This is an electronic reprint of the original article.
This reprint may differ from the original in pagination and typographic detail.

Tedre, Matti; Simon, Simon; Malmi, Lauri
Changing aims of computing education

Published in:
Computer Science Education

DOI:
[10.1080/08993408.2018.1486624](https://doi.org/10.1080/08993408.2018.1486624)

Published: 01/01/2018

Document Version
Peer reviewed version

Please cite the original version:
Tedre, M., Simon, S., & Malmi, L. (2018). Changing aims of computing education: a historical survey. *Computer Science Education*, 28(2), 158-186. <https://doi.org/10.1080/08993408.2018.1486624>

This material is protected by copyright and other intellectual property rights, and duplication or sale of all or part of any of the repository collections is not permitted, except that material may be duplicated by you for your research use or educational purposes in electronic or print form. You must obtain permission for any other use. Electronic or print copies may not be offered, whether for sale or otherwise to anyone who is not an authorised user.

Changing aims of computing education: a historical survey

Matti Tedre^a, Simon^b and Lauri Malmi^c

^aSchool of Computing, University of Eastern Finland, Joensuu, Finland; ^bSchool of Electrical Engineering and Computing, University of Newcastle, Ourimbah, Australia; ^cDepartment of Computer Science, Aalto University, Espoo, Finland

ABSTRACT

Computing education has been an important and sometimes contentious issue ever since the advent of modern computing. Debates about computing education have closely followed job markets, technological development, academic interests, societal concerns, and changes in the perception of computing. The themes in computing education debates can be characterized by emergence and formation, standardization and organization, accommodation to change, and divergence. The focus of computing education has expanded outward from the computer to programming, algorithms, and information, as well as to the organizational, social, and cultural environment of computer systems.

This survey gives computing education researchers an overview of some of the central issues and disputes in computing education over the brief history of modern computing. The survey highlights the emergence of educational initiatives, concepts, joint efforts, and institutions of computing education, and outlines the relatively short history of computing education research. The survey is structured around four overlapping themes: computing education as technological training, as training for software development, as a central element for the field's academic recognition, and as training for computational problem-solving in any domain of knowledge. Each theme has played a role throughout the history of modern computing, but their relative emphases have changed over the years.

KEYWORDS

Computing education; computer science education; CSE; computing curricula; history; History of computer science education

1. Introduction

Computing machinery and the theory of computing have rich and long histories, but the history of computing as an academic discipline is much shorter. Computing as a discipline has no clear date of origin, having emerged from several intellectual strands each with its own history (Mahoney, 2011). On the theoretical side of computing, it is not possible to pinpoint a moment when computing-related topics started to be associated more strongly with

computing than with mathematical logic. On the engineering side, the fields of electrical engineering and computing co-evolved for decades before clear dividing lines started to develop (Ceruzzi, 1988). Likewise, academic education for computing has undergone a gradual development of its own identity, and over the past two decades computing education research has matured into a respectable field (Simon, 2015).

Along with the development of computing as a discipline we have seen a rise in research on the history of computing. The history of computing is now a rich discipline with multiple progressive branches – and a history of a discipline is a telltale sign of a mature field. Accounts of the history of computing education have emerged slowly, usually as a side track to some other topic (Ensmenger, 2010; Mounier-Kuhn, 2012; Tedre, 2014) but occasionally focused on the history as a topic in its own right (Goldweber et al., 1997; Gupta, 2007; Tomayko, 1998). However, these accounts are often narrow in scope and aimed at specialist audiences of historians. There is a shortage of accessible accounts of the history of computing education from which computing educators and computing education researchers can learn about the roots of their field: how did computing education emerge, what has driven its development, and how have its concerns changed over the years?

This article aims to fill that gap in the literature by presenting a broad overview of the development of computing education and its different trajectories that have arisen, flourished, and faded away. The article outlines the emergence of computing education efforts and the roles played by different groups of stakeholders, along with debates about its proper focus, subject matter, and skill set. The article deals principally with developments on the U.S. scene; this is due in part to the large amount and ready availability of historical material on that scene, and in part to the major role that has been played by the US-based *Association for Computing Machinery* (ACM). As a historical work, the article covers the span from the late 1940s to the early 2000s, stopping short of several recent educational initiatives and curriculum recommendations by ACM and other central organizations. The article relies on some research by professional historians, but because of its purpose as a general overview, its main audience is computing educators.

In the absence of a clear defining moment for the birth of computing as a discipline, this review of gradual shifts in computing education starts from one of the most influential advocacy events of the stored-program computer idea – the 1946 ‘Moore School Lectures’ on the construction of digital computers, organized at the Moore School of Electrical Engineering at the University of Pennsylvania. Those lectures gathered together many leading people in computing-related topics, who in the subsequent years undertook computer construction projects at their own institutions, usually following the logical computer designs advocated at the Moore School lectures. Computing education has some roots of slightly older origin – for instance, Grier (2005, p.239) discusses training initiatives for

human computing projects that preceded the advent of stored-program computers – but it was in the late 1940s that it began to appear in universities as a distinct field of study.

2. Training for technical jobs

In the late 1940s computing was generally not a separate topic of education, although pioneers such as Aiken, Eckert, and Grosch launched educational programs for the use and design of large-scale machine computing and numerical methods (Aspray, 2000; Cohen & Welch, 1999, p.xvi). Although the computing pioneers came from a variety of backgrounds, such as electrical engineering, natural sciences, and mathematics (Ceruzzi, 1988), the central problems in the field at the time were about how to make the new kind of machinery work reliably or even work at all, and accordingly, the field was dominated by technologists (Wegner, 1970). For example, the famous 1953 book *Faster than Thought* saw the problems of computing at the time as mostly technological, requiring no familiarity with mathematical logic (Bowden, 1953, pp.30–31). At the same time, an academic computing community began to emerge and to search for a direction and a place in the academic world (Tedre, 2014).

When each computer had to be designed and built from scratch and when the user base was limited to engineers and specialists, there was little need for mass education in computing, but the introduction of mass-produced stored-program computers just before the mid-1950s changed the game. When fully electronic stored-program computers started their march into universities, research laboratories, and offices, and when the user base started to diversify, a clear need emerged for academic and professional education for a broad variety of computing jobs that were not electrical engineering jobs (Ensmenger, 2001, 2010; Grudin, 1990; Jacobson, 1955). Pioneers such as Aiken had in the 1940s anticipated a growing need for computing specialists (Cohen & Welch, 1999), and indeed, in the 1950s there was broad acknowledgment of an urgent need for education for computing professions. In June 1954 the *First Conference on Training Personnel for the Computing Machine Field*, held at Wayne University in Michigan, brought together more than 150 influential people from industry and academia, including a large number of famous pioneers of computing (Jacobson, 1955), to discuss the workforce problem and educational needs. Educational efforts took place in a variety of contexts, such as business computing and scientific computing.

In the 1950s universities faced difficulties in responding to the sudden need for academic computing education (Fein, 1959, 1961). Education had to be started from scratch: curricula had to be designed, staff trained, textbooks written, and course material, facilities, and examinations prepared. The 1954 conference on computing machine education noted an acute shortage of personnel and inadequacy of training programs, and the attendees bemoaned

a severe lack of literature, courses, teaching staff, suitable curricula, and even mental models for thinking about computing (Hohn, 1955; Kircher, 1955). The first textbook on programming is generally considered to be *Preparation of Programs for an Electronic Digital Computer* (Wilkes, Wheeler, & Gill, 1951), which used a newly developed notation that preceded the modern assembler, introduced innovations such as subroutine libraries, and even had a chapter on debugging.¹ But textbooks, material, and experience were rare, largely machine-specific, and aged quickly as the discipline and the technology matured.

While universities were facing great problems in meeting the training demands of industry, most of the early computing workforce in the U.S. was trained by private companies using their own training programs (Ensmenger, 2010). One survey in 1954 found that just 37% of responding companies found colleges and universities to be productive sources for the recruitment of computing personnel, and 85% found it necessary to train their own personnel (Mengel, 1955). However, Ensmenger (2010, p.56) argued that the company training was “ludicrously inefficient”: in 1954 the combined training capacity of all computer manufacturers was estimated to be 260 programmers annually, which was woefully inadequate for a period when mass production of computers had started and the number of computer installations was growing rapidly (Ceruzzi, 1988). Educating programmers was not perceived to be very time-consuming: one manufacturer estimated that programmers could be trained in two weeks, while another said that it might take twelve weeks (Chinitz, 1955). Compounding the lack of consensus over skills, best practices, standards, and tools, each distinct type of computer had its own idiosyncrasies and tricks to make it work (e.g. Metropolis, Howlett, & Rota, 1980). To improve their training results, companies used oft-criticized aptitude tests and personality profiles to select good candidates, even when these were shown by the companies’ own research to be of dubious value (Rowan, 1957). Despite their shortcomings, companies’ training programs were able to keep the computing industry supplied with workforce; for instance, the *System Development Corporation* (SDC) was able to effectively double the programmer workforce of the U.S. over just a few years (Ensmenger, 2010, p.60).

Meanwhile, universities began to catch up on computing education. IBM’s educational computer “grants” to universities made it easier and cheaper to start computing education – for IBM brand computers, of course (Ceruzzi, 2003; pp.43 – 44; Galler, 1986). Towards the end of the decade, computing education began to become institutionalized, programs began to resemble one another, and academic visions for educating computing researchers began to crystallize. By 1959, about 150 universities were offering some training in computing (Fein, 1959), although the necessary curriculum content was still being debated and training was often still directed towards specific jobs.

The courses being offered were a mixture of theory, scientific computing, and engineering. Courses on the theory side included information theory and theory of coding; those on the science side included linear programming and numerical analysis of differential equations; and those on the engineering side, electrical engineering and digital circuitry (Fein, 1959). Given IBM's role in computerizing universities, the disorganized state of computing theory, and the image of computing as a service discipline, it is no surprise that a large number of applied topics were included, topics such as business forecasting, industrial analysis, and use of electronic data processing. Yet these topics were far from teaching how to use computers: they were often very technical and demanding, with mastery of programming dependent upon an underlying knowledge of the internal workings of computers (Cohen & Welch, 1999, p.78). In the absence of computing as a disciplinary entity within the university structure, courses were often taught by people who were not faculty members.

Early computing education efforts were hampered by the absence of shared views of the profession of computing (Fein, 1959) and the qualities of a capable computing professional (Editors of DATA-LINK, 1958; Ensmenger, 2001, 2010), which were central to the debates of the era. Coders and computer operators, for instance, were deemed technical staff or service personnel who did not require university training (Atchison et al., 1968). Programmers were treated with suspicion, as programming was seen largely – and with some justification – as a craft and a domain of artistic talent (Backus, 1980). Backus (1980) recollected that programmers were often self-taught and that the technical limitations of early computers required tricks, tinkering, and familiarity with their idiosyncrasies. The resulting programs were often messy, hard to understand, and extremely difficult to debug. Among the self-taught experts in a young field there was a lack of commonly shared mental models for thinking about programming. Dijkstra (1980) expressed “a sense of marvel” that despite the “primitive way of thinking about the programming task”, programmers at the time could still produce so many non-trivial programs. In the absence of higher-level models for algorithmic thinking, coding and tinkering was what most programmers did (Backus, 1980; Dijkstra, 1980).

As the field continued to grow, an increasing number of conferences and specialist periodicals were established, consensus about requisite skills began to develop, professions emerged, vocabulary began to stabilize, and tools and theories advanced. But at the same time, in discussions about computing education, gaps between theorists, engineers, and practitioners continued to grow. As computer technology matured and mass-produced computers took over the markets, universities no longer needed to build their own computers (Fein, 1959) and the face of computing in higher education institutions changed. Computer engineering found its natural home in engineering schools and technically- oriented institutes of technology, while theory- and software-

related topics were often seen as better suited for traditional research universities. Much work that is today considered to be core computing research was done in the field of mathematical logic; examples include Turing's work on computability, the non-deterministic machines of Rabin and Scott, the regular languages of Kleene, and von Neumann's work on automata theory. Pioneering theoretical research was carried out in fields such as complexity theory, information theory, and formal semantics (Mahoney, 2011). Through the 1950s Turing's work gained increasing recognition and connection with the computing discipline (Daylight, 2014, 2016), but the disciplinary identity of computing and its proper administrative location in the university structure remained unclear (Aspray, 2000). Mathematics and science were highly regarded in traditional research universities, but technical topics, as computing was considered by many universities, were not always seen so favorably.

3. Training for software development

By the 1960s several major organizations had started educational and curriculum efforts in computing, and developments took place on a broad front. The software industry grew from individual software contractors in the 1950s to corporate software product developers in the 1960s, and to mass-market software houses in the 1970s, each decade expanding the combined revenue of the industry tenfold (Campbell-Kelly, 2003, pp. 9–19). The computing education sector continued to struggle to meet the rapidly growing demand for workforce, but universities were catching up. In the early 1960s, university computing centers played a major role in computing education (Finerman, 1968b). A 1960 seminar organized by IBM brought together university computing center directors to discuss computing curricula and to exchange information about curriculum development (Morse, 1960). The seminar foresaw a number of developments: computing becoming an independent discipline, formation of departments of computing, and computers becoming as important for scientific research as laboratories.

Universities as well as other organizations competed to establish the legitimacy of their training programs. The *Data Processing Management Association* (DPMA; formerly the *National Machine Accountants Association*, NMAA, and now the *Association of Information Technology Professionals*, AITP) had for years developed its educational program, offering its first Certificate in Data Processing professional examination in 1962 (Ashmore, 1985). DPMA's competitor, ACM, started an education committee, headed from 1960 by Louis Fein. The *International Federation for Information Processing* (IFIP) also organized a panel on education in information processing in 1962 (Gupta, 2007).

Not only did academic initiatives compete against one another and against the other organizations involved in computing education, but academia also harbored numerous pockets of resistance towards the academic independence

of computing, and the resistance came from both without and within. Even ACM president Anthony Oettinger wrote that, “departments of computer science have no place in the eternal scheme of things [...] it is an intellectual mistake to have departments of computer science” (Oettinger, 1968, pp. 27–28). Despite the opposition, academic computing programs mushroomed throughout the 1960s, and the first university departments for computing emerged in the early 1960s. Purdue University launched the first department called “computer science” in 1962. Degree programs were sufficiently diverse that there was little consensus regarding the field’s nomenclature, including its name (Tedre, 2014). The lack of consensus extended to the principles and nature of the computing discipline, so programs and course contents were typically determined by the computing facilities available (Gupta, 2007), personal agendas, research contracts, and collective interests.

Despite many educational initiatives at the grassroots level and on the broad front by associations that in many ways competed but that also complemented one another, the problems were many. Fein’s (1961) critical opinion was that the wider importance of the field was not well understood, the field was disunited, policies were underdeveloped, education and research were organized in an ad hoc manner, staff were often inexperienced, recruitment was haphazard, and the broader academic community did not consider a design- and application-oriented field to be an academic discipline in the classical sense. In addition, university administrations were sometimes indifferent and some faculty members were hostile (Fein, 1961, p.159). Before the 1960s there were concerns about computing pedagogy being underdeveloped (Perlis in Katz, 1960), books on computing were rare and textbooks virtually non-existent (Keenan, 1964; Cohen & Welch, 1999; p.145), and faculty availability was limited (Keenan, 1964). At the same time, the field’s struggles with disciplinary self-understanding hindered educational efforts (Keenan, 1964). The task of computing educators was often broadly conceived, from the intellectual base of computing to the uses of computing in other fields and the impact of computers on society (e.g. Piore, 1969).

Notwithstanding the lack of a disciplinary identity, articulations began to emerge of computing’s unique thought processes and disciplinary approaches to thought and practice. Ideas like those termed “algorithmizing” by Perlis in the late 1950s (cited in Forsythe, 1959; Katz, 1960) were in the subsequent decades called “algorithmic thinking” (e.g. Knuth, 1985; Statz & Miller, 1975) and “computational thinking” (e.g. Papert, 1980; Wing, 2006). Yet when visionaries such as Forsythe (1968) and Perlis (cited in Katz, 1960) championed algorithmic thinking for all fields of science, the time was not yet ripe for those ideas to enter the scientific mainstream; it took another twenty years and initiatives from people outside computing to appropriate and embrace the new computational modes of doing science (Denning and Martell, 2015). Computing education was also shaped by the emphasis placed by these

pioneers on algorithms and algorithmic thinking, and in particular by the growing understanding that algorithmic thinking, rather than coding, is the central skill in computing.

The role of programming in computing education has long been a source of major disputes, as programming did not fit well with the image of an aspiring academic discipline. In the 1950s and 1960s, programming – a “black art” led by a “priesthood” of programmers (Backus, 1980) – had a bad reputation in the academic world. That reputation was well founded in the anarchistic, “freewheeling” era (Backus, 1980), when programming was portrayed as an impenetrable craft requiring innate talent (Ensmenger, 2010). The reputation of the field of computing as “coding”, “hacking”, and “tinkering” haunted the professional and academic image of programming from early on (e.g. Bauer, 1975; Grosch, 1966), and several decades were spent in trying to elevate programming from a sloppy craft into a rigorous discipline (Dijkstra, 1980). The change was not always smooth: many computer gurus were proud of their craft, and in their opposition to more rigorous approaches, took it as a challenge to use their bag of tricks to create programs so complex that they could not be expressed using rigorous, structured notations (Hoare, 1996; Wirth, 2008). Despite repeated efforts to change the image of computing as programming, the image has persisted and has been documented in numerous studies over the years. It did not help that for years, computing educators continued to engage in disputes about better and worse programming languages, about programming techniques such as structured programming, and about better and worse programming paradigms for education.

In the early 1950s computing was often seen as number crunching, but by the early 1960s competing visions of computing had matured and there were strong arguments about where the field should be heading. On the computational science side, many educators emphasized the inclusion of “the gentle art of simulation” in the curriculum (Hamming in Katz, 1960). Electrical engineers, who considered themselves to be in the majority in ACM and in IEEE’s electronic computer group, wanted to improve the status of electrical engineering in computing degrees (Zadeh, 1965). Mathematically oriented pioneers advocated a theoretical and analytical approach to computing (Mahoney, 2011). But amidst the clashes between different branches of computing there was a strong feeling about the interdisciplinarity and breadth of the field. Gorn (1963) argued that computing combines principles from a large number of fields including mathematics, philosophy, linguistics, and library science. Education diversified similarly, and a 1963 panel discussion at the *ACM National Conference* divided computing education into five important categories, each with its own goals and standards: general education about computing, training of programmers, training of scientists in computing, education of computer specialists (systems specialists), and training of computing researchers (Keenan, 1964).

High-level computing curriculum discussions were also under way; for example, in 1960 many computing pioneers attended a conference that dealt with curriculum issues in the university context (Morse, 1960). The conference report posited computing as a discipline in its own right and proposed a number of subjects at both undergraduate and graduate levels. Yet a 1964 survey of computing curricula suggested that “practically everyone” who was then a computer professional had become one through apprenticeship or self-training (Atchison & Hamblen, 1964). The field’s technical slant was also criticized, and people voiced concerns over whether the field’s ability to build machinery had outstripped its ability to educate people who could put the machinery to intelligent use (Keenan, 1964). Another high-level conference on computing education was organized in 1967, with an impressive list of computing pioneers attending (Finerman, 1968a). That interest reflected the rapid growth of the number of computing majors in the U.S.: in 1964–1965 one survey found an estimated 4,338 undergraduate computing majors, mainly in the field of data processing, and the numbers were doubling roughly every fifteen months (Hamblen, 1969, 1972). Academic computing education was ripe for organized action.

Competing with organizations such as IEEE, DPMA, and IFIP, the ACM Curriculum Committee, formed in 1962, organized panel discussions at ACM National Conferences and worked on a definitive set of curriculum recommendations for academic computing (Gupta, 2007). The committee released the first draft of its curriculum recommendations in 1965 (Conte et al., 1965), and the final version of the ACM Curriculum was published in 1968 (Atchison et al., 1968). Although the curriculum document was followed by some passionate debates, Computing Curriculum (CC) ’68 became an authoritative guideline that played an important role in establishing computing education in the U.S.: it was visionary, well organized, and complete, and it encouraged the publication of textbooks for the courses it listed (Austing, Barnes, & Engel, 1977b; Gupta, 2007). In a survey four years later, nearly four in five colleges with majors in computing reported that the ACM Curriculum report had helped them to structure their computing education (Walker, 1973).

ACM’s Computing Curriculum ’68 was aimed explicitly at academic programs in computing and not at industrial training, so upon its publication it was immediately criticized for being too academic, too theoretical, and too narrowly focused. Its creators came mostly from a mathematical background (Atchison, 1981), which clearly showed in the end result. One critic complained that the purpose of computing education should not be to “turn out recursively new generations of academicians” (Wishner, 1968); another criticized Curriculum ’68 for involving too little laboratory work (Hamming, 1969); and computing education in general was accused of “deep nostalgia for mathematics”, where people would like to label themselves mathematicians but certainly were not (Hammer, 1970). Computing Curriculum ’68 also explicitly excluded all sociological,

economic, philosophical, and educational implications of computing as “not the exclusive nor even the major responsibility of computer science” (p. 155).

From the viewpoint of the software industry, which at the time was experiencing a self-proclaimed crisis, the 1968 curriculum tried to fix broken practice not by improving practice but by introducing more theory. The industry side demanded more practitioner orientation, more hands-on laboratory work, and more application areas of computing (Glaser, 1974; Hamming, 1969; Kandel, 1972; Spier, 1974). There was a consensus that computing education could not be based on tinkering and toying with programs, but although no alternative curriculum had demonstrated an educational high ground either, practically oriented players, such as businesses, often found the competing IEEE and DPMA curricula to be better suited for their needs (Ensmenger, 2001). All of this was taking place in the context of the continued desire of computing to be administratively independent of mathematics departments (Atchison, 1981).

The publication year of ACM Curriculum '68 also saw the birth of the software engineering movement. While the phrase “software engineering” was of older origin (e.g. Oettinger, 1967), a 1968 NATO-sponsored conference on the topic brought “software engineering” to everyone’s lips (Naur & Randell, 1969). That conference brought together people from diverse sectors in computing to discuss the problems in software production. Although the practical significance of that conference may be debated, it led many influential people to jointly declare that software production had serious problems that needed to be solved. It also set the “software engineering” bandwagon moving (Brennecke & Keil-Slawik, 1997) and created a sense of urgency and excitement about the possibility of modeling the field of computing after the more established engineering disciplines (Buxton & Randell, 1970; Tedre, 2014).

Following the 1968 curriculum, in 1969 the ACM founded a committee for computer science education, which later became the *ACM Special Interest Group for Computer Science Education* (SIGCSE), an important hub for those interested in computing education. The group’s newsletter *SIGCSE Bulletin* quickly became a central venue for airing ideas, course descriptions, news, and announcements of activities and events related to computing education in the context of traditional research universities. The first year’s debates were indicative of what was to come: their topics included the value of course descriptions in *SIGCSE Bulletin*, quality differences between institutions, theory versus practice, programming language choices, accreditation of programs, and the cost of computing education and facilities. However, although many discussants would have identified themselves both as computing educators and as computing researchers, their computing research was in various other fields of computing, not in “computing education research”.

4. Training for academia

The early 1970s saw the development of a number of communities of practice in computing education. Various computing curriculum recommendations had been proposed through the 1960s (including ACM's definitive curriculum in 1968), SIGCSE had begun in 1969, and textbooks were appearing at a rapid pace. The computing education community started by exchanging opinions and experiences, while research reports on computing education were more rare. Contributions to computing education in conferences, journals, and magazines were dominated by course and program descriptions and by reports of experiences, not only in computing departments but in other contexts: liberal arts colleges, developing countries, rural schools, small colleges, and non-computing disciplines. A number of other themes surfaced frequently, the most prominent being programming language debates, theory vs. practice, the role of mathematics, and how to teach programming. The SIGCSE symposium, which began in 1970, joined the SIGCSE bulletin in hosting discussions on programming paradigms, software engineering, structured programming, pedagogy, professionalism, law, and social relevance. Towards the end of the decade, presentations of new teaching tools and educational technology had increased in popularity.

One of the most prominent debates on computing education in the 1970s was about how to teach computing as an academically respectable, theoretically-based discipline. Those initiatives were certainly not new to computing; for example, in the mid-1950s Wesley Clark at Lincoln Lab provided a theoretically oriented course for engineers (De Mol, Bullynck, & Daylight, 2018), and ACM's Curriculum '68 was very academically oriented. In the 1970s debates often centered on the balance between theory and practice in computing education. In the theoretical-vs-practical debates, theory was often discussed almost synonymously with mathematics – sometimes even more so than with theoretical computer science – and practice synonymously with programming (Tedre, 2014). Numerous panel discussions, opinion pieces, and essays debated the role and relevance of mathematics in computing (e.g. Dijkstra, 1974; Egan, 1976; Forsythe, Galler, Hartmanis, Perlis, & Traub, 1970; Knuth, 1974; Pitts & Bateman, 1974).

Although many universities had established departments of computing, the field did not enjoy universal recognition in the academic world, and many saw mathematically based, rigorous foundations as the key to achieving recognition in academia. The first volume of Knuth's (1968) *The Art of Computer Programming* solidified the place of algorithms at the core of computing and placed programming on a mathematical foundation (Ceruzzi, 1988). The atmosphere of high-level educational discussions of the early 1970s was dominated by a late-1960s surge of techniques for proving program correctness, such as McCarthy's recursion induction, Floyd's and Naur's inductive assertion,

Burstall's structural induction, and Hoare's axioms and verification rules. Proponents of formal verification had little doubt about the intellectual supremacy of their approach and did not hesitate to speak out (Tedre, 2014). However, the reality of computing education programs in various universities, colleges, and institutes was different. Pressures of the 1960s and 1970s software crisis were growing, and many educators struggled to close the gap between academic education and the industrial reality of computing (Ensmenger, 2010).

Although education efforts were sensitive to the academic visions for mathematically based computing, they were also sensitive to the crisis of the software industry. Formal verification of computer programs was the pinnacle of formalizing and theorizing computing practice, while the newly coined term "software engineering" emphasized industry-oriented, practical, and engineering approaches to software construction (Naur & Randell, 1969). The two approaches, along with intentionally distorted images of both, became sticking points in computing education debates. On the one hand, unlike the technical visions of computing that were disliked by many university administrators, the mathematically- and theoretically-oriented visions of computing offered a good sales pitch for computing's disciplinary and administrative independence in traditional research universities (Tedre, 2014). There was a broadly shared feeling that if computing aspired to be a serious academic discipline, it needed to be built on a sound theoretical and mathematical basis (Tedre, 2014). On the other hand, software engineering provided a catchy phrase that drew a parallel between software construction and the solid safety record of fields such as civil engineering and mechanical engineering. There was a common feeling that both the computing industry and computing education would benefit from adopting practices and principles from the more established branches of engineering.

While the theory-oriented perspective of computing guided many academic institutions, there remained a wide gulf of rhetoric between the academic computing establishment and what took place in teaching practice and in the computing industry. Formal verification, while holding an intellectual high ground, was unable to provide solutions to the software crisis (MacKenzie, 2001), and the computing profession rapidly broadened from programmers, operators, technicians, and other professional computing personnel to include an increasing number of non-experts (Grudin, 1990). Programming work, once deemed a technical job that could be learnt in half a dozen lectures (Morse, 1960) over the space of a few weeks (Chinitz, 1955), expanded to cover all aspects of software construction while remaining a central skill for computing graduates. The software industry, whose needs for capable programmers were as high as ever, argued the need for programming to be taught as an applied topic rather than a theoretical one.

The increasing influence of software engineering led to discussions on its role in computing, but also influenced discussions on how to teach programming, teamwork, professional accreditation, human-computer interaction, and the psychology of programming. As with other engineering-oriented fields, accreditation of computing professionals continued to be a hot topic in education discussions. Structured programming, a concept developed in the late 1960s, was a common topic in the 1970s and occasionally as late as the 1980s, and was often discussed in relation to Dijkstra's (1968) opinion piece *Go To Statement Considered Harmful* – although the use of go to statements does not have a necessary relationship with structured programming. In addition to structured programming, the discussions about how programming should best be taught often concerned the choice of programming language.

The theory-vs-practice debates divided educators and researchers alike. Many theoretically-oriented people saw computing as little more than a numerical extension to mathematics and logic (e.g. Hoare, 1969), but many practicing programmers felt that the connection between programming competence and mathematical skills was weak (Glaser, 1974; Kandel, 1972; Spier, 1974) and that one could easily become an expert programmer without any knowledge of Turing machines, automata theory, or formal verification. The perceived need for mathematics in computing curricula slowly gravitated away from traditional mathematics topics towards discrete mathematics (Atchison, 1981).

Of course, neither side of the theoretical-vs-practical debate was unified in its views and intentions. On the theoretical side, views ranged from strong formal verificationism to seeing mathematics as the best tool for reasoning about computing (Tedre, 2014). On the practical side there were dividing lines between, for example, advocates of a management-oriented software engineering curriculum and advocates of a technically oriented professional software engineering curriculum (Tomayko, 1998).

Computing was also maturing as a discipline, keeping computing education in a state of constant flux. Many branches of computing built their own bodies of theories and first principles, which were increasingly incorporated into curricula and course contents. For instance, Denning (1972) proposed that some fields of computing had developed a sufficiently advanced intellectual core, independent of mathematics or other traditional disciplines, that they should be included in the core curriculum of computing in their own right. In addition, fueled by the continued problems in the software industry and inspired by the *Software Engineering Institute* (SEI), which was supported by the U.S. Department of Defense, the 1970s saw the advent of single engineering-type courses related to software and then, in the late 1970s, of whole programs with the same focus (Tomayko, 1998). There was also a gradual appropriation of concepts, as many mathematical concepts – such as graphs, queues, and matrices – were adopted into computing and could be learnt without a strong link to their mathematical counterparts.

Criticism of ACM's Curriculum '68 intensified over the 1970s. It was criticized for being driven by theoretically-oriented people, with only one committee member having a career in software (Tomayko, 1998). Furthermore, the broad spectrum of changes in the computing scene in the 1970s required new thinking about computing curricula, and in 1978 the ACM published a new curriculum. The 1978 curriculum report (Austing, Barnes, Bonnette, Engel, & Stokes, 1979) was a major effort, aimed at the whole academic field of computing and involving a large number of well-recognized individuals and institutions. As such, it carried weight and influence in directing the future of the whole discipline. Yet compared to the 1968 curriculum, which was received as a definitive description of course combinations that entitled a student to get a degree in computing, the objective of the 1978 curriculum was "to stimulate computer science educators to think about their programs" (Austing et al., 1979).

Most importantly, the 1978 curriculum presented a clear shift from a mathematically-oriented discipline to a more diverse discipline. Noticeably, compared with the 1968 curriculum, which was developed before the software crisis and the emergence of software engineering in 1968, the 1978 curriculum emphasized hands-on work, programming, and applications. While in the 1968 curriculum programming skill was a by-product of theoretical training (Gupta, 2007), the 1978 curriculum considered programming courses to "provide a philosophy of discipline which pervades all of the course work" (Austing et al., 1979).

The perceived role of mathematics changed greatly between the 1968 and 1978 curricula. The preliminary version of the 1978 curriculum made a radical statement that mathematics is not regarded as a prerequisite for any core material in computing (Austing, 1977a), but the final version contained a less controversial wording, noting the importance of mathematical techniques and concepts for a computer scientist. In just over ten years, computing in the ACM Curriculum had transformed from a theoretical, mathematically based discipline into a programming- and application-centered discipline. However, the mathematically-oriented educators were not yet ready to give up, and the tone for computing education debates was set for another half decade.

The 1978 curriculum was immediately criticized for equating the field of computing with programming (Ralston & Shaw, 1980), and many wanted to see mathematics, logic, and formal verification take a bigger role in computing education. Ralston (1984) claimed that among computing researchers there was a consensus about the centrality of mathematics in computing and about its insufficient emphasis in the 1978 curriculum. Walters (1988) wrote that students became painfully aware that computing required a mathematical background. Krantz (1984) wrote that unlike mathematics, computing in academia was mostly trivial – all hardware and few ideas – and despite computing being "sexy stuff these days", students aspiring to a computing career should study mathematics instead. Similarly, Walters (1988) argued that computing is

mathematical and in dire need of accreditation for industry's purposes. A series of accreditation efforts by a variety of organizations were a source of constant debate and critique (Cannon et al., 1986; Gibbs, 1985; Gibbs & Tucker, 1986; Impagliazzo, 1994).

The 1978 curriculum was also criticized for being poorly suited to small liberal arts colleges with limited faculty and computer resources (Bruce, Cupper, & Drysdale, 2010; Gibbs & Tucker, 1986), prompting a number of computing educators in liberal arts colleges to propose alternative computing curriculum recommendations better suited for their own purposes (Bruce et al., 2010; Walker, 2018). The Liberal Arts Computer Science Consortium (LACS), founded in 1984, produced its first curriculum report in 1986 (Gibbs & Tucker, 1986). In addition to resource concerns, the LACS curriculum followed a philosophy different from that of the ACM Curriculum: it de-emphasized what the LACS consortium perceived as "engineering-style education" in CC'78 and put more emphasis on mathematical and logical analysis of computing topics (Bruce et al., 2010). The consortium criticized computing for being "the only science whose applications comprise the core curriculum, and whose principles and theories are left for the senior electives" (Gibbs & Tucker, 1986) and advanced a more theoretical and mathematical approach than the programming and applications-oriented ACM CC'78 (Gibbs & Tucker, 1986; Walker, 2018). The next ACM Curriculum in 1991 (Turner, 1991) involved LACS members in the committee, and proposed several sample liberal arts curricula in computing. Again, a few years later the LACS consortium presented its own recommendations with a more theoretical orientation and a number of recommendations, such as to change programming paradigms early in order to expose students to different ways of thinking about the programming task (Walker & Schneider, 1996). Similarly, the 2001 ACM Curriculum was met with another 2007 LACS model curriculum (Bruce et al., 2010).

The 1980s continued with debates about the advantages and disadvantages of different programming languages for teaching programming as well as debates over different programming paradigms for teaching programming. Technological development was increasingly visible in computing education discussions, most prominently in the use of tools such as simulators and programming environments, but also in the use of microcomputers or personal computers in education. The community's publishing venues were filled with course descriptions and reports of experiences, but whereas a 1976 survey of education literature (Austing et al., 1977b) found almost exclusively non-research papers (surveys of programs, tool papers, activity reports by professional organizations, philosophies of programs, and program and course descriptions), from the early 1980s a growing number of empirical research papers started to appear in conferences and magazines. However, recent work on the development of computing education research suggests that it still took over 20 more

years for the field to mature as a primarily empirical research field (Simon, 2015).

Conference proceedings published by organizations such as ACM, AFIPS, IFIP, and IEEE provided a central venue for disseminating information on computing education from early on (Austing et al., 1977b). Computing education papers were also occasionally published in other venues such as engineering education journals, generic computing journals, educational technology journals, and educational journals, such as *IEEE Transactions on Education* and *Communications of the ACM*. But aside from *SIGCSE Bulletin*, periodicals devoted to computing education were slow to appear, computing education research journals are still not many, and their focus is often the development of tools or the use of computers in the classroom rather than empirical research (Joy, Sinclair, Sun, Sitthiworachart, & López-González, 2009). Computing education researchers had to wait until 1991 for computing education to be added as a field in ACM's computing research classification system.

As the number of computer users grew rapidly, diminishing proportions of those users were engineers or computer specialists, and usability concerns were growing fast. At the time of the CC'68 report, human-computer interaction (HCI, or CHI) was not yet a research field of its own, but there was growing research on the psychology of computer programming, human-machine systems, computer graphics, and programmer-computer interaction (Baecker, Grudin, Buxton, & Greenberg, 1995; Grudin, 1990). In the 1970s large organizations started to concentrate efforts on usability, and in the 1980s there was a significant industry representation in the nascent academic HCI community (Grudin, 1991). Grudin (1991) pointed out that user needs received surprisingly little emphasis until the 1980s. Accordingly, HCI was included in the curricular recommendations relatively late: for example, CC'78 included HCI as a minor aspect of social "implications".

Participation in computing broadened at a rapid pace. Computing education was the "in" thing, it was supported by UNESCO, EEC, and other major players, and it was gradually moving into various levels of school education (Atchison, 1981; Guzdial, 2015). Although many initiatives primarily involved computer literacy (Atchison, 1981; Guzdial, 2015; Tedre & Denning, 2016), there were also epistemologically and pedagogically radical ideas, such as Papert's (1980) and Beynon and Russ's empiricist views (Tedre & Denning, 2016). However, teacher training was an obstacle and there was a dire lack of computer- and computing-literate teachers in primary and secondary schools (Atchison, 1981).

The broadening participation in computing was also visible in topics that emerged in conferences and newsletters. Computing jobs became gender polarized by the early 1960s (Ensmenger, 2010), leading to increasing concern about gender issues in computing education in the 1980s. The 1970s also saw

a rise of concern about developing countries and ethical questions, which influenced the already vivid discussions of computing in other fields of research. In response to the active discussions on ethical questions in computing's major publications, CC'78 suggested that social, philosophical, and ethical considerations must "permeate the instruction", and strongly recommended an elective course in Computers and Society despite objections that this topic lay outside the field of computing (Austing et al., 1979, p. 155).

Initiatives for introducing computing's unique ways of thinking and practice in K-12 education received a major push from Papert's (1980) book *Mindstorms*, which crystallized over a decade's worth of research, advocating an empiricist approach to knowledge construction using computers and the LOGO language. Papert's work on computers and education had started in the late 1960s, and contributed to a number of influential ideas through the following decades, such as Kay's (1972) *Dynabook*, Solomon's (1986) work on computers and learning, and Abelson and diSessa's (1980) *Turtle Geometry* as well as their *Boxer* programming environment (diSessa & Abelson 1986). *Mindstorms* was groundbreaking and comprehensive, it was about how children learn, it argued for a radical epistemological shift, and its pedagogical influence spread well beyond the computing communities.

While formal verification and mathematics played a central role in high-level discussions on the formation of a computing identity in the 1960s and 1970s, in the 1980s formal verification was rapidly losing its perceived importance in computing education. Its real role in computing education and computing industry was minuscule compared to the mathematical visions conjured by prominent members of the formal verification community (Tedre, 2014). The '78 curriculum reflected a general attitude change towards strong versions of formal verificationism, but the broader computing narrative would have to wait almost ten years more for a major shift in perceptions of the role of formal verification in computing (Tedre, 2014).

Computing education discussions at the beginning of the 1980s saw another influential initiative that would come to characterize the field for years to come. A number of leading computer scientists and U.S. government officials published a number of reports that raised alarms about a looming crisis in computing departments, involving faculty shortages, obsolescent facilities, low salaries, and heavy teaching burdens (Denning, 1981; Denning et al., 1981; Feldman & Sutherland, 1979). There was a brain drain of competent teachers and researchers, who could design and build computing systems, from academia to industry (Tedre & Denning, 2016). Reports called for increased investment in "experimental computer science", and although increased NSF funding programs helped to overcome the crises with workforce and facilities, there was never a consensus as to what exactly experimental computer science was: the term meant different things to computer scientists from different traditions (Tedre & Moisseinen, 2014).

5. Training for computational problem-solving

A decade further on, designers of the next computing curriculum were faced with an even more diverse discipline. In 1984, wanting the new curriculum update to reflect the dramatic changes in computing and its pedagogy, the ACM Education Board setup two task forces that produced a pair of documents that greatly influenced perceptions of computing as an academic discipline. The first, *Computing as a Discipline* (Denning et al., 1989), described computing as an irrevocably intertwined combination of three traditions: mathematics, engineering, and science. The second (Turner, 1991) joined two large computing associations, IEEE-CS and ACM, behind a single, unified set of curriculum recommendations with many implementation examples that followed the breadth-first approach of the *Computing as a Discipline* report.

While the previous computing curricula had had a transformational effect on computing education, the 1991 curriculum recommendations did not trigger major changes in computing education (Bruce, 1996; Goldweber et al., 1997; Joint Task Force on Computing Curricula, 2001). On the other hand, its sister document, Denning et al.'s (1989) *Computing as a Discipline* report, resonated with the way in which many people saw their field, making it a widely read and oft-quoted document in computing's disciplinary and education discussions for years to come. While "computer science" was the phrase of choice until the 1990s, starting from the *Computing as a Discipline* document the phrase began to be replaced in an increasing number of contexts with phrases such as "computing", "the discipline of computing", and "the field of computing".

Problem-solving was a central concept in computing education and curricula from very early on. Each iteration of the ACM Curriculum paid more attention to problem-solving in computing education (Atchison et al., 1968; Austing et al., 1979; Turner, 1991; Joint Task Force on Computing Curricula, 2001; Joint Task Force on Computing Curricula, Association for Computing Machinery (ACM) and IEEE Computer Society, 2013). Pedagogically, computing education began to appreciate the teaching of problem-solving strategies and processes and to avoid fixed problem-statements (Goldweber et al., 1997; Sutinen & Tarhio, 2001). From the 1960s, scientists from various fields had considered computers as problem-solving environments for their fields of science as well as for science education (Gallopoulos, Houstis, & Rice, 1994). People were now increasingly seeing computing as a generic problem-solving activity (Tedre & Denning, 2016). Computational ways of thinking and practice started to be viewed as focused predominantly on problem-solving, on formulating problems so that their solutions could be represented in an algorithmic or computational form (Aho, 2011; Wing, 2006). Those views were developing alongside paradigmatic changes in science in general, where computing had started to change the very structure of the natural sciences.

Until the 1980s many scientists had treated computers as very fast calculators that could help them to solve their equations and to analyze large data sets, but the 1980s marked a shift in the epistemology and methodology of science (Tedre & Denning, 2017). Field by field, scientists started to re-interpret their subjects of study as information processes, to explore natural phenomena by simulating them, and to track the behavior of systems for which no analytical models were known. Interpreting natural processes as information processes and simulating those processes with the computer became a “third pillar of science” alongside the traditional theory and experiment (Benioff & Lazowska, 2005). Dodig-Crnkovic and Müller (2011) called this the “info-computational” turn of science. For instance, the biologist David Baltimore started to treat biological processes as information processes amenable to computer simulation, marking a dramatic change in the way biological processes were interpreted and studied (Baltimore, 2002). In the U.S., the “Grand Challenges” policy guaranteed substantial funding for computational sciences and high-performance computing from the late 1980s onward.

The shift in the epistemology and methodology of science spread quickly (Tedre & Denning, 2017); it was claimed to be “the most disruptive paradigm shift in the sciences since quantum mechanics” (Chazelle, 2006) and the new millennium was pronounced “the age of computer simulation” (Winsberg, 2010). This shift necessarily cast the field of computing in a new light. Computing was no longer perceived as a science of the artificial; it had become a study of information processes both artificial and natural (Denning, 2007). However, while the 1991 ACM/IEEE Curriculum did extend its recommendations to other academic contexts, such as liberal arts, natural sciences, and engineering, the ramifications of the info-computational turn of science were recognized only weakly in CC’91 (Turner, 1991).

Among scientifically oriented computing people there grew a feeling that computational science in general, and computational science education in particular, did not fit well under the prevailing umbrella of computing as a discipline (Stevenson, 1993). Computing education was perceived as lacking scientific rigor, providing too little mathematics, of the wrong kind, and preparing students poorly for real-life applications and the uses of computing in science and engineering (Stevenson, 1993). For a while there was a feeling that computational science would split from computing as a discipline, but when funders began to expect tighter integration of core computing into research projects in other sciences, researchers followed the money (Tedre, 2014).

The 1991 curriculum did promote diversification in terms of academic contexts, discussing pedagogical aspects, communication skills, legal and ethical issues, and social and professional questions, although rather superficially (Turner, 1991). However, computing was diversifying at a record pace, and in increasingly divergent branches of computing, such as computability theory, artificial intelligence, and human-computer interaction, it was becoming more

difficult to specify a common core of study. In the late 1990s a diversity of curricula were developed by various organizations, and ACM and IEEE were working on several computing curricula. The continuing diversification left the 1991 curriculum as the last effort to combine all of computing under a single, unified set of curriculum recommendations. In the 2000s the new ACM/IEEE computing curricula split the field into five computing disciplines: computer engineering (CE2004), computer science (CS2008), software engineering (GSWE2009), information technology (IT2008), and information systems (IS2010) (Shackelford et al., 2005, p. 12). The ideal was no longer a one-size-fits-all curriculum; instead, different educational programs could access many alternative guidelines and model their curricula on those that best suited their purposes.

By the 2000s computing education was also becoming established as a research discipline, as attested by many measures (Simon, 2015). The SIGCSE conference was joined by other computing education conferences: ITiCSE (Innovation and Technology in Computer Science Education) in 1996, ACE (Australasian Computing Education) in 1996, Koli Calling in 2001, and ICER (International Computing Education Research) in 2005. All of these conferences showed increasing proportions of computing education research publications, and ICER was created as a forum solely for computing education research. Computing education research was increasingly carried out with reference to educational research, using pedagogical approaches, theories, methods, and results from the field of education (Simon, 2015). Computing education research became the subject of dedicated professorships, research groups, research training, journals, and other markers of a mature, living research field.

6. Conclusions

As technological, theoretical, and scientific knowledge about computing increased, computing education expanded its coverage to incorporate new and emerging areas. Today the old computing curricula serve as a record of the unprecedented growth and development of computing as a discipline. With the development of computing as an independent discipline, computing education expanded outward, from technical training given predominantly by companies for their own purposes to a programming- and practice-oriented activity that aimed at producing workforce for the industry as well as a theoretically oriented activity where mathematics played an important role, becoming a truly interdisciplinary undertaking that respected and tried to adapt to the unique areas of specialization in computing. Each decade saw trends wax and wane along with new attempts to encapsulate the soul of the discipline. Table 1 presents examples of the expanding focus of computing education over a number of decades.

Table 1. Computing Education Expanding Outward.

Theme	Training for technical jobs	Training for software development	Training for academia	Training for computational problem-solving
Focal Period	1950s	1960s–1980s	1970s–1980s	1990s–2000s
Focus	Coping with technology	Programmer productivity	Theoretical sophistication	Design, application in problem domains
Subject matter	The computer	Programming	Algorithms	Information processes
Educators	Companies	Computing centers	Computing departments	All departments
Curriculum function	Local	Prescriptive	Descriptive	Indicative
Skills and knowledge	Technical	Applied	Theoretical	Design, social
Characterizing debate	Accreditation of professionals	Structured programming, software crisis	Formal verification, experimental CS	Computational science, computational thinking

Each column in [Table 1](#) refers to a rough period of change, divided into the emergence of computing education (1950s), its organization and standardization (1960s–1980s), maturing (1970s–1980s), and diversification (1990s–2000s). Rows in [Table 1](#) describe different aspects of computing education: for example, the “subject matter” row describes how the 1950s commonly saw computer technology as the subject of study in computing education; the 1960s expanded the subject matter to focus increasingly on programming; in the 1970s there arose a broad agreement that computing education was concerned primarily with algorithms; and in the 1990s it became commonplace to say that it encompasses information processes of all kinds. In reality each aspect of the subject matter has been a part of computing all along – for example, theory has surely been a part of computing education from the beginning – so the table should be read as a characterization of changes of priority and expansion of focus rather than clear-cut, mutually exclusive time periods.

Four clear trends can be identified in the development of computing education over the years. Firstly, over the years the focus of computing education has expanded, continually reaching further away from the machine. The early academics and professionals needed detailed knowledge about hardware technology, but the rising level of abstraction and theory progressively extended computing education to concerns of the programming process, theory of computing, contexts of computing, and organizational, social, and cultural aspects. As the discipline, computing technology, and theory expanded to cover more and more ground, education naturally followed to the new areas. In the same vein, provision of education diversified from computer manufacturers and software producers to computing centers, computing departments, and then to various other departments involved in computational sciences.

Secondly, curriculum functions have gradually changed in terms of abstraction and prescriptiveness. Bringing an end to the sprawl of local curricula based on local needs and on the competencies and interests of teaching staff, ACM’s 1968 curriculum was able to unite the academic computing community by prescribing what a duly recognized academic

education in computing should teach. The 1978 curriculum started to become more descriptive (Goldweber et al., 1997), as its explicit objective was to “stimulate computer science educators to think about their programs”. The 1991 curriculum was the last attempt to provide a single set of curriculum recommendations for all of academic computing. It suggested knowledge units but no longer specific course designs. Finally the computing curricula reached a point where they could no longer encompass the accumulating additions, and the new curriculum recommendations in the 2000s by ACM and IEEE began with the principle that one single set of recommendations is not enough for computing. Today the ACM/IEEE curricula are divided into many, and a large number of organizations continue to propose numerous more specialized curriculum recommendations.

Thirdly, a number of debates have remained sticking points throughout the history of the discipline. “Language wars” over programming language choice for education have been a bone of contention ever since the days of Fortran and Cobol, along with related “paradigm wars” over procedural, functional, and object-oriented approaches. The balance between theory and practice has manifested in a large number of debates on topics such as the balance of mathematics and programming, abstract and concrete, form and content, computing and the computer, academia and industry, algorithms and programs, science and technology, general and particular, and so forth. Some concerns have been almost constantly on the agenda: for example, apart from the first decade, gender disparity has been a continuing concern and discussion topic of the computing education community, as has how to expand computing education to schools, continuing education, and non-CS majors (Guzdial, 2015).

Fourthly, computing education has gradually broadened its use of reference disciplines. The technically focused and application-driven early local curricula and ACM’s 1968 curriculum were relatively narrow in their reference disciplines: CC’68 included a great deal of mathematics, and discussed physics and engineering, although as possible minors it mentioned arts, humanities, and social sciences because of the potential benefit of computing to those fields. In subsequent years it became obvious that the many branches of computing all require different combinations of minor studies to support them. For example, human-centered fields would benefit from studies in psychology and cognitive science, and information systems from studies in organizational psychology, business, and management.

Over the years computing education has had a multifaceted relationship with the discipline of computing. Its primary overt function has been educational transfer: to facilitate the learning of what the computing community believes fresh graduates should know about computing. In the early years, when the disciplinary identity of computing was not yet clearly established, it also served

the important academic purpose of building and strengthening the field's image in academia. Computing education also contributed to building that very identity, and computing education and computing curricula were at times used as an arena for debates between strong ideological movements to define computing as a discipline. A discipline based on formal verification of programs would surely look very different from a discipline based on software engineering, and the two would have different emphases on the use of funds, student and staff selection processes, relationship to other stakeholders, and administrative structures.

Note

1. Wilkes et al.'s (1951) book is available online at <https://archive.org/details/programsforelect00wilk>.

Acknowledgments

The authors would like to thank Peter J. Denning and the participants of the Siegen University Workshop "Beyond ENIAC: Early Digital Platforms & Practices", June 10–12, 2016, for conversations and comments that greatly improved this article.

Disclosure statement

No potential conflict of interest was reported by the authors.

Notes on contributors

Matti Tedre is a professor at the School of Computing, University of Eastern Finland, and the author of *The Science of Computing: Shaping a Discipline* (CRC Press, 2014). His research focuses on the philosophy of computer science, educational technology, computer science education research, and the disciplinary history of computing.

Simon is a senior lecturer in computing at the University of Newcastle in Australia. His research in computing education focuses on academic integrity in computing and on the development of computing education research as a discipline, which was the subject of his recent doctoral dissertation.

Lauri Malmi is a professor at Department of Computer Science, Aalto University, Finland, where he is leading the Learning + Technology (LeTech) research group. His research focuses on development and evaluation of advanced software tools to support programming education. He is a regular columnist in ACM Inroads magazine and an associate editor in IEEE Transactions on Learning Technologies.

ORCID

Matti Tedre <http://orcid.org/0000-0003-1037-3313>

Lauri Malmi <http://orcid.org/0000-0003-1064-796X>

References

- Abelson, H., & diSessa, A. A. (1980). *Turtle geometry: The computer as a medium for exploring mathematics*. Cambridge, MA, USA: MIT Press.
- Aho, A. V. (2011, January). Ubiquity symposium: Computation and computational thinking. *Ubiquity*, 2011, 1.
- Ashmore, E. (1985). *An interview with Willis Daniel*. University of Minnesota, Minneapolis, USA: Charles Babbage Institute, The Center for the History of Information Processing.
- Aspray, W. (2000). Was early entry a competitive advantage? US universities that entered computing in the 1940s. *IEEE Annals of the History of Computing*, 22(3), 42–87.
- Atchison, W. F. (1981). Computer education, past, present, and future. *ACM SIGCSE Bulletin*, 13(4), 2–6.
- Atchison, W. F., & Hamblen, J. W. (1964). Status of computer sciences curricula in colleges and universities. *Communications of the ACM*, 7(4), 225–227.
- Atchison, W. F., Schweppe, E. J., Viavant, J. W., Young, D. M., Conte, S. D., Hamblen, J. W., ... Rheinboldt, W. C. (1968). Curriculum 68: Recommendations for academic programs in computer science: A report of the ACM curriculum committee on computer science. *Communications of the ACM*, 11(3), 151–197.
- Austing, R. H., Barnes, B. H., Bonnette, D. T., Engel, G. L., & Stokes, G. (1977a). Curriculum recommendations for the undergraduate program in computer science: A working report of the ACM committee on curriculum in computer sciences. *ACM SIGCSE Bulletin*, 9(2), 1–16.
- Austing, R. H., Barnes, B. H., Bonnette, D. T., Engel, G. L., & Stokes, G. (1979). Curriculum '78: Recommendations for the undergraduate program in computer science— A report of the ACM curriculum committee on computer science. *Communications of the ACM*, 22(3), 147–166.
- Austing, R. H., Barnes, B. H., & Engel, G. L. (1977b). A survey of the literature in computer science education since curriculum '68. *Communications of the ACM*, 20(1), 13–21.
- Backus, J. (1980). Programming in America in the 1950s—Some personal impressions. In N. Metropolis, J. Howlett, & G.-C. Rota (editors), *A history of computing in the twentieth century* (pages 125–135). New York, NY, USA: Academic Press.
- Baecker, R. M., Grudin, J., Buxton, W. A., & Greenberg, S. (editors). (1995). *Readings in human-computer interaction: Toward the year 2000* (2nd edition). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.
- Baltimore, D. (2002). How biology became an information science. In P. J. Denning (editor), *The Invisible Future* (pages 43–55). New York, NY, USA: McGraw-Hill.
- Bauer, F. L. (1975). Software engineering. In F. L. Bauer (editor), *Software engineering: An advanced course, volume 30 of lecture notes in computer science* (pages 522–545). Berlin/Heidelberg, Germany: Springer-Verlag.
- Benioff, M. R., & Lazowska, E. D. (editors). (2005). *Computational science: Ensuring America's competitiveness*. Washington, DC, USA: President's Information Technology Advisory Committee.
- Bowden, B. V. (1953). A brief history of computation. In B. V. Bowden (editor), *Faster than thought: A symposium on digital computing machines* (pages 3–32). London, UK: Sir Isaac Pitman & Sons.
- Brennecke, A., & Keil-Slawik, R. (editors). (1997). *Position papers for Dagstuhl seminar 9635 on history of software engineering*. Paderborn, Germany: Dagstuhl Seminar Series.
- Bruce, K. (1996). Thoughts on computer science education. *ACM Computing Surveys*, 28(4es), 93-es.

- Bruce, K. B., Cupper, R. D., & Drysdale, R. L. S. (2010). A history of the liberal arts computer science consortium and its model curricula. *ACM Transactions on Computing Education*, 10(1), 1–12.
- Buxton, J. N., & Randell, B., editors (1970). *Software Engineering Techniques: Report on a Conference Sponsored by the NATO Science Committee*. NATO Science Committee, Brussels, Belgium.
- Campbell-Kelly, M. (2003). *From airline reservations to sonic the hedgehog: A history of the software industry*. Cambridge, MA, USA: MIT Press.
- Cannon, R. L., Cupper, R. D., Dalphin, J. F., Scragg, G. W., Sjoerdsma, T., & Turner, A. J. (1986). Computer science accreditation and liberal arts programs. In G. L. Engel (editor), *Proceedings of the 1986 ACM Fourteenth Annual Conference on Computer Science*, SC '86, (pages 399–402). New York, NY, USA: ACM.
- Ceruzzi, P. (1988). Electronics technology and computer science, 1940–1975: A coevolution. *IEEE Annals of the History of Computing*, 10(4), 257–275.
- Ceruzzi, P. E. (2003). *A History of Modern Computing*. (2nd ed.). Cambridge, MA, USA: MIT Press.
- Chazelle, B. (2006). Could your iPod be holding the greatest mystery in modern science? *Mathematical Horizons*, 13(4), 14–31.
- Chinitz, M. P. (1955). Contributions of industrial training courses in computers. In A. W. Jacobson (editor), *Proceedings of the First Conference on Training Personnel for the Computing Machine Field*. (pages 29–32) Detroit, MI, USA: Wayne University Press.
- Cohen, I. B., & Welch, G. W. (editors). (1999). *Makin' numbers: Howard Aiken and the computer*. Cambridge, MA, USA: MIT Press.
- Conte, S. D., Hamblen, J. W., Kehl, W. B., Navarro, S. O., Rheinboldt, W. C., Young, J., D. M., & Atchinson, W. F. (1965). An undergraduate program in computer science—Preliminary recommendations. *Communications of the ACM*, 8(9), 543–552.
- Daylight, E. G. (2014). A Turing tale. *Communications of the ACM*, 57(10), 36–38.
- Daylight, E. G. (2016). *Turing Tales*. Belgium: Lonely Scholar.
- diSessa, A. A., & Abelson, H. (1986). Boxer: A reconstructible computational medium. *Communications of the ACM*, 29(9), 859–868.
- De Mol, L., Bullynck, M., & Daylight, E. G. (2018). *Less is more in the fifties: Encounters between logical minimalism and computer design during the 1950s*. Article in Press.
- Denning, P. J. (1972). Operating systems principles and undergraduate computer science curricula. *SIGMIS Database*, 4(2), 5–10.
- Denning, P. J. (1981). ACM president's letter: Eating our seed corn. *Communications of the ACM*, 24(6), 341–343.
- Denning, P. J. (2007). Computing is a natural science. *Communications of the ACM*, 50(7), 13–18.
- Denning, P. J., Comer, D. E., Gries, D., Mulder, M. C., Tucker, A., Turner, A. J., & Young, P. R. (1989). Computing as a discipline. *Communications of the ACM*, 32(1), 9–23.
- Denning, P. J., Feigenbaum, E., Gilmore, P., Hearn, A., Ritchie, R. W., & Traub, J. (1981). A discipline in crisis. *Communications of the ACM*, 24(6), 370–374.
- Denning, P. J. and Martell, C. H. (2015). *Great principles of computing*. Cambridge, MA, USA: MIT Press.
- Dijkstra, E. W. (1968). Letters to the editor: Go to statement considered harmful. *Communications of the ACM*, 11(3), 147–148.
- Dijkstra, E. W. (1974). Programming as a discipline of mathematical nature. *The American Mathematical Monthly*, 81(6), 608–612.
- Dijkstra, E. W. (1980). A programmer's early memories. In N. Metropolis, J. Howlett, & G.-C. Rota (editors), *A history of computing in the twentieth century* (pages 563–573). New York, NY, USA: Academic Press.

- Dodig-Crnkovic, G., & Müller, V. C. (2011). A dialogue concerning two world systems: Info-computational vs. mechanistic. In G. Dodig-Crnkovic & M. Burgin (editors), *Information and computation: Essays on scientific and philosophical understanding of foundations of information and computation, volume 2 of World Scientific Series in Information Studies*. Singapore: World Scientific.
- Editors of DATA-LINK. (1958). What's in a name?. *Communications of the ACM*, 1(4), 6.
- Egan, L. G. (1976). Closing the "gap" between the university and industry in computer science. *ACM SIGCSE Bulletin*, 8(4), 19–25.
- Ensmenger, N. L. (2001). The 'question of professionalism' in the computer fields. *IEEE Annals of the History of Computing*, 23(4), 56–74.
- Ensmenger, N. L. (2010). *The computer boys take over: Computers, programmers, and the politics of technical expertise*. Cambridge, MA, USA: MIT Press.
- Fein, L. (1959). The role of the university in computers, data processing, and related fields. *Communications of the ACM*, 2(9), 7–14.
- Fein, L. (1961). The computer-related sciences (synnoetics) at a university in the year 1975. *American Scientist*, 49(2), 149–168.
- Feldman, J. A., & Sutherland, W. R. (1979). Rejuvenating experimental computer science: A report to the National Science Foundation and others. *Communications of the ACM*, 22(9), 497–502.
- Finerman, A. (editor). (1968a). *University education in computing science*. New York, NY, USA: Academic Press.
- Finerman, A. (1968b). University education in computing science: Introduction. In A. Finerman (editor), *University education in computing science* (pages 1–4). New York, NY, USA: Academic Press.
- Forsythe, G. E. (1959). The role of numerical analysis in an undergraduate program. *The American Mathematical Monthly*, 66(8), 651–662.
- Forsythe, G. E. (1968). What to do till the computer scientist comes. *The American Mathematical Monthly*, 75(5), 454–462.
- Forsythe, G. E., Galler, B. A., Hartmanis, J., Perlis, A. J., & Traub, J. F. (1970). Computer science and mathematics. *SIGCSE Bulletin*, 2(4), 19–29.
- Galler, B. A. (1986). The IBM 650 and the universities. *IEEE Annals of the History of Computing*, 8(1), 36–38.
- Gallopoloulos, E., Houstis, E., & Rice, J. R. (1994). Computer as thinker/doer: Problem-solving environments for computational science. *IEEE Computational Science and Engineering*, 1(2), 11–23.
- Gibbs, N. E. (1985). The LameDuck SIG chairman's message: A parting shot at accreditation. *ACM SIGCSE Bulletin*, 17(3), 1–2.
- Gibbs, N. E., & Tucker, A. B. (1986). A model curriculum for a liberal arts degree in computer science. *Communications of the ACM*, 29(3), 202–210.
- Glaser, G. (1974). Education 'inadequate' for business DP. *Computerworld*, VIII(45), 1–2.
- Goldweber, M., Impagliazzo, J., Bogoiavlenski, I. A., Clear, A. G., Davies, G., Flack, H., ... Rasala, R. (1997). Historical perspectives on the computing curriculum. *SIGCUE Outlook*, 25(4), 94–111.
- Gorn, S. (1963). The computer and information sciences: A new basic discipline. *SIAM Review*, 5(2), 150–155.
- Grier, D. A. (2005). *When computers were human*. Princeton, NJ, USA: Princeton University Press.
- Grosch, H. (1966). Programmers: The industry's cosa nostra. *Datamation*, 12(10), 202.
- Grudin, J. (1990). The computer reaches out: The historical continuity of interface design. In *CHI '90: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 261–268, New York, NY, USA. ACM.

- Grudin, J. (1991). Interactive systems: Bridging the gaps between developers and users. *IEEE Computer*, 24(4), 59–69.
- Gupta, G. K. (2007). Computer science curriculum developments in the 1960s. *IEEE Annals of the History of Computing*, 29(2), 40–54.
- Guzdial, M. (2015). *Learner-centered design of computing education: research on computing for everyone..* Synthesis Lectures on Human-Centered Informatics San Rafael, CA, USA: Morgan & Claypool.
- Hamblen, J. W. (1969). Computer science and related degree programs in U.S. higher education. *ACM SIGCSE Bulletin*, 1(4), 9–13.
- Hamblen, J. W. (1972). Production and utilization of computer manpower in U.S. higher education. In *Proceedings of the May 16–18, 1972, Spring Joint Computer Conference, AFIPS '72 (Spring)* (pages 627–632). Atlantic City, NJ, USA: American Federation of Information Processing Societies, Inc.
- Hammer, P. C. (1970). Computer science and mathematics. In B. Scheepmaker & K. L. Zinn (editors), *Papers of the first IFIP world conference on computer education* (pages 1/65–67). Amsterdam, The Netherlands: International Federation for Information Processing.
- Hamming, R. W. (1969). One man's view of computer science. *Journal of the ACM*, 16(1), 3–12.
- Hoare, C. A. R. (1969). An axiomatic basis for computer programming. *Communications of the ACM*, 12(10), 576–580.
- Hoare, C. A. R. (1996). How did software get so reliable without proof?. In M.-C. Gaudel & J. Woodcock (editors), *FME'96: Industrial benefit and advances in formal methods, volume 1051 of lecture notes in computer science* (pages 1–17). Heidelberg, Germany: Springer.
- Hohn, F. (1955). Review: The first conference on training personnel for the computing machine field. *The American Mathematical Monthly*, 62(1), 8–15.
- Impagliazzo, J. (1994). An industry perspective on computer science accreditation (abstract). In J. Werth & L. Werth (editors), *Proceedings of the 22nd Annual ACM Computer Science Conference, CSC '94* (pages 435). New York, NY, USA: ACM.
- Jacobson, A. W. (editor). (1955). *Proceedings of the First Conference on Training Personnel for the Computing Machine Field*. Detroit, MI, USA: Wayne University Press.
- Joint Task Force on Computing Curricula. (2001). *Computing curricula 2001. Computer science*. New York, NY, USA: ACM and IEEE-CS.
- Joint Task Force on Computing Curricula, Association for Computing Machinery (ACM) and IEEE Computer Society. (2013). *Computer science curricula 2013: Curriculum guidelines for undergraduate degree programs in computer science*. New York, NY, USA: ACM Press.
- Joy, M., Sinclair, J., Sun, S., Sitthiworachart, J., & López-González, J. (2009). Categorising computer science education research. *Education and Information Technologies*, 14(2), 105–126.
- Kandel, A. (1972). Computer science—A vicious circle. *Communications of the ACM*, 15(6), 470–471.
- Katz, D. L. (1960). Conference report on the use of computers in engineering classroom instruction. *Communications of the ACM*, 3(10), 522–527.
- Kay, A. C. (1972). *A personal computer for children of all ages*. Technical memo. Palo Alto, CA, USA: Xerox Palo Alto Research Center.
- Keenan, T. A. (1964). Computers and education. *Communications of the ACM*, 7(4), 205–209.
- Kircher, P. (1955). Review: Proceedings of the first conference on training personnel for the computing machine field. *The Accounting Review*, 30(4), 725–726.
- Knuth, D. E. (1968). *The art of computer programming, volume 1* (1st edition). Reading, MA, USA: Fundamental Algorithms. Addison-Wesley.
- Knuth, D. E. (1974). Computer science and its relation to mathematics. *The American Mathematical Monthly*, 81(4), 323–343.

- Knuth, D. E. (1985). Algorithmic thinking and mathematical thinking. *The American Mathematical Monthly*, 92(3), 170–181.
- Krantz, S. G. (1984). Letters to the editor. *The American Mathematical Monthly*, 91(9), 598–600.
- MacKenzie, D. (2001). *Mechanizing proof: Computing, risk, and trust*. Cambridge, MA, USA: MIT Press.
- Mahoney, M. S. (2011). *Histories of computing*. Cambridge, MA, USA: Harvard University Press.
- Mengel, M. E. (1955). Present and projected computer manpower needs in business and industry. In A. W. Jacobson (editor), *Proceedings of the First Conference on Training Personnel for the Computing Machine Field* (pages 4–9). Detroit, MI: USA. Wayne University Press.
- Metropolis, N., Howlett, J., & Rota, G.-C. (editors). (1980). *A history of computing in the twentieth century: A collection of essays with introductory essay and indexes*. London, UK: Academic Press.
- Morse, P. M. (1960). Report on a conference of university computing center directors (June 2–4, 1960). *Communications of the ACM*, 3(10), 519–521.
- Mounier-Kuhn, P. (2012). Computer science in French universities: Early entrants and late-comers. *Information & Culture*, 47(4), 414–456.
- Naur, P., & Randell, B., editors (1969). *Software Engineering: Report on a Conference Sponsored by the NATO Science Committee*, Garmisch, Germany. NATO Scientific Affairs Division: Brussels, Belgium.
- Oettinger, A. G. (1967). The hardware-software complementarity. *Communications of the ACM*, 10(10), 604–606.
- Oettinger, A. G. (1968). Computers and education. In A. Finerman (editor), *University education in computing science* (pages 27–38). New York, NY, USA: Academic Press.
- Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York, NY, USA: Basic Books.
- Piore, E. R. (1969). Challenges to progress in computing: People, technology, purposes. *SIGCSE Bulletin*, 1(3), 6–8.
- Pitts, G. N., & Bateman, B. (1974). A software oriented computer science program. In D. S. Kerr (editor), *SIGCSE '74: Proceedings of the Fourth SIGCSE Technical Symposium on Computer Science Education* (pages 33–36). New York, NY: USA. ACM.
- Ralston, A. (1984). The first course in computer science needs a mathematics corequisite. *Communications of the ACM*, 27(10), 1002–1005.
- Ralston, A., & Shaw, M. (1980). Curriculum '78—Is computer science really that unmathematical?. *Communications of the ACM*, 23(2), 67–70.
- Rowan, T. C. (1957). Psychological tests and selection of computer programmers. *Journal of the ACM*, 4(3), 348–353.
- Shackelford, R., Cross, I. I., Davies, J. H., Impagliazzo, G., Kamali, J., LeBlanc, R., ... Topi, H. (2005). *Computing curricula 2005: The overview report*. New York, NY, USA: ACM, AIS, and IEEE-CS.
- Simon (2015). *Emergence of Computing Education as a Research Discipline*. PhD thesis, Aalto University, Finland.
- Solomon, C. (1986). *Computer environments for children: A reflection on theories of learning and education*. Cambridge, MA, USA: MIT Press.
- Spier, M. J. (1974). A critical look at the state of our science. *ACM SIGOPS Operating Systems Review*, 8(2), 9–15.
- Statz, J., & Miller, L. (1975). Certification of secondary school computer science teachers: Some issues and viewpoints. In E. C. Joseph and J. D. White (editors), *Proceedings of the 1975 Annual Conference, ACM '75* (pages 71–73). New York, NY: USA. ACM.

- Stevenson, D. E. (1993). Science, computational science, and computer science: At a crossroads. In S. C. Kwasny and J. F. Buck (editors), *CSC '93: Proceedings of the 1993 ACM conference on Computer science* (pages 7–14). New York, NY: USA: ACM.
- Sutinen, E., & Tarhio, J. (2001). Teaching to identify problems in a creative way. In *Proceedings of the FIE'01 Frontiers in Education Conference, volume T1D* (pages 8–13). Reno, NV: USA.
- Tedre, M. (2014). *The Science of computing: Shaping a discipline*. New York, NY, USA: CRC Press/Taylor & Francis.
- Tedre, M., & Denning, P. J. (2017). Shifting identities in computing: From a useful tool to a new method and theory of science. In H. Werthner & F. Van Harmelen (editors), *Informatics in the future—In the year 2025*. Cham, Germany: Springer.
- Tedre, M., & Denning, P. J. (2016). The long quest for computational thinking. In J. Sheard and C. Suero Montero (editors), *Proceedings of the 16th Koli Calling International Conference on Computing Education Research Koli Calling '16* (pages 120–129). New York, NY: USA: ACM.
- Tedre, M., & Moisseinen, N. (2014). Experiments in computing: A survey. *The Scientific World Journal*, 2014(#549398), 1–11.
- Tomayko, J. E. (1998). Forging a discipline: An outline history of software engineering education. *Annals of Software Engineering*, 6(1/4), 3–18.
- Turner, A. B. (1991). Computing Curricula 1991. *Communications of the ACM*, 34(6), 68–84.
- Walker, H. M. (2018). *Teaching computing: A practitioner's perspective*. Boca Raton, FL, USA: CRC Press.
- Walker, H. M., & Schneider, G. M. (1996). A revised model curriculum for a liberal arts degree in computer science. *Communications of the ACM*, 39(12), 85–95.
- Walker, T. M. (1973). Computer science curricula survey. *ACM SIGCSE Bulletin*, 5(4), 19–28.
- Walters, F. G. (1988). Computer science—A mathematical science and accreditation. *ACM SIGCSE Bulletin*, 20(3), 53–56.
- Wegner, P. (1970). Three computer cultures: Computer technology, computer mathematics, and computer science. In F. L. Alt & M. Rubinoff (editors), *Advances in computers, volume 10* (pages 7–78). New York, NY, USA: Elsevier.
- Wilkes, M. V., Wheeler, D. J., & Gill, S. (1951). *The preparation of programs for an electronic digital computer*. Cambridge, MA, USA: Addison-Wesley.
- Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33–35.
- Winsberg, E. B. (2010). *Science in the age of computer simulation*. Chicago, IL, USA: The University of Chicago Press.
- Wirth, N. (2008). A brief history of software engineering. *IEEE Annals of the History of Computing*, 30(3), 32–39.
- Wishner, R. P. (1968). Letters to the editor: Comment on curriculum 68. *Communications of the ACM*, 11(10), 658.
- Zadeh, L. A. (1965). Electrical engineering at the crossroads. *IEEE Transactions on Education*, 8(2), 30–33.