathematicians of the field would not be expected to have much interest in experimental methods, the engineers would. However, most of the engineers of the early days were so focused on getting technology working that they had a different notion of experimentation from traditional scientists. Engineers used the term more in the sense of "tinkering"—a search to find implementations of systems that worked. Traditional scientists think of experiments as means to confirm hypotheses by setting up an apparatus to generate data. These differences of terminology contributed to a sense of vagueness about what experimental computer science is (Tedre and Moisseinen 2014)—was it proof-of-concept, product testing, comparisons of implementations, or controlled experiments? Moreover, after the Feldman and Snowbird reports (Feldman and Sutherland 1979; Denning et al. 1981) the matter of experimental computer science got politicized, which further hindered its adoption (Tedre 2014). But despite terminological wrangling, the efforts to acknowledge computing's unique ways of working gradually produced a more tolerant attitude in academic tenure committees toward design-oriented experimental system research, and the experimental computer science crisis abated by the late 1980s.

Secondly, the software engineering aspects of computing's identity crisis were the outgrowth of a new movement begun in the late 1960s to establish a subfield called *software engineering*. A group of concerned experts from academia, government, and industry convened at a NATO conference in 1968 to decry the expanding "software crisis" and develop an engineering approach to contain it (Naur and Randell 1969). One aspect of the software crisis was that software systems were getting ever larger and more complex, and thus less reliable and dependable, creating many safety and economic hazards. The NATO group called for an engineering approach to developing software, arguing that engineers had mastered the reliability and safety issues in many other fields. This drew many academic departments into the work of defining the software engineering field and designing curricula to teach it.

Despite substantial progress in the next 20 years, software engineers had not tamed the "software crisis". In 1987 Brooks published a famous assessment of software engineering, "No silver bullet" (Brooks 1987). He claimed that a wide

variety of technologies had been developed to help software developers—such as new languages, visualization methods, and version tracking systems—but these were addressing low-hanging fruit and not the hardest problem of all, which is to gain an intellectual grasp of the system and its components. Much software engineering research and education has since concentrated not only on tool development and use, but also on the intellectual disciplines needed to deal with the complexities of systems.

Software engineers soon clashed with the more traditional computer scientists who sought mathematical rigor. Some of these debates got so heated that frustrated software engineers such as David Parnas called for software engineers to split off from computer science and set up as a separate, new software engineering department in the School of Engineering (Parnas 1998). As software engineering matured and understanding of computing's constructive character developed, this controversy also settled down and most computer science departments took software engineers on board.

The third debate centered around computational science. It was much more challenging. Unlike the other two it was not an internal clash among computer scientists; it challenged the relationship between computer science and other sciences. In 1982, Ken Wilson, a theoretical physicist, was awarded the Nobel Prize for his work on a computational method he invented called "renormalization group", which yielded new discoveries about the nature of phase transitions in materials. He began to advocate that all of science could benefit from computational methods implemented by highly parallel supercomputers. He articulated "grand challenge problems" from different areas of science that would be cracked with sufficient computing power. He and others advocated that computation was a new method of doing science alongside the traditional theory and experimental methods. They challenged computer scientists to join with them to help build the systems and work out the methods that would solve grand challenge problems. They started a movement that accumulated considerable political momentum behind its focus on "big science" and was culminated in 1991 by the passage of the High Performance Computing and Communication (HPCC) Act of the US Congress. Many fields of science and engineering started to set up "computational" branches.

The computational science movement was a real challenge for computer scientists, many of whom were troubled with the idea that advances in science from supercomputers seemed to be more valued politically than advances in algorithms and computing theory. Some computer scientists stepped up to join grand challenge teams, only to find that the scientist team members viewed them mainly as programmers rather than full-fledged team members. Wilson and others became exasperated with the reluctance of computer scientists to participate and began calling on Deans of Science to establish new departments of computational science in their universities. Many computer science leaders saw that such a bifurcation would be a disastrous schism in computing and worked hard to avert it. The US NSF established a program in HPCC that enticed many computer scientists to join in computational science projects on the stipulation that they be part of cross disciplinary teams.

By the late 1990s these crises were past. Computer scientists had developed much more compelling articulations of what they do and how they could collaborate with other fields. Many others were taking the field much more seriously. The calls to "fold computer science back into the fields of its roots" faded.

1.5 Emergence of a Science

Just as the old controversies were settling a new challenge to computing's identity began to appear. There were a growing number of references to "natural computing", meaning computational processes in nature. One of the influential pioneers was Nobel laureate David Baltimore, who claimed that biology had become an information science (Baltimore 2002). Others argued similar things for economics, physics, chemistry, cognitive science, and other fields in the physical, life, and social sciences (Kari and Rozenberg 2008).

The rapid emergence and development of natural computing is an amazing turnaround from the 1960s, when Simon argued that computing is a science even if it studies artificial phenomena, and when many others believed that the field of computing is fully reducible to mathematics. Today scientists in other fields are saying that their fields include information processes that occur naturally. They are telling computer scientists, "We have information processes too!" This shift has led to a new definition of computing as a discipline as the study of information processes, both artificial and natural. This definition is important because it shifts the focus from the computer to information processes and their transformations. It is also a much more inclusive definition that allows for the computational branches of other fields, including the natural sciences.

This definition also accommodates the three intellectual traditions embedded in computing, which we mentioned earlier—theory, design, and abstraction. The theoretical tradition focuses on mathematical relationships and proofs. The design tradition focuses on the design and construction of computational circuits, machines, systems, and software. The abstraction tradition focuses on experimental work to test algorithms, validate software, find workable system configurations, and support design choices; it was advocated early (Feldman and Sutherland 1979; McCracken et al. 1979) but took many years to develop and earn a stature comparable to the theory and design methods. A combination of these three traditions gained wider currency in computing when other fields acknowledged computing as a third way of doing science and developed computational branches (Denning and Martell 2015).

The 1980s experimental computer science debates, which brought methodology to limelight, were supported in the mid-1990s by methodological meta-analyses, fashionable in many other fields. Many people and research groups analyzed journal articles and conference papers in computing and other fields in order to describe methodology in computing and compare computing research with research in other fields (Tedre 2014). Those meta-analyses found great methodological differences between computing's branches, but also revealed widespread disregard

of methodological rigor in computing publications. For example, in a meta-analysis of software engineering literature, Walter Tichy found that only a small fraction of published papers performed experiments that would validate their claimed hypotheses (Tichy et al. 1995).

Meta-analysts urged their computer science colleagues to follow the example of other fields, especially physics, and strive to make computing similar to the older, more established fields. And indeed, by the end of the millennium natural sciences and computing were converging at a large scale—but mostly because natural sciences were becoming more like computing rather than computing becoming more like natural sciences (Denning and Martell 2015; Tedre 2014). Simulation started to compete in popularity with experiments in sciences and with prototyping in engineering (Wilson 1984). Following the success of computational sciences, national governments increased investments in high performance computing and in computing-intensive research to an extent that was called the "supercomputer race" between countries (Wilson 1984). Numerical analysts, having long felt being the "queer people in computer science departments" (Forsythe 1968) found themselves in the limelight again.

With the rise of computational science and penetration of computing into literally all areas of life in an increasing number of countries, computer scientists' disciplinary ways of thinking and practicing gained currency among educators, too. Those ways of thinking and practicing were called "algorithmizing" by Perlis in 1960 (Katz 1960), a "general-purpose thinking tool," by Forsythe in the late 1960s (Forsythe 1968), and "algorithmic thinking" in the 1970s and 1980s by, for instance, Knuth and Statz (Knuth 1985; Statz and Miller 1975). In the 1990s, with the computational science movement, the catchphrase became "computational thinking". Computational thinking was claimed as a valuable educational approach in the 1990s by Papert (1996) and was popularized as valuable for all children by Wing in the 2000s (Wing 2006). Educators gradually understood that in order to prepare students for the computing-pervaded work and world they will face in the future, it is crucial to familiarize them with computing's disciplinary ways of thinking and practicing.

This has raised a new question: where does computing fit in the firmament of all the sciences? There are three generally accepted domains, or families, of science: physical, life, and social sciences. Where does computing fit in? In 2004 Paul Rosenbloom examined the nature of the interactions between computing and other fields. He found that computing either influences or implements processes in virtually every field of science, engineering, and humanities (Rosenbloom 2004). He also found that these are two-way interactions, with the other fields influencing computing. There was no neat fit of computing into any of the three traditional domains. He made a bold claim that computing is a new domain, which he called the computing sciences, and is a peer of the three traditional domains.

The triumph of computing in sciences was evident by the beginning of the new millennium. The first computing revolution in science—the introduction of powerful tools for solving scientific problems—was largely complete. A look into any laboratory in natural sciences would show that computer simulations, numerical

methods, and computational models had become standard tools for science, and many other fields in social sciences and humanities were developing computational branches, too. Computer proofs also led mathematicians to re-think the very idea of "proof" and computers became popular at all stages of mathematical discovery (Horgan 1993).

At the same time, a second computing revolution in sciences was well underway. An increasing number of researchers from natural sciences to humanities started to look at their fields through a computational lens, interpreting phenomena in their field as information processes. The new, info-computational model of science (Dodig-Crnkovic and Müller 2011) was greeted as "algorithmization of sciences," (Easton 2006) "the idiom of modern science," (Chazelle 2006) and "the age of computer simulation" (Winsberg 2010). These arguments harkened back to the 1980s, when computational scientists argued that the twin pillars of science—theory and experiment—were now joined by a third, equally important pillar—computing.

Today, computing's effectiveness in sciences has led to two versions of natural computing: the weak argument and the strong argument. "Weak" natural computing states that computers are a great tool for studying the world, and info-computational interpretations of phenomena are very useful abstractions. Few researchers today would disagree with that standpoint. But with the rise of info-computational interpretations of natural phenomena, various researchers started to advance "strong" natural computing—that calculation and interpretation are not enough, but computing plays a fundamental role in the naturally occurring processes of their domains. The strong argument for natural computing is one of the most exciting (and controversial) in the modern history of science: That argument suggests that there must be a reason for the amazing success of computing in predicting and modeling phenomena across all fields of science, and maybe that reason is that the world itself is an information processor. Maybe the world computes.

Over the years pioneers of computing have argued, on various levels and in different ways, for info-computational views of the world. Zuse argued in 1967 that the universe is a cellular automaton (Zuse 1970), Hillis wrote that molecules compute their spatial configurations (Hillis 1998), and Mitchell wrote that living organisms perform computations (Mitchell 2011). Wolfram wrote a whole book making the claim that computing is a new kind of science (Wolfram 2002). Chaitin wrote that the universe constantly computes its own future states from its current ones—"everything is made out of 0/1 bits, everything is digital software, and God is a computer programmer, not a mathematician!" (Chaitin 2006). Whereas Galilei's famous dictum was that the book of nature is written in mathematics, the proponents of weak natural computing would argue that the book of nature is written in algorithms, and the proponents of strong natural computing would argue, in the words of Dodig-Crnkovic, that the book of nature is an e-book: The book itself computes. And, in the minds of many, that makes computing not only *a* science, but *the* science.

1.6 Conclusions

As computing technology moved ever deeper into people's lives, people's perceptions about what computers are evolved and set a context in which the field's identity developed as discussed here.

The first stage of public understanding was computers as number and symbol crunchers. From the earliest days, computing was linked in the public mind to the brain and to intelligence, perhaps because the machines were doing computational tasks that everyone thought only humans could do. The news reports of the first commercial Univac computer in 1950 used the terms "thinking machines" and "electronic brains" (Martin 1993). The business world quickly embraced the computer revolution bringing its own long tradition of "symbolic data processing"—punched-card machines analyzed data such as gender, literacy, and occupation in the 1890 U.S. census and IBM built a strong business machines company in the 1920s. These traditions quickly shifted the interpretation of computers from crunchers of numbers to crunchers of arbitrary patterns of symbols (Hamming 1980).

The second stage of public understanding happened in the 1980s with the emergence of the Internet—computers were seen as communication machines rather than symbol crunchers. The modern cloud-mobile computing age, which realizes a 1960s dream of "computer utility", is a culmination of this way of seeing computing.

The third stage of public understanding started in the early 2000s with the claims that information processes are found through the natural sciences. This interpretation is supported by amazing developments throughout science, where computing blends with other technologies. It has also fostered anxieties about artificially intelligent machines automating most jobs out of existence and perhaps becoming an apocalypse for the world.

In the academic world, the first 40 years of computing focused a lot on developing the technology of computers and networks. Much of the content of curricula reflected the core technologies of computing. With the Internet in the 1980s and Web in 1990s, computing curricula began to adopt social dimensions that featured applications in and interactions with other fields. Today, with the rise of natural information processes, computing is now seen as fundamental in all the sciences and engineering and may even define a new scientific domain. Computing's development as a science and its integration into other areas of scientific inquiry is unrivalled.

Computing's self image has influenced and evolved with these larger changes. Gone are the defensive essays over whether computing is engineering, math, or science. Gone are the internal fights about how to deal with experimental science, software engineering, or computational science.

Computing has fostered two revolutions. The first was computing as a tool with unprecedented power and versatility for scientific computing and simulation, fundamentally changing how science was done in practice. The second was computing as a completely new way of looking at natural and artificial phenomena, fundamentally changing how other fields see themselves and do their work. The info-computational theory of science provides a new ontology, epistemology, methodology, and principles of scientific inquiry (Dodig-Crnkovic and Müller 2011).

The circle closed in the 2000s, when reductionist claims that computing was really other fields were flipped: now we see essays heralding a new era of science, explaining why all other sciences can be reduced to computing. Computing has begun its second revolution in science.

References

Akera A (2007) Calculating a natural world: scientists, engineers, and computers during the rise of U.S. cold war research. MIT Press, Cambridge

Aspray W (2000) Was early entry a competitive advantage? US universities that entered computing in the 1940s. IEEE Ann Hist Comput 22(3):42–87

Baltimore D (2002) How biology became an information science. In: Denning PJ (ed) The invisible future. McGraw-Hill, New York, pp 43–55

Brooks FP Jr (1987) No silver bullet: essence and accidents of software engineering. IEEE Comput 20(4):10–19

Chaitin G (2006) Epistemology as information theory: from Leibniz to Ω. In: Stuart SAJ, Dodig-Crnkovic G (eds) Computation, information, cognition: the nexus and the liminal. Cambridge Scholars, Newcastle, pp 2–17

Chazelle B (2006) Could your iPod be holding the greatest mystery in modern science? Math Horiz 13(4):14–15

Denning PJ (1981) Eating our seed corn. Commun ACM 24(6):341–343

Denning PJ, Freeman PA (2009) Computing's paradigm. Commun ACM 52(12):28–30

Denning PJ, Martell CH (2015) Great principles of computing. MIT Press, Cambridge

Denning PJ, Feigenbaum E, Gilmore P, Hearn A, Ritchie RW, Traub J (1981) A discipline in crisis. Commun ACM 24(6):370–374

Denning PJ, Comer DE, Gries D, Mulder MC, Tucker A, Turner AJ, Young PR (1989) Computing as a discipline. Commun ACM 32(1):9–23

Dijkstra EW (1972) The humble programmer. Commun ACM 15(10):859–866

Dodig-Crnkovic G, Müller VC (2011) A dialogue concerning two world systems: info-computational vs. mechanistic. In: Dodig-Crnkovic G, Burgin M (eds) Information and computation: essays on scientific and philosophical understanding of foundations of information and computation, world scientific series in information studies, vol 2. World Scientific, Singapore

Easton TA (2006) Beyond the algorithmization of the sciences. Commun ACM 49(5):31–33

Ensmenger NL (2010) The computer boys take over: computers, programmers, and the politics of technical expertise. MIT Press, Cambridge

Feldman JA, Sutherland WR (1979) Rejuvenating experimental computer science: a report to the National Science Foundation and others. Commun ACM 22(9):497–502

Forsythe GE (1968) What to do till the computer scientist comes. Am Math Mon 75:454–461

Hamming RW (1980) The unreasonable effectiveness of mathematics. Am Math Mon 87(2): 81–90

Hillis WD (1998) The pattern on the stone: the simple ideas that make computers work. Basic Books, New York

Hoare CAR (1969) An axiomatic basis for computer programming. Commun ACM 12(10): 576–580

Horgan J (1993) The death of proof. Sci Am 269(1993):74–82

Kari L, Rozenberg G (2008) The many facets of natural computing. Commun ACM 51(10):72–83

Katz DL (1960) Conference report on the use of computers in engineering classroom instruction. Commun ACM 3(10):522–527

Knuth DE (1985) Algorithmic thinking and mathematical thinking. Am Math Mon 92:170–181

Martin CD (1993) The myth of the awesome thinking machine. Commun ACM 36(4):120–133
McCarthy J (1962) Towards a mathematical science of computation. In: Proceedings of IFIP congress 62: information processing, Munich, pp 21–28
McCracken DD, Denning PJ, Brandin DH (1979) An ACM executive committee position on the crisis in experimental computer science. Commun ACM 22(9):503–504
Mitchell M (2011) Ubiquity symposium: biological computation. Ubiquity, 2011 February, pp 1–7
Naur P, Randell B (eds) (1969) Software engineering: report on a conference sponsored by the NATO Science Committee. NATO Scientific Affairs Division, Brussels
Newell A, Perlis AJ, Simon HA (1967) Computer science. Science 157(3795):1373–1374
Papert S (1996) An exploration in the space of mathematics educations. Int J Comput Math Learn 1(1):95–123
Parnas DL (1998) Software engineering programmes are not computer science programmes. Ann Softw Eng 6(1998):19–37
Rosenbloom PS (2004) A new framework for computer science and engineering. Computer 37(11):23–28
Simon HA (1969) The sciences of the artificial, 1st edn. MIT Press, Cambridge
Statz J, Miller L (1975) Certification of secondary school computer science teachers: some issues and viewpoints. In: Proceedings of the 1975 annual conference. ACM, New York, ACM '75, pp 71–73
Tedre M (2014) The science of computing: shaping a discipline. CRC Press/Taylor & Francis, New York
Tedre M, Moisseinen N (2014) Experiments in computing: a survey. Sci World J 2014(#549398): 1–11
Tichy WF, Lukowicz P, Prechelt L, Heinz EA (1995) Experimental evaluation in computer science: a quantitative study. J Syst Softw 28(1):9–18
Wilson KG (1984) Planning for the future of U.S. scientific and engineering computing. Commun ACM 27(4):279–280
Wing JM (2006) Computational thinking. Commun ACM 49(3):33–35
Winsberg EB (2010) Science in the age of computer simulation. The University of Chicago Press, Chicago
Wolfram S (2002) A new kind of science. Wolfram Media, Champaign
Zuse K (1970) Calculating space. Technical translation AZT-70-164-GEMIT. Massachusetts Institute of Technology, Cambridge