The Computer Boys Take Over

Computers, Programmers, and the Politics of Technical Expertise

By Nathan L. Ensmenger. Cambridge, MA: M.I.T. Press, 2010, 336 pages.

Digital computers seem to improve constantly, and their programs are impressive, even if not flawless. From within the software industry, however, the view is of a series of crises — crises in the availability of programmers, in their education, in their management, and in the very software engineering enterprise in which they engage. Yet the history of software and of its producers has received far less attention than the history of computer hardware. The attention that it has received has often been akin to the "great machines" and "great men" focuses of some other computer history; one book on programming languages even refers to the "hero languages."Historians of computing such as Martin Campbell-Kelly, Atsushi Akera, and Tom Haigh have recently been addressing that imbalance — the last two, like Ensmenger, studied in the History and Sociology of Science Department at the University of Pennsylvania. The book under review is a welcome addition.

Ensmenger describes the origins of computer programming and the competing interests that shaped its practice. It is about the growth of a practice — of programming or of software engineering — and about differing views of it as a craft, a profession, or a technology, and about the periodic crises that seemed to characterize the activity and its products. It is also about the masculinization of an activity in which women initially had a large role.

Computer programming as a job began with the ENIAC at the end of World War II. The ENIAC was not initially a stored-program computer, and so the task of translating a plan of computation into machine terms was carried out by wiring the components instead of writing machine code. At first it appeared that this task would be routine once the plan was worked out, and would be done by the women who had previously computed gunnery tables by following simple plans of calculation. What had seemed to be a job of routine coding, however, rapidly turned out to be a more difficult one that required understanding how the computer worked, how it might fail, and how to get the most out of its very limited capabilities. Rather than simple coding, programming became a black art at which certain people were highly skilled. Programmers, it seemed, were born, not made.

As the computer industry grew during the 1950s and 1960s, industry publications warned of a shortage of programmers. Ensmenger describes recruiting attempts such as IBM ads in the *New York Times* and other publications in 1956 that sought people with an interest in music, chess, or simply with "a lively imagination."The first ads elicited only two applicants new to programming, and only one of those worked out. A 1968 effort included testing Sing-Sing Prison inmates, with job offers to good candidates upon release.

In the 1960s, manufacturers such as the System Development Corporation (SDC) and IBM used aptitude tests and psychological profiles to identify programmers. The tests frequently emphasized mathematical skills, though there was little evidence that those skills were important in much data processing. Psychological traits of good programmers, such as creativity and emotional stability were also hard to identify in tests or profiling. One set of studies found that disinterest in people characterized programmers, both male and female, reinforcing the lore of the geeky hacker.

The notion of programming ability as innate, even if hard to characterize, should make the field as accessible to women as to men. It didn't turn out that way. Tests that focused on mathematical skills selected against women, who often had not studied

Digital Object Identifier 10.1109/MTS.2013.2247728
Date of publication: 14 March 2013

David Hemmendinger is Professor Emeritus of Computer Science at Union College, Schenectady, NY 12308; email:hemmendd@union.edu.

it extensively. Personality profiling that emphasized antisocial traits also fell into the trap of treating this as typically male. A field that had initially appeared quite gender-neutral rapidly became less so.

"Tower of Babel," the chapter on the role of programming languages in the late 1950s and the 1960s, presents them as means for achieving control of the programming activity as much as means for controlling computers."Automatic programming" was a term used to characterize programming languages — not that programming became automated, but that by contrast with low-level machine code, higher-level languages were supposed to let computer users rather than the "computer boys" write programs. Scientists could write algebraic expressions in Fortran; business people could write in Cobol, and "Susie Meyer" in an IBM advertisement for the PL/I language "could find happiness handling both commercial and scientific applications" with it, despite having no programming experience — and if she could, so could anyone.

Programming languages could thus be seen as deskilling, enabling managers to replace computer boys with cheaper and less powerful labor, whether male or female. A language like Cobol, with its English-like notation, could let managers understand programs. Of course, these languages were also expected to help skilled programmers to resolve the "software crisis" — the widespread difficulty in producing system software on time or to function properly. It is not surprising that for all their merits, programming languages fell short of fulfilling these (inconsistent) expectations.

One of the strengths of Ensmenger's book is its showing how what we may take as "obvious," such as the basic character of computer science as an academic discipline, emerges from multiple claims to define the field. Indeed, it was not obvious at the start that there should be such a field — new technologies often don't give rise to new disciplines. Academics who worked in computing wanted an intellectual foundation for programming, employers wanted educational standards, and computing workers wanted their activity to have professional standards. The major academic computing organization, the Association for Computing Machinery (ACM), also had non-academic members, but when it developed the first computer science curriculum standard, it was heavily theoretical and mathematical, and in the view of many in computing occupations, gave too little attention to practical data processing.

The book is not primarily about academic computer science, though in its discussion of the ways in which women became excluded from the programming profession, it might have added something on university departments. They largely grew out of electrical engineering and mathematics departments, both fields that were heavily male in the 1960s, and so the composition of computer science departments shared that imbalance. The computer science network was an old-boys network from the start.

An early ACM definition of computer science was that it was the study of information. This definition could unite several fields, and it gave rise to the term "informatics" that several European languages use instead of "computer science" Ensmenger uses Thomas Kuhn's notion of "normal science" to argue that computer science became established as the science of algorithms when it acquired its major textbook in Donald Knuth's *Art of Computer Programming*, which made algorithms central, traced their mathematical lineage, and helped to set the agenda for further work to define the profession.(My first reaction, as an academic computer scientist, was "of course that's what it's about!," but Ensmenger rightly points out that this was a matter of construction.)

The following chapters treat the professionalization of programming. During the 1960s the cost of computing rose, particularly the cost of software. Although the term "software crisis" that came from the 1968 NATO conference on software engineering originally referred to system software, from the standpoint of corporate profit, the complexity of application software presented similar problems. Furthermore, the computer boys continued to appear hard to manage — unkempt, unruly, and unsocial. They seemed to require autonomy in order to deal with the computer as only they knew how, while managers sought to define their role more narrowly as technicians.

During the 1960s two primary organizations competed to define the programming profession, the Data Processing Management Association (DPMA), and the ACM. The former offered its Certified Data Processor (CDP) exam, while the latter had its computer science curricular guidelines, as well as regular conferences of academics and others. Although the DPMA was focused on professional programmers, interest in its CDP program declined after a few years of growth. It was not clear that its certification met industrial needs, and as with earlier aptitude tests, there were charges of fraud. The ACM, while narrower, came to play a larger role in defining the computing field through standard degree programs. Its definition, however, did not directly address the professional status of programming.

Software engineering is concerned with managing complexity in programs of thousands or millions of lines of code. Ensmenger discusses several standard approaches, focusing on the multiple senses of "manage."There were attempts to make software development an industrial discipline, modeled on factory mass production. There was the "chief programmer team" (CPT), akin to a surgical team with a master programmer and skilled assistants who divided the work. This design could conflict with corporate managerial

authority. In effect it reinforced the notion of the skilled programmer as beyond standard authority. Finally, the "egoless," or adaptive approach used teams of peers rather than a hierarchy. It emphasized programming as creative, and sought to minimize programmers' supposed antisocial tendencies while encouraging them as professionals. Ensmenger argues, though, that managers co-opted this view by emphasizing instead that egoless programming meant that programmers were replaceable units. In the end, neither the CPT nor the egoless approach gained dominance, and software continued to be designed by methods that did not meet standards of engineering practice.

In his conclusion, Ensmenger argues that in defining themselves, programmers adopted rhetoric that allowed them variously to present their work as an art and as science or engineering. They failed fully to professionalize, though, and there is no established certification process as there is in law and medicine. His somewhat deflationary conclusion is that programmers are best regarded as technicians, who draw both on scientific or engineering education and on craft knowledge. As workers, they carry briefcases, but those often contain tools — they are white-collar, but sometimes get their hands dirty. It appears that the crises in software — in producing it, in managing programmers, and in the tensions between the two sides of the technician — will persist because, in the words of a 1996 quote, "excellent developers, like excellent musicians and artists, are born, not made" [1].

The book is thoroughly documented, with 41 pages of notes and an extensive bibliography, though it could use more editing to eliminate duplications. A quote by Maurice Wilkes on realizing that he would spend the rest of his time debugging programs appears four times. John Tukey's coining of "software" is described or cited twice, as is a quote from John Backus on programming as a black art, Grace Hopper's comparison of programming and planning a dinner, along with several other such repetitions.

The first chapter, which introduces the argument of the book, has several passages with strong claims that are not fully supported by their citations. It calls Tukey's definition of software "strictly negative," though he describes software as "comprising the carefully planned interpretive routines, compilers, and other aspects of automative programming" — far from negative. The same passage of the book suggests that early uses of the term can refer even to personnel, citing five sources. Of those, one actually has "The programming and the programmers (the 'software')," but the other four clearly use "software" to refer only to computer systems programs. It does not help the argument to include personnel among software. The book goes on to say that the "hard/soft" contrast embodies hierarchy and gender distinctions and that the softer aspects are implicitly secondary — yet Tukey's article refers to software as "at least as important" to computers as their hardware.

The book says of early papers by Herman Goldstine and John von Neumann on programming [2] that they distinguish between "the headwork of the (male) scientist or 'planner', and the handwork of the (largely female) 'coder'" (p. 15). It lists their programming steps, the last two of which are the dynamic analysis of control flow of the program, and the static coding, which the book says is done by low-status female coders. But von Neumann and Goldstine explicitly say that *both* of these last two steps are "the coding proper," and that every mathematician should be able to do the dynamic coding. What they call coding is thus not merely the "handwork" level, and the distinction between male planners [programmers] and female coders is not as sharp as the book suggests.

These are minor flaws in a book that generally develops its argument well. Its analyses of the competing claims to authority in defining programming help to reveal the social construction of this activity, and how the efforts to form a profession tended to exclude women.

Finally, although the book is about computing in the United States, it finds a supporting view in a 1947 lecture by Alan Turing [3], foresighted in this matter as in so many others:

> The masters [programmers of his proposed ACE computer] are liable to be replaced because as soon as any technique becomes at all stereotyped it becomes possible to devise a system of instruction tables [programs] which will enable the electronic computer to do it for itself. It may happen however that the masters will refuse to do this. They may be unwilling to let their jobs be stolen from them in this way. In that case they would surround the whole of their work with mystery and make excuses, couched in well chosen gibberish, whenever any dangerous suggestions were made. I think that a reaction of this kind is a very real danger.

References

[1] W.W. Gibbs, "Software's chronic crisis," *Scientific American*, vol. 271, no. 3, p. 86, 1994.

[2] H.H. Goldstine and J. von Neumann, "Planning and coding problems for an electronic computing instrument," in J. von Neumann, *Collected Works*, A. Taub, Ed., vol. 5. New York, NY: Macmillan, 1963, pp. 100–101.

[3] A.M. Turing, "Lecture to the London Mathematical Society on 20 February 1947," in *Collected Works of A. M. Turing: Mechanical Intelligence*, D.C. Ince, Ed. New York, NY: North-Holland, 1992, p. 102.