IRSH 48 (2003), Supplement, pp. 153–180 DOI: 10.1017/S0020859003001305

Letting the "Computer Boys" Take Over: Technology and the Politics of Organizational Transformation

Nathan L. Ensmenger

> Most experts agree that another barrier to the most desirable use of the computer is the immense culture and communication gap that divides managers from computer people. The computer people tend to be young, mobile, and quantitatively oriented, and look to their peers both for company and for approval [...]. Managers, on the other hand, are typically older and tend to regard computer people either as mere technicians or as threats to their position and status – in either case they resist their presence in the halls of power.
>
> T. Alexander, "Computers Can't Solve Everything", *Fortune*, (1969)

INVENTING THE COMPUTER PROGRAMMER

In the decades following the development of the first electronic digital computers, the computer industry in the United States grew from nothing into an important and expansive sector of the American economy. Whereas in the early 1950s electronic computers were generally regarded as interesting but extravagant scientific curiosities, by 1963 these devices and their associated peripherals formed the basis of a billion-dollar industry. By the beginning of the 1970s, more than 165,000 computers had been installed in the United States alone, and the computer and software industries employed several hundred thousand individuals worldwide.[1]

Co-evolving with these flourishing new information industries was a novel species of technical professional, the computer programmer. In 1945 there were no computer "programmers", professional or otherwise; by 1967 industry observers were warning that although there were at least 100,000 programmers working in the United States, there was an immediate need for at least 50,000 more.[2] "Competition for programmers", declared a contemporary article in *Fortune* magazine, "has driven salaries up so fast that programming has become probably the country's highest paid technological occupation [...]. Even so, some companies can't

1. Kenneth Flamm, *Creating the Computer Government, Industry, and High Technology* (Washington DC, 1988), p. 135.
2. Bruce Gilchrist and Richard Weber (eds), *The State of the Computer Industry in the United States* (New York, 1972).

find experienced programmers at any price."[3] By the end of the 1960s they had become the center of the "software crisis", a debate about the health and future of the computer industry that was to continue for the next quarter of a century.

It many respects it is the history of the computer programmer, rather than the computer itself, that is most important to our understanding of this crucial period of rapid and fundamental transformation in the history of information technology. While electronic computing held a certain high-tech appeal for many corporate executives in the late 1950s and early 1960s, few had any idea how to integrate effectively this expensive, unfamiliar, and often unreliable technology into their existing operations. It was the computer programmers who developed the applications software that transformed the latent power of a general-purpose computer into a specific tool for solving actual real-world problems. For many organizations, it was the availability of software that most determined the success or failure of their computerization efforts. As computer hardware became faster, more reliable, and less expensive, the relative importance of software – and software developers – became even more pronounced.

Despite their obvious importance to the history of information technology, computer programmers represent a perplexing problem for the historian. We know almost nothing about who they were, where they came from, or what their daily work lives were like. Neither laborers nor professionals, they defy traditional occupational categorizations. The ranks of the elite programmers included both high-school dropouts and ex-Ph.D. physicists. Originally envisioned as little more than glorified clerical workers, they quickly assumed a position of power within the corporation vastly disproportionate to their official organizational role. Defined by their mastery of the highest of high technology, they were often derided for their adherence to artisanal practices. Although associated with the emerging academic discipline of computer science, they were never widely considered to be either scientists or engineers. Even to this day, their occupational expertise remains difficult to clearly define or delineate.[4] The term "programmer" itself encompasses such a wide range of occupational categories – from the narrow and highly technical "coder" to the elite and influential "systems man" – that it is more useful as a rhetorical device than as an analytical category.[5]

3. Gene Bylinsky, "Help Wanted: 50,000 Programmers", *Fortune*, 75 (March 1967), pp. 445–556, 141.

4. Bruce Webster, "The Real Software Crisis", *Byte*, 21 (1996), p. 218.

5. Although many of the earliest programmers were women, by the beginning of the 1960s programming was generally considered a male profession. Certainly the elite ranks of "systems men" were, quite literally, men. As Margaret Rossiter and others have suggested, masculinity was a cultural resource that aspiring professionals could draw upon in order to improve the standing of their discipline. For many in this period the very concept of a professional was

It is precisely this ambiguous occupational identity that makes the computer programmer such a fascinating and controversial figure. Throughout the 1950s and 1960s, the identity of the computer programmer was negotiated and renegotiated in response to a changing social and technological environment. By virtue of their close relationship to the increasingly powerful technology of electronic computing, computer programmers became the subject of highly contested boundary disputes with traditional corporate employees. In the language of the contemporary management literature, programmers represented influential "change-agents". In their role as mediators between the technical system (the computer) and its social environment (existing structures and practices), computer programmers played a crucial role in transforming the computer from a scientific instrument into a powerful tool for corporate control and communication. As such, they also served as a focus for opposition to and criticism of the use of new information technologies. From the "programmer personnel" problem of the 1950s to the "software turmoil" of the 1960s to the "world-wide shortage of information technology workers" in the 1990s, the focus of debate about the software crisis has continued to center around the unique nature of computer programming as an intellectual and occupational activity.[6] The Y2K crisis, the H1-B visa debates, recent concerns about the loss of programming jobs to India and Pakistan, are all part of a much larger pattern of debate about the new structure of technical labor in the late twentieth and early twenty-first centuries. This paper will explore the emergence of the computer programmer as a central figure in an ongoing debate about the role of information technology in organizational transformation. It focuses on the conflict between the craft-centered practices of the computer programmers and the "scientifically" oriented management techniques of their corporate managers. It argues that the skills and expertise that computer programmers possessed transcended traditional boundaries between business knowledge and technical expertise, and that computer programmers constituted a substantial challenge to established corporate hierarchies and power structures. It suggests that the continued persistence of a "software crisis" mentality among industrial and governmental managers, as well as the seemingly unrelenting quest of these managers to develop a software development methodology that would finally eliminate corporate dependence on the craft knowledge of individual programmers, can best be understood in light of this struggle over workplace authority.

synonymous with an all masculine and thus high-status occupation. A more thorough discussion of the systems men can be found in Thomas Haigh, "Inventing Information Systems: The Systems Men and the Computer, 1950–1968", *Business History Review*, 75 (2001), pp. 15–61.

6. Office of Technology Policy, United States Department of Commerce, "America's New Deficit: The Shortage of Information Technology Workers", technical report (1997).

THE ORIGINS OF PROGRAMMING

The first clear articulation of what a programmer was and should be was provided in the late 1940s by Herman Goldstine and John von Neumann in a series of volumes on "Planning and Coding of Problems for an Electronic Computing Instrument". These volumes, which served as the principal textbooks available on the computer programming until at least the early 1950s, outlined a clear division of labor in the programming process that seems to have been based on the practices used in programming the ENIAC. These practices were themselves adapted from those used at the large manual computation projects at the nearby Aberdeen Proving Grounds. In these projects, the most senior women (by this point in time manual computation had become an almost exclusively feminine occupation) developed elaborate "plans of computation" that were carried out by their fellow human "computers". Since electronic computing was envisioned by the ENIAC developers as "nothing more than an automated form of hand computation", it seemed natural that similar plans could be constructed for their electronic counterparts.[7]

Drawing on their experience with the ENIAC, Goldstine and von Neumann spelled out a six-step programming process that clearly differentiated between the high-level conceptual activities involved in "planning" an algorithm and the tedious but straightforward work of "coding" it into machine-readable form. The planner would conceptualize the problem mathematically and physically, perform a numerical analysis to determine precision requirements and evaluate potential approximation errors, and design the overall algorithm. The coder would merely write out steps of a computation in a form that could be read by the machine, either by encoding them on punch cards, or in the case of ENIAC, by plugging cables and setting switches. Whereas the planners were typically scientists or engineers (and therefore also overwhelmingly male), the coders were low-status and almost inevitably female (at least at the ENIAC project).

The use of the designation "coder" in this context is significant. "Coding" suggested a highly specialized – and rigidly circumscribed – set of occupational activities that required little more than conscientious precision and a small degree of technical training. Like keypunch operators or stenographers, coders existed only to transcribe the thoughts of others. They merely operated the machinery designed by others. Coders obviously ranked low on the intellectual and professional hierarchy. As the historian Jennifer Light has suggested, coders were quite literally the "invisible technicians" of the ENIAC project: in press coverage of the

7. David Allan Grier, "The ENIAC, the Verb to Program and the Emergence of Digital Computers", *Annals of the History of Computing*, 18 (1996), pp. 51–55, 53.

project they were never referred to individually, and in many of the publicity photos they were cropped out entirely.[8]

The idea that the aspects of the programming process most intimately connected to the manipulation of the machine (rather than the definition of the problem) were both low-status and uninteresting carried forward into the commercial computer projects of the early 1950s. An early manuscript version of the UNIVAC "Introduction to Programming" manual, for example, highlighted the distinction between the managerial "programmer" and the technical "coder":

> [...] in problem preparation, the detailed work may be accomplished by two individuals. The first, who may be called the "programmer", studies the problem, determines the appropriate method of solution, and prepares the flow chart. This person must be well versed in the particular field in which the problem lies, and should also be able to fully exploit the flexibility and versatility of the UNIVAC system. The second person, referred to as the "coder" need only be familiar with the technique of reducing the flow chart to the specific instructions, or coding, required by the UNIVAC to solve the problem.[9]

By differentiating between these two tasks, one clerical and the other analytical, the manual reinforced the Goldstine/von Neumann model of the programming process. It is important to note that the versatile "programmer" referred to above was not a computer specialist. He was the planner, the provider of abstract knowledge, not the master of the machinery; the actual "computer" aspect of electronic computing was to be delegated to specialist technicians. In this model the real business of software development was analysis; the actual coding aspect of programming was trivial and mechanical. "Problems must be thoroughly analyzed to determine the many factors that must be taken into consideration", suggested the aforementioned manual, but the once this analysis had been completed, the "pattern of the [programming] solution would be readily apparent". Although this division of the programming process into two distinct and unequal phases did not survive into the published version of the UNIVAC documentation, its earlier inclusion highlighted the ubiquity of such distinctions.

Despite this general insistence of the planning/coding distinction, however, in actual practice it was often difficult to differentiate between the two functions. As the ENIAC project leaders themselves discovered to their dismay, controlling the operation of an automatic computer was nothing like the process of hand computation, and the "ENIAC girls"

8. Jennifer Light, "When Computers Were Women", *Technology & Culture*, 40 (1999), pp. 455–483.

9. Sperry Rand Univac, "Introduction to Programming", Programming for the UNIVAC, Part 1 (typewritten manuscript, 11 June 1949); Hagley Achives, Box 372, Accession 1825.

were therefore responsible for defining the first state-of-the-art methods of programming practice. Programming was a very imperfectly understood activity in these early days, and much more of the work devolved on the coders than anticipated. To complete their coding, the coders would often have to revisit the mathematical analysis of the problem at hand; and with their growing skills, some scientific users left many or all of the six programming stages to the coders. In order to debug their programs and to distinguish hardware glitches from software errors, they developed an intimate knowledge of the ENIAC machinery. "Since we knew both the application and the machine", claimed ENIAC programmer Betty Jean Jennings, "we learned to diagnose troubles as well as, if not better than, the engineers".[10] Like many later observers, Goldstine and von Neumann appear to have described the division of labor in the programming process as they would have preferred it to be, rather than how it existed in practice.

THE "BLACK ART" OF PROGRAMMING

The transformation of the computer programmer from clerical worker to technical specialist was not confined solely to the ENIAC project. In electronic computer installations all over the country the limitations of first generation hardware devices demanded of programmers the development of creative innovations and "work-arounds". For example, many of these machines did not have floating-pointing hardware, so the programmers had to do complicated calculations to ensure that the values of the variables would stay within the machine's fixed range throughout the course of the calculation. Little was known about the best algorithms and numerical methods to use for this purpose, so a programming problem could often turn into a research excursion in numerical analysis. Memory devices had very little capacity, and programmers had to develop great skill and craft knowledge to fit their programs into the available memory space. Devices were also slow, so tricks and intricate calculations were required to make sure to get every bit of speed out of the machines, such as carefully placing an instruction at a particular location on the drum memory so that the read head would be passing by that very location on the drum at the time when it came time to execute that instruction.

It is during this period that a peculiar cultural stereotype of the computer programmer emerges. The emphasis on individual creativity and idiosyncratic technique in contemporary programming practice suggested that computer programmers, like chess masters or virtuoso musicians, were endowed with a uniquely creative ability. Programmers were therefore selected for their intellectual gifts and aptitudes, rather than

10. W. Barkley Fritz, "The Women of ENIAC", *Annals of the History of Computing*, 18 (1996), pp. 13–28, 20.

their business knowledge or managerial savvy. "Look for those who like intellectual challenge rather than interpersonal relations or managerial decision-making. Look for the chess player, the solver of mathematical puzzles."[11] Because the presence of one of these gifted programmers could often determine the difference between the success and failure of an expensive electronic data processing (EDP) project, companies went to great lengths to identify and retain them.

The popular notion that good programmers were born, not made, was supported by a series of aptitude tests and personality profiles developed by employers and human resources experts. One widely cited IBM study determined that code produced by a truly excellent programmer was twenty-six times more efficient than that produced by his merely average colleagues.[12] Despite the serious methodological flaws that compromised this particular study (including a sample population of only twelve individuals), the 26:1 performance ratio quickly became part of the standard lore of the industry. Dr. E.E. David of Bell Telephone Laboratories spoke for many when he argued that large software projects could never be managed effectively, because "the vast range of programmer performance indicated earlier may mean that it is difficult to obtain better size-performance software using machine code written by an army of programmers of lesser average caliber".[13] Skilled programmers were thought to be effectively irreplaceable, and were treated and compensated accordingly.

It is during this period that most corporations stopped formally differentiating between "programmers" and "coders". The now commonplace designation "programmer" was adopted to describe the entire process of application development. The verb "to program", with its military connotations of "to assemble" or "to organize", suggested a more thoughtful and system-oriented activity.[14]

Even the development of new "automatic programming systems" such as FORTRAN and COBOL, although originally intended to eliminate the need for skilled programmers altogether, had the unintended effect of elevating their status. For those interested in advancing the academic status of computer science, the design of programming languages provided an ideal forum for exploring the theoretical aspects of their discipline. More practical-minded programmers saw programming languages as a means of

11. Joseph O'Shields, "Selection of EDP Personnel", *Personnel Journal*, 44 (1965), pp. 472–474, 472.

12. Hal Sackman, W.J. Erickson and E.E. Grant, "Exploratory Experimental Studies Comparing Online and Offline Programming Performance", *Communications of the ACM*, 11 (1968), pp. 3–11, 3.

13. Peter Naur, Brian Randall, and J.N. Buxton, *Software Engineering: Proceedings of the NATO Conferences* (New York, 1976), p. 33.

14. Grier, "The ENIAC, the Verb to Program", pp. 51–55, 53.

eliminating the more onerous and error-prone aspects of software development. By eliminating much of the tedium associated with low-level machine coding, they allowed programmers to focus less on technical minutia and more on high-status activities such as design and analysis.

It is this last development that is most significant. Software development managers soon discovered that although automatic programming systems helped eliminate simple syntax and transcription errors, they did little to reduce the underlying complexity of the programming process. As the long-time industry analyst Willis Ware suggested in a 1965 editorial:

> We lament the cost of programming; we regret the time it takes. What we really are unhappy with is the total programming process, not programming (i.e. writing routines) per se. Nonetheless, people generally smear the details into one big blur; and the consequence is, we tend to conclude erroneously that all our problems will vanish if we can improve the language which stands between the machine and the programmer. 'Tain't necessarily so. All the programming language improvement in the world will not shorten the intellectual activity, the thinking, the analysis, that is inherent in the programming process. Another name for the programming process is "problem solving by machine", perhaps it suggests more pointedly the inherent intellectual content of preparing large problems for machine handling.[15]

Since so much of the programming process involved "intellectual activity, mathematical investigation, discussions between people", very often, individuals who were trained as programmers actually do the early stages of the programming process but do none of the actual writing. Ware estimated that at least one-half of the total programming man-hours in a project was occupied by analysis and definition of the problem.[16]

Willis Ware was not the only observer to argue for the expansion of the programmer's occupational bailiwick to include design and analysis. A 1959 Price-Waterhouse report on "Business Experience with Electronic Computers" argued that, whereas a knowledge of business of operations could usually be obtained by an adequate expenditure of time and effort, "innate ability [...] seems to have a great deal to do with a man's capacity to perform effectively in the fields of computer coding and systems design".[17] In fact, the study's authors suggested,

> [...] the term "programmer" [...] is unfortunate since it seems to indicate that the work is largely machine oriented when this is not at all the case [...] training in systems analysis and design is as important to a programmer as training in

15. Willis Ware, "As I See It: A Guest Editorial", *Datamation*, 11 (1965), pp. 27–28, 27.
16. *Ibid.*, p. 27.
17. B. Conway, J. Gibbons and D.E. Watts, *Business Experience with Electronic Computers: A Synthesis of What Has Been Learned from Electronic Data Processing Installations* (New York, 1959), p. 83.

> machine coding techniques; it may well become increasingly important as systems get more complex and coding becomes more automatic.[18]

The clear implication of recent experience, in both scientific computation and business data processing, seemed to be that programmers should be given more responsibility for design and analysis, that the idea that coding could be left to less experienced or lower-grade personnel was "erroneous", and that "the human element [was] crucial in programming".[19]

The growing role of programmers in high-level design activities, combined with a continued emphasis on innate skills and ability, provided individual programmers with a certain degree of immunity from managerial imperatives. Indeed, throughout the 1950s software development projects were thought to be almost impossible to manage using conventional methods. [20] The general consensus was that computer programming was "the kind of work that is called creative [and] creative work just cannot be managed".[21] One industry observer went so far as to argue that the "major managerial task is finding – and keeping – 'the right people': with the right people, all problems vanish".[22] During this period, many corporate programmers enjoyed an unprecedented degree of personal authority and professional autonomy. Programmers were not only "encouraged to feel they are professionals", but they were included as active participants in all phases of application development, from design to implementation, in order to ensure their cooperation and enthusiasm.[23]. For the time being, the power to control the computer rested with the individual programmer, rather than with the management bureaucracy.

The ambiguous nature of their corporate identity proved to be something of a mixed blessing for programmers, however. The perceived lack of managerial control over the programming process provoked tension within the corporate structure. As the electronic computer became increasingly central to the social and economic life of commercial organizations, the exceptional status and practices the computers programmers began to attract increased an unwelcome attention. The same personality traits that were seen as indicative of genius could also be seen as antisocial and subversive. The lack of widely accepted formal methods for evaluating programmer aptitude and ability weakened their claims to "professional" status: if programming was indeed more art than science, its

18. *Ibid.*
19. *Ibid.*, p. 90.
20. Bylinsky, "Help Wanted: 50,000 Programmers", pp. 445–556, 141; Charles Lecht, *The Management of Computer Programming Projects* (New York, 1967), p. 9.
21. Robert Gordon, "Review of Charles Lecht, *The Management of Computer Programming Projects*", *Datamation*, 14 (1968), pp. 200–202, 200.
22. Robert Gordon, "Personnel Selection", in Fred Gruenberger and Stanley Naftaly (eds), *Data Processing [...] Practically Speaking* (Los Angeles, CA, 1967), p. 88.
23. Conway *et al.*, *Business Experience with Electronic Computers*, p. 81.

practitioners could hardly claim the same status as other white-collar professionals. [24] When John Backus (the IBM researcher best known as the inventor of the FORTRAN programming language) famously described programming in the 1950s as "a black art, a private arcane matter [...] [in which] the success of a program depended primarily on the programmer's private techniques and inventions", he did not intend it to be a compliment.[25] The same qualities that had previously been thought essential indicators of programming ability, such as creativity and a mild degree of personal eccentricity, could also be perceived as being merely unprofessional.

Nevertheless, the long-standing association of programming ability with creative genius provided individual programmers with a powerful claim to personal and professional authority. By the end of the 1950s it had become clear that computer programmers were anything but routine clerical workers. But what kind of employee did that make them, exactly? The possession of valuable technical expertise did not automatically translate into professional standing or even long-term occupational survival. Other similarly skilled craftsmen had seen their occupations deskilled or eliminated – a historical fact that computer programmers seem to be well aware of. Computer programmers in this period seem to have been aware of their own ambiguous status, and worked to established the structures of professionalism: academic computer science curriculum, certification programs, and professional societies.[26]

SOFTWARE TURMOIL

By the beginning of the 1960s, however, developments occurred in both the technical and social environment of electronic computing that

24. In 1955 IBM introduced its Programmer Aptitude Test (PAT), which correlated performance in training programs with subsequent performance ratings by project managers and served for many years as a *de facto* industry standard. The test was adapted from a psychological examination developed by the American Council on Education, and included questions about number series, figure analogies, and arithmetic reasoning. By 1962 an estimated 80 per cent of all businesses used some form of aptitude test when hiring programmers. Although the IBM PAT was used by almost 40 per cent of these businesses, numerous alternatives were developed, and the other 60 per cent used some combination of more than 60 different exams. Although aptitude tests were accused of being inaccurate, irrelevant, and susceptible to widespread cheating, many employers continued to use them well into the 1970s. No single test was widely accepted as being very accurate or definitive, however; they were simply one of the only tools available for dealing with the problem of programmer labor.

25. Nick Metropolis, J. Howlett and Gian-Carlo Rota (eds), *A History of Computing in the Twentieth Century : A Collection of Essays* (New York, 1980), p. 126.

26. Nathan Ensmenger, "The 'Question of Professionalism' in the Computer Fields", *IEEE Annals of the History of Computing*, 23 (2001), pp. 56–73.

prompted a re-evaluation and re-negotiation of the computer programmer's proper place in corporate and professional hierarchies. In the first half of the decade innovations in transistor and integrated circuit technology increased the memory size and processor speed of computers by a factor of 10, providing an effective performance improvement of almost 100. The falling cost of hardware allowed computers to be used for more and larger applications, which in turn required larger and more complex software. As the scale of software projects expanded, they became increasingly difficult to supervise and control. They also became much more expensive. Large software development projects acquired a reputation for being behind schedule, over budget, and bug-ridden.

Commercial software development projects changed not only in size but in character. Whereas the first electronic computers were produced for military and scientific purposes, the second generation of computers were designed explicitly for business. In addition to producing general purpose computers that were relatively reliable and affordable, manufacturers like IBM could also provide the services and peripherals necessary to integrate the electronic computer into existing systems and processes. As the computer became more of a tool for business than a scientific instrument, the nature of its use – and of its primary user, the computer programmer – changed dramatically. The projects that business programmers worked on tended to be larger, more highly structured, and less mathematical than those involved in scientific computing. The needs of business demanded a whole new breed of programmers, and plenty of them.

The "personnel problem" posed by the shortage of programmers quickly assumed crisis proportions. As early as 1961 observers were already warning of a "gap in programming support" that threatened to "get worse [...] before it gets better".[27] Five years later "one of the prime areas of concern" to EDP managers was "the shortage of capable programmers", a shortage which had "profound implications, not only for the computer industry as it is now, but for how it can be in the future".[28] Large corporations like IBM struggled to develop costly internal training programs. Fly-by-night vocational schools sprung up all over the country, promising golden opportunities but delivering little more than trained typists.

The widespread programmer labor shortage, combined with a series of highly publicized software disasters, including the software related destruction of the Mariner 1 spacecraft and the infamous IBM OS/360 debacle (which cost IBM more than half a billion dollars – four times the original budget – the single largest expenditure in company history),

27. Robert Patrick, "The Gap in Programming Support", *Datamation*, 7 (1961), p. 37.
28. Richard Tanaka, "Fee or Free Software", *Datamation*, 13 (1967), pp. 205–206, 206.

lent credence to claims that an industry-wide software crisis was imminent.[29]

The focus of much of the debate about the burgeoning software crisis was not so much the computer itself as the computer programmer. In the late 1960s the venerable consulting company McKinsey & Company issued a series of reports suggesting that the real reason that most data processing installations were unprofitable is that "many otherwise effective top managements [...] have abdicated control to staff specialists – good technicians who have neither the operation experience to know the jobs that need doing nor the authority to get them done right".[30] The reports helped redefine contemporary understandings of the nature and causes of the software crisis by suggesting that the real "personnel problem" was not shortage but mismanagement. The solution to "unlocking the computer's profit potential", according to McKinsey & Company, was to restore the proper balance between managers and programmers: "Only managers can manage the computer in the best interests of the business. The companies that take this lesson to heart today will be the computer profit leaders of tomorrow."[31]

Freed from some of the constraints of earlier technology and eager to take advantage of a less-skilled (and less-expensive) workforce, managers began to look for solutions to the software crisis that would eliminate corporate dependence on the craft knowledge of individual programmers. New perspectives on these problems began to appear in the industry literature.

> There is a vast amount of evidence to indicate that writing – a large part of programming is writing after all, albeit in a special language for a very restricted audience – can be planned, scheduled, and controlled, nearly all of which has been flagrantly ignored by both programmers and their managers,

argued Robert Gordon in a 1968 review of contemporary software development practices.[32] The professional journals of this period are replete with exhortations towards better software development management: "Controlling Computer Programming"; "New Power for Management"; "Managing the Programming Effort"; "The Management of

29. The Mariner I incident involved a software problem that resulted in the destruction of multi-million-dollar spacecraft. The IBM OS/360 project, delivered nine months late and riddled with errors, took an enormous toll on the company, in both personal and financial terms. Its failure was the subject of one of the most widely read books on software project management, Frederick Brook's *The Mythical Man-Month* (Reading, MA, 1975).

30. McKinsey & Company, "Unlocking the Computer's Profit Potential", *Computers & Automation*, 18 (1969), pp. 24–33, 33.

31. *Ibid.*, p. 33.

32. Gordon, "Personnel Selection", p. 200.

Computer Programming Efforts".[33] Although it was admittedly true "that programming a computer is more an art than a science, that in some of its aspects it is a creative process", this new perspective on software management suggested that "as a matter of fact, a modicum of intelligent effort can provide a very satisfactory degree of control".[34]

It was the 1968 NATO Conference on Software Engineering that irrevocably established software management as one of the central rhetorical cornerstones of all future software engineering discourse. In the fall of that year a diverse group of influential computer scientists, corporate managers, and military officials gathered in Garmisch, Germany, to discuss their growing concern that the production of software had become "a scare item for management [...] an unprofitable morass, costly and unending". "We build software like the Wright brothers built airplanes", complained one prominent participant: "build the whole thing, push it off the cliff, let it crash, and start over again".[35] The solution to the so-called "software crisis", suggested the conference organizers, was for software developers to adopt "the types of theoretical foundations and practical disciplines that are traditional in the established branches of engineering".[36] In the interest of efficient software manufacturing, the "black art" of programming had to make way for the "science" of software engineering.

By defining the software crisis in terms of the discipline of "software engineering", the NATO Conference set an agenda that influenced many of the technological, managerial, and professional developments in commercial computing for the next several decades. For a number of conference participants, the key word in the provocative NATO manifesto was "discipline". For example, in his widely quoted paper on "mass-produced software components", Douglas McIlroy forcefully articulated his plan for "industrializing" software production:

> We undoubtedly produce software by backward techniques. We undoubtedly get the short end of the stick in confrontations with hardware people because they are the industrialists and we are the crofters. Software production today appears in the scale of industrialization somewhere below the more backward construction agencies. I think that its proper place is considerably higher, and would like to investigate the prospects for mass-production techniques in software.[37]

33. C.I. Keelan, "Controlling Computer Programming", *Journal of Systems Management*, 20 (January 1969), pp. 30–33; D. Herz, *New Power for Management* (New York, 1969); Richard Canning, "Managing the Programming Effort", *EDP Analyzer*, 6 (1968), pp. 1–15; Charles Lecht, *The Management of Computer Programming Projects* (New York, 1967).
34. Keelan, "Controlling Computer Programming", p. 30.
35. R.M. Graham, quoted in Naur *et al.*, *Software Engineering: Proceedings of the NATO conferences*, p. 32.
36. *Ibid.*, p. 4.
37. *Ibid.*, p. 7.

McIroy's vision of a software "components factory" invokes familiar images of industrialization and proletariatization. According to his proposal, an elite corps of "software engineers" would serve as the Frederick Taylors of the software industry, carefully orchestrating every action of a highly stratified programmer labor force. And like the engineers in more traditional manufacturing organizations, these software engineers would identify themselves more as corporate citizens than as independent professionals.[38]

The turn towards management solutions to the software crisis that followed the 1968 Garmisch Conference reflected a significant shift in contemporary attitudes towards programmers and other computer specialists. Indeed, many of the most significant innovations in software engineering to be developed in the immediate post-Garmisch era were as much managerial as they were technological or professional. When a prominent adherent of object-oriented programming techniques spoke of "transforming programming from a solitary cut-to-fit craft, like the cottage industries of colonial America, into an organizational enterprise like manufacturing is today", he was referring not so much to the adoption of a specific technology, but rather to the imposition of established and traditional forms of labor organization and workplace relationships.[39]

By reconstructing the software crisis as a problem of management technique rather than technological innovation, advocates of these new management-oriented approaches also relocated the focus of its solution, removing it from the domain of the computer specialist and placing it firmly in the hands of traditional managers.

A NEW THEOCRACY – OR INDUSTRIAL CARPETBAGGERS?

Prior to the invention of the electronic digital computer, information processing in the corporation had largely been handled by conventional clerical staffs and traditional office managers. There had been attempts by aspiring "systems managers" to leverage expertise in the technical and bureaucratic aspects of administration into a broader claim to authority over the design of elaborate custom information processing systems.[40] In certain cases, strong-willed executives were able to use information technology to consolidate control over lower levels of the organizational hierarchy. For the most part, however, the use of such technologies did not

38. Ensmenger, "The 'Question of Professionalism'".
39. Brad Cox, "There is a Silver Bullet", *Byte*, 15 (1990), p. 209.
40. Thomas Haigh, "Technology, Information and Power: Managerial Technicians in Corporate America: 1917–2000", (unpublished Ph.D. thesis, University of Pennsylvania, 2002).

contribute to the rise of a class of technical professionals capable of challenging the power of traditional management.[41]

As more and more corporations began to integrate electronic computers into their data processing operations, however, it became increasingly clear that this new technology threatened the stability of the established managerial hierarchy. Early commercial computers were large, expensive, and complex technologies that required a high level of technical competence to operate effectively. Many nontechnical managers who had adapted readily to other innovations in office technology such as complicated filing systems and tabulating machinery, were intimidated by computers – and by computer specialists. As the electronic computer became an increasingly valuable source of institutional and economic power and authority, programmers and other computer personnel emerged as influential organizational "change-agents" (to use the management terminology of the era).[42] This was particularly true of business programmers. The systems they developed often replaced, or at least substantially altered, the work of traditional white-collar employees. Traditional corporate managers, not unsurprisingly, often resented the perceived impositions of the "computer boys", regarding them as threats to their position and status.[43]

The rising power of EDP professionals did not go unnoticed by other middle-level managers. In a 1967 essay on "The Impact of Information Technology on Organizational Control", management consultant, Thomas Whisler, warned his colleagues "it seems most unlikely that one can continue to hold title to the computer without assuming and using the effective power it confers".[44] A decade earlier, Whisler and his colleague Harold Leavitt had coined the term "information technology", and had predicted that within thirty years the combination of management science and information technology would decimate the ranks of middle management and lead to the centralization of managerial control.[45] His 1967 article suggested that EDP specialists were the direct beneficiaries of such centralization, which occurred at the expense of traditional managers. He quoted one insurance executive who claimed that "There has actually been a lateral shift to the EDP manager of decision-making from other

41. JoAnne Yates, *Control Through Communication: The Rise of System in American Management* (Baltimore, 1989).

42. John Golda, "The Effects of Computer Technology on the Traditional Role of Management", (M.A. thesis, Wharton School, University of Pennsylvania, 1965), p. 34.

43. For example, see T. Alexander, "Computers Can't Solve Everything", *Fortune* (October 1969), p. 169, and Thomas Whisler, "The Impact of Information Technology on Organizational Control", in Charles Myers (ed.), *The Impact of Computers on Management* (Cambridge, MA, 1967), pp. 16–48, 44.

44. *Ibid.*, p. 44.

45. Harold Leavitt and Thomas Whisler, "Management in the 1980s", *Harvard Management Review*, 36 (1958), pp. 41–48.

department managers whose departments have been computerized." Another manager complained about the relative decline of managerial competence in relationship to computer expertise:

> The supervisor [...] has been replaced as the person with superior technical knowledge to whom the subordinates can turn for help. This aspect of supervision has been transferred, at least temporarily, to the EDP manager and programmers or systems designers involved with the programming [...] underneath, the forward planning function of almost all department managers has transferred to the EDP manager.[46]

Information technology, argued Whisler, "tends to shift and scramble the power structure of organizations [...]. The decision to locate computer responsibility in a specific part of an organization has strong implications for the relative authority and control that segment will subsequently achieve."[47]

Whisler was hardly alone in his assessment of the impending danger of an organizational power shift. In her 1971 book, *How Computers Affect Management*, Rosemary Stewart described how computer specialists mobilized the mystery of their technology to "impinge directly on a manager's job and be a threat to his security or status".[48] In his 1969 article "Computers Can't Solve Everything", Thomas Alexander emphasized the cultural differences that existed between "computer people" and business managers: "Managers [...] are typically older and tend to regard computer people either as mere technicians or as threats to their position and status – in either case they resist their presence in the halls of power."[49] Authors Porat and Vaughan listed several deprecating titles that managers used to describe their upstart rivals, including "the new theocracy", "prima donnas", "the new breed", "industrial carpetbaggers", and "other similarly unflattering titles".[50]

It is not difficult to understand why many managers came to fear and dislike computer programmers and other software specialists. In addition to the usual suspicion with which established professionals generally regarded unsolicited changes in the status quo, managers had particular reasons to resent EDP departments. The unprecedented degree of autonomy that corporate executives granted to "computer people" seemed a deliberate affront to the local authority of departmental managers. "All too often management adopts an attitude of blind faith (or at least hope) toward decisions of programmers", complained one management-oriented

46. Whisler, "The Impact of Information Technology on Organizational Control", p. 44.
47. *Ibid.*, p. 48.
48. Rosemary Stewart, *How Computers Affect Management* (Cambridge, MA, 1971), p. 196.
49. Alexander, "Computers Can't Solve Everything", p. 169.
50. Avner Porat and James Vaughan, "Computer Personnel: The New Theocracy – or Industrial Carpetbaggers", *Personnel Journal*, 48 (1968), pp. 540–543.

computer textbook.[51] As a result of the "inability or unwillingness of top management to clearly define the objectives of the computer department and how it will be utilized to the benefit of the rest of the organization", many operational managers "expect the worse and, therefore, begin to react defensively to the possibility of change".[52] The adoption of computer technology threatened to bring about a revolution in organizational structure that carried with it tangible implications for the authority of managers: "What has not been predicted, to any large degree, is the extent to which political power would be obtained by this EDP group. Top management [...] have abdicated their responsibility and let the 'computer boys' take over."[53]

As Thomas Haigh has suggested, it was during the late 1950s that the concept of management information (and computerized management information systems) was developed; by the beginning of the 1960s the computer had become not just a tool to be managed, but also a tool for management.[54] A whole host of new would-be management experts, including systems men, operations research experts, and management consultants emerged to threaten the professional authority of middle-level managers. The frequent association of "computer boys" with external consultants only compounded the resentment of regular employees.

There were other reasons why traditional managers felt threatened by computers and computer specialists. The continuous gap between the demand and supply of qualified computer personnel had in recent years pushed up their salary levels faster than those of other professionals and managers. It also provided them with considerable opportunities for horizontal mobility, either in pursuit of higher salaries or more challenging positions. These opportunities were often resented by other, less mobile employees. In addition, the unprecedented degree of autonomy that corporate executives granted to "computer people" seemed a deliberate affront to the local authority of departmental managers. In the eyes of many nontechnical managers, the personnel most closely identified with the digital computer "have been the most arrogant in their willful disregard of the nature of the manager's job. These technicians have clothed themselves in the garb of the arcane wherever they could do so, thus alienating those whom they would serve."[55]

In response to these perceived challenges to their authority, managers developed a number of interrelated responses intended to restore them to their proper roles in the organizational hierarchy. The first was to define

51. Michael Barnett, *Computer Programming in English* (New York, 1969), p. 3.
52. Porat and Vaughan, "Computer Personnel: The New Theocracy", p. 542.
53. Golda, "The Effects of Computer Technology", p. 34.
54. Haigh, "Inventing Information Systems".
55. *Datamation* editorial, "The Thoughtless Information Technologist", *Datamation*, 12 (1966), pp. 21–22, 21.

programming as an activity, and by definition programmers as professionals, in such a way as to assure it and them a subordinate role as mere technicians or service staff workers. The rhetoric of management literature reinforced the notion that computer specialists were self-interested, narrow technicians rather than future-minded, bottom-line-oriented good corporate citizens. "People close to the machine can also lose perspective", argued one computer programming "textbook" for managers. "Some of the most enthusiastic have an unfortunate knack of behaving as if the computer were a toy. The term 'addictive' comes to mind [...]."[56] Managers emphasized the youthfulness and inexperience of most programmers. They cited aptitude tests and personality profiles (often of questionably scientific validity) that suggested that computer programmers were particularly antisocial, that they "preferred to work with things rather than people", as examples of the "immaturity" of the computer professions.[57]

Another common strategy for deprecating computer professionals was directly to challenge their technical monopoly. If working with computers was in fact not all that difficult, then dedicated programming staffs were superfluous. One of the alleged advantages of the COBOL programming language frequently touted in the literature was its ability to be read, understood – and perhaps even written – by informed managers.[58] In its "Meet Susie Meyer" advertisements for its PL/1 programming language, the IBM Corporation asked its users an obviously rhetorical question: "Can a young girl with no previous programming experience find happiness handling both commercial and scientific applications, without resorting to an assembler language?" The answer, of course, was an enthusiastic "Yes!" Although the advertisement promised a "brighter future for your programmers", (who would be free to "concentrate more on the job, less on the language") it also implied a low-cost solution to the labor crisis in software. The subtext of appeals like this were non-too-subtle: If pretty little Susie Meyer, with her spunky miniskirt and utter lack of programming experience, could develop software effectively in PL/1, so could just about anyone.

Experienced managers stressed the critical differences between "real-world problems" and "EDP's version of real-world problems".[59] The assumptions about programmers embedded in the infamous McKinsey reports – that they were narrowly-technical, inexperienced, and "poorly qualified to set the course of corporate computer effort" – resonated with

56. Barnett, *Computer Programming in English*, p. 5.
57. For example, see Dallis Perry and William Cannon, "Vocational Interests of Computer Programmers", *Journal of Applied Psychology*, 51 (1967), pp. 28–34.
58. Gordon, "Review of Charles Lecht", p. 85.
59. Harry Larson, "EDP – A 20-Year Ripoff!", *Infosystems*, 21 (1974), pp. 26–30, 28.

many corporate managers.[60] They provided a convenient explanation for the burgeoning software crisis. Computer department staffs, although "they may be superbly equipped, technically speaking, to respond to management's expectations", are "seldom strategically placed (or managerially trained) – to fully assess the economics of operations or to judge operational feasibility".[61] Only the restoration of the proper balance between computer personnel and managers could save the software projects from a descent into "unprogrammed and devastating chaos".[62]

In much of the management literature of this period, computer specialists were often cast as self-interested peddlers of "whizz-bang" technologies. "In all too many cases the data processing technician does not really understand the problems of management and is merely looking for the application of his specialty."[63] The 1969 book *New Power for Management* emphasized the myopic perspective of programmers: "For instance, a technician's dream may be a sophisticated computerized accounting system; but in practice such a system may well make no major contribution to profit."[64] Others attributed to them even more Machiavellian motives: "More often than not the systems designer approaches the user with a predisposition to utilize the latest equipment or software technology – for his resumé – rather than the real benefit for the user."[65] Calling programmers the "Cosa Nostra" of the industry, the colorful former-programmer turned technology management consultant H.R.J. Grosch warned managers to "refuse to embark on grandiose or unworthy schemes, and refuse to let their recalcitrant charges waste skill, time and money on the fashionable idiocies of our racket".[66] Like many of his management-oriented colleagues, he argued that programmers needed to "accept reality, not to rebel against it". Many of the technological, managerial, and economic woes of the software industry became wrapped up in the problem of programmer management.

The idea that the so-called "software crisis" could largely be attributed to mismanagement by technicians served a dual purpose for traditional middle-level managers. First of all, it placed them solidly in the role of corporate champion. Many of the most prominent software engineering methodologies developed in the immediate post-Garmisch-conference era

60. *Datamation* Editorial, "Trouble [...] I Say Trouble, Trouble in DP City", *Datamation*, 14 (1968), p. 21, 21.
61. Herz, *New Power for Management*, p. 169.
62. Robert Boguslaw and Warren Pelton, "Steps: A Management Game for Programming Supervisors", *Datamation*, 5 (1959), pp. 13–16.
63. W.R. Walker, "MIS Mysticism (Letter to Editor)", *Business Automation*, 16 (1969), p. 8.
64. Herz, *New Power for Management*, p. 169.
65. H.L. Morgan and J.V. Soden, "Understanding MIS Failures", *Data Base*, (Winter 1973), pp. 157–171, 159.
66. Herb Grosch, "Programmers: The Industry's *Cosa Nostra*", *Datamation*, 12 (1966), p. 202.

were management-related or driven. Secondly, this particular construction of the software crisis provided an unflattering image of the computer specialists vis-à-vis management. By representing programmers as short-sighted, self-serving technicians, managers reinforced the notion that they were ill-equipped to handle "big-picture", mission-critical responsibilities. After all, according to the McKinsey reports, "Only managers can manage the computer in the best interests of the business."[67] And not just any managers would do: only those managers who had traditional business training and experience were acceptable, since "managers promoted from the programming and analysis ranks are singularly ill-adapted for management".[68]

THE STRUGGLE FOR OCCUPATIONAL TERRITORY

To many observers of computer revolution of the late twentieth century – both historians and practitioners alike – the emergence of new, managerially-oriented, "rational" solutions to the software crisis marked "a major cultural shift in the perception of programming", the welcome beginning of a new era in which software development "started to make the transition from being a craft for a long-haired programming priesthood to becoming a real engineering discipline".[69] In this conventional and essentially Chandlerian interpretation, an important but immature industry, driven by the changing economics of commercial computing and guided by well-established managerial and organizational principles, simply restructured itself along the lines of traditional industrial manufacturing. In the internal language of the software engineering discipline, an "inversion in the hardware-software cost ratio curve" occurred in the mid-1960s that clearly demanded a managerial response.[70] Put more simply, the cost of the actual computers went down at the same time that the cost of using them (developing and maintaining software) went up. By the end of the decade the expenses associated with commercial data processing were dominated by software maintenance and programmer labor rather than equipment purchases. And since the management of labor fell under the traditional domain of the middle-level manager, these managers quickly developed a deep interest in rationalizing the practices of their computer programmers.

67. McKinsey & Company, "Unlocking the Computer's Profit Potential", p. 33.
68. J.L. Ogdin, "The Mongolian Hordes versus Superprogrammer", *Infosystems*, 20 (1973), pp. 20–23, 20.
69. Martin Campbell-Kelly and William Aspray, *Computer: A History of the Information Machine* (New York, 1996), p. 201.
70. Barry Boehm, "Software and its Impact: A Quantitative Assessment", *Datamation*, 19 (1973), pp. 48–59. See also Michael Mahoney, "Software: the Self-Programming Machine", in Atsushi Akera and Fred Nebeker (eds), *From 0 to 1: An Authoritative History of Modern Computing* (New York, 2001).

For labor historians and historians of technology the story is a little more complicated. If we take seriously the claim – widely accepted in both disciplines – that technologies and technological systems represent more than just the "one best way" to accomplish a particular function but are also the embodiment of very specific social, political, and power relationships, then the ongoing debate about the software crisis assumes a much larger significance. Computer programmers are in many ways the paradigmatic "knowledge workers" of postindustrial society.[71] At the very least, they play a central role in the development of the computerized information systems that have become ubiquitous components of the modern work environment, whether office building, retail establishment, mechanic's shop, or assembly line. Surely then it is crucial that we understand the nature of the computer programmer's work, if only to understand the politics of the technologies that they build. As Shoshona Zuboff argues in her book *In the Age of the Smart Machine: The Future of Work and Power*, "computer based technologies are not neutral; they embody essential characteristics that are bound to alter the nature of work within our factories and offices, and among workers, professionals, and managers".[72] What then are the "essential characteristics" of software and software development that shape our understanding of work, identity, and power in the information technology industry (and the many industries that rely on information technology)? How can we understand the social and occupational history of the computer programmer in terms of a larger debate about the role of information technology in organizational transformation?

One possible interpretation of the burgeoning software crisis of the late 1960s and the emergence of new management-oriented solutions to the problem of software production might situate these developments within the context of a larger struggle between labor and the forces of capital. Indeed, the few scholarly treatments of software workers that do exist adopt this approach. Building on the work of Harry Braverman and David Noble, the labor historian Philip Kraft argued in his 1977 book *Programmers and Managers: The Routinization of Computer Programming in the United States*, that "programmers, systems analysts, and other software workers are experiencing efforts to break down, simplify, routinize, and standardize their own work so that it, too, can be done by machines rather than people". Cloaked in the language of progress and efficiency, the imposition of increasingly rigorous management controls on the process of programming was envisioned primarily as a means of

71. Daniel Bell, *The Coming of Post-Industrial Society* (New York, 1973).
72. Shoshana Zuboff, *In the Age of the Smart Machine: The Future of Work and Power* (New York, 1988).

disciplining and controlling a recalcitrant work force.[73] Joan Greenbaum, in her 1979 study of "Management Theory and Shopfloor Practice in Data-Processing Work" arrived at a similar conclusion.[74]

A superficial reading of the management literature of this period, with its confident claims about the ability of performance metrics, development methodologies, and automatic programming languages to reduce corporate dependence on individual programmers, might suggest that this is indeed a straightforward story of the routinization and degradation of programmer labor. Certainly, the twentieth century is replete with such stories. In fact, many of the software management methodologies proposed in late 1960s do indeed represent "elaborate efforts" that "are being made to develop ways of gradually eliminating programmers, or at least reduce their average skill levels, required training, experience, and so on".[75] Their authors would have been the first to admit it.

If computer programming had remained, as was originally intended, a form of glorified clerical labor, then the deskilling hypothesis might serve a more useful interpretive function. There is an existing literature on the routinization and feminization of clerical work.[76] As we have seen, however, computer programmers in this period generally managed not only to maintain their status and autonomy, but also improbably to extend it. What began as low-status, clerical, and feminized labor emerged as one of the most well-paid, highly romanticized, and stereotypically masculine of white-collar occupations. Writing in 1971, the occupational sociologist Enid Mumford actually lauded data processing as one area "where the philosophy of job reducers and job simplifiers – the followers of Taylor – has not been accepted".[77] More than four decades after corporate managers first began their attempts to rationalize software development along the lines of traditional manufacturing, computer programming remains a distinctively craft-oriented and idiosyncratic discipline. Although complaints about the quality and reliability of software still plague software developers – the rhetoric of crisis continues to dominate discussions about the health and future of the industry – it is clear that computer programmers in the 1960s were active participants in the struggle to define the boundaries of their own professional competence and authority.

An alternative interpretation might view this history in terms of the

73. Philip Kraft, *Programmers and Managers: The Routinization of Computer Programming in the United States* (New York, 1977), p. 32.
74. Joan Greenbaum, *In the Name of Efficiency: Management Theory and Shopfloor Practice in Data-Processing Work* (Philadelphia, PA, 1979).
75. Kraft, *Programmers and Managers*, p. 26.
76. Sharon Strom, *Beyond the Typewriter: Gender, Class and the Origins of Modern American Office Work, 1900–1930* (Urbana, IL, 1992); Margery Davies, *Woman's Place is at the Typewriter: Office Work and Office Workers, 1870–1930* (Philadelphia, PA, 1982).
77. Enid Mumford, *Job Satisfaction: A Study of Computer Specialists* (London, 1972).

professionalization literature. During the 1950s and 1960s many white-collar occupations attempted to professionalize, and computer programmers were no exception.[78] They established professional societies, codes of ethics, and certification and curriculum standards.[79] Belonging to a profession provided an individual with a "monopoly of competence", the control over a valuable skill that was readily transferable from organization to organization.[80] Professionalism provided a means of excluding undesirables and competitors; it assured basic standards of quality and reliability; it provided a certain degree of protection from the fluctuations of the labor market; and it was seen by many workers as a means of advancement into the middle class.[81] Programmers in particular saw professionalism as means of distinguishing themselves from "coders" or other "mere technicians". Corporate managers generally embraced the concept of professionalism. It appeared to provide a familiar solution to the increasingly complex problems of programmer management: "The concept of professionalism", argued one personnel research journal from the early 1970s, "affords a business-like answer to the existing and future computer skills market".[82] The rhetoric of professionalism was ideologically neutral, and appealed to a wide variety of individuals and interest groups. Professionalization was one of several widely adopted strategies for dealing with the software crisis.[83]

Thinking in terms of professionalization provides several benefits. It allows us to locate the history of computer programming in a familiar literature, and it provides a number of useful explanatory devices. One of the most useful is the sociologist Andrew Abbott's "ecological" model for understanding professional change and development. In *The Systems of Professions: An Essay on the Division of Expert Labor*, Abbott describes the "jurisdictional struggles" that occur among groups of professionals struggling for control over a particular occupational territory.[84] In Abbott's model, professions are fluid organisms able to adapt and expand when occupational niches become available to them and to respond and defend themselves when their particular territory becomes threatened by

78. Harold Wilensky, "The Professionalization of Everyone?", *American Journal of Sociology*, 70 (1964), pp. 137–158.
79. Ensmenger, "The 'Question of Professionalism'".
80. Magali Sarfatti Larson, *The Rise of Professionalism: A Sociological Analysis* (Berkeley, CA, 1977).
81. Robert Zussman, *Mechanics of the Middle Class: Work and Politics Among American Engineers* (Berkeley, CA, 1985).
82. *Personnel Journal* Editorial, "Professionalism Termed Key to Computer Personnel Situation", *Personnel Journal*, 51 (February 1971), pp. 156–157.
83. Nathan Ensmenger, "From 'Black Art' to Industrial Disciple: The Software Crisis and the Management of Programmers", (unpublished Ph.D thesis, University of Pennsylvania, 2001).
84. Andrew Abbott, *The Systems of Professions: An Essay on the Division of Expert Labor* (Chicago, IL, 1988).

competitors. Disruptive new technologies often allow for the creation of new niches or the expansion of existing occupational territory. It is clear that this is in part what happens with the electronic computer in the late 1950s. As the electronic digital computer technology became an increasingly important tool for corporate control and communication, existing networks of power and authority were uncomfortably disrupted. The conflicting needs and agendas of users, manufacturers, managers, and programmers all became wrapped up in highly public struggle for control over the occupational territory opened up by the technology of computing.

This professionalization narrative is not entirely satisfactory, however. Despite their best efforts to establish the institutional structures of a profession, computer programmers were never able to achieve widespread professional recognition. They were unable, for example, to develop two of the most defining characteristics of a profession: control over entry into the profession and the adoption of a shared body of abstract occupational knowledge – a "hard core of mutual understanding" – common across the entire occupational community. They failed to convince employers sufficiently of the value of professionalism, and were often divided among themselves over issues involving academic standards and certification requirements. Complaints about the lack of professional standards among computer programmers continue to play a central role in discussions about the nature and causes of the software crisis. Despite the widespread adoption of the rhetoric of software engineering, most computer programmers are not engineers and would not identify themselves as such. Although the "question of professionalism" continues to be a very live issue in the programming community, in general computer programmers are not in general considered to be professionals.[85]

So if not professionals, managers, or clerical support staff, what exactly are computer programmers? How can does their unique history tell us about larger patterns in work practices and the organizational of labor in the late late twentieth century?

Perhaps the most useful way to think about the computer programmer is as a technician. As the organizational theorist Stephen Barley has suggested, technicians are a relatively recent addition to the pantheon of occupations.[86] Although technicians do not fit easily into the interpretive framework of either labor history or the sociology of professions, they represent the fastest growing sector of the American labor force. They include such occupations as radiological technicians, science technicians, engineering technicians, and medical technicians. Their work often

85. Ensmenger, "The 'Question of Professionalism'".

86. Stephen Barley, "Technicians in the Workplace: Ethnographic Evidence for Bringing Work Into Organization Studies", *Administrative Science Quarterly*, 41 (1996), pp. 404–441.

transgresses traditional occupational boundaries; according to Barley, technicians "often wear white collars, carry briefcases, and conduct sophisticated scientific and mathematical analyses. Yet they use tools, work with their hands, make objects, repair equipment, and, from time to time, get dirty."[87] They are often – albeit at times grudgingly – granted a great deal of autonomy by their employers.[88] Like computer programmers, technicians occupy an ambiguous occupational space that is difficult to categorize.

Also like computer programmers, technicians serve as mediators between the technological and social architectures of the organization. Technicians are often responsible for building, repairing, and monitoring the complex systems that keep a company running. Because they play a support role that is tangential to the core business of the organization and generally possess skills radically different from those of their colleagues, they are often seen as foreigners to the worksite.[89] Traditional employees often resent their dependence on technicians and consider them insufficiently subservient.[90]. Like the "computer boys" of the late 1960s, technicians often wield power disproportionate to their official position in the occupational hierarchy.

There are a number of other similarities between Barley's description of technicians and the history of the computer programmer. Although they are generally well-educated and rely heavily on scientific or engineering training, technicians also value intuition and craft knowledge. They tend to learn on the job, rather than from formal academic or vocational training programs. They make extensive use of social networks and community-based systems of information exchange. Their expertise is often local and idiosyncratic and difficult to communicate or define as a set of abstract principles.[91]

It seems clear from these descriptions that computer programmers can be considered as a type of technician. In fact, this seems to be the most useful way to make connections between software workers and other forms of technical labor. It captures the tension inherent in the practices of software development: the curious coexistence of high technology and artisanal sensibilities; the inability of programmers to conform to conventional professional, scientific, or engineering categories; the persistent attempts by corporate managers to restructure software

87. *Ibid.*, p. 412.
88. Stacia Zabusky and Stephen Barley, "Redefining Success: Ethnographic Observations on the Careers of Technicians", in Paul Osterman (ed.), *Broken Ladders: White Collar Careers* (Oxford, 1996), pp. 185–214.
89. Barley, "Technicians in the Workplace", p. 422.
90. *Ibid.*, p. 430.
91. *Ibid.*, p. 427.

development along the lines of traditional manufacturing; the remarkable persistence of the forty-year-old software crisis.

By looking beyond simplistic explanations of computer programmers as either degraded "software factory" workers or failed software engineers – or as *sui generis* exceptions to larger historical patterns – we can recapture the broader relevance of the history of information technology to social and labor history. Thinking of computer programmers as technicians allows us to locate them in a larger historical context. They are both like and unlike traditional workers, and both the similarities and differences are revealing.

At the very least, the history of computer programmers provides a reinterpretation of what has generally been treated as a purely technical debate; it suggests that corporate workers, managers, and computer programmers have been active participants in shaping the technology of electronic computing. Like any new technological innovation, the computer could not simply be inserted, unchanged and unnoticed, into the well-established social, technological, and political systems that comprised modern corporate and academic organizations. Just as the computer itself was gradually reconstructed, in response to a changing social and technical environment, from a scientific and military instrument into a mechanism for corporate control and communication, modern businesses and universities had to adapt themselves to the presence of a powerful new technology. Over the course of the 1950s and 1960s, the identity of the computer programmer was continually invented and reinvented in response to a changing social and technical environment. Embedded into all of the major technical innovations of period was a particular model of what the users/programmers of these inventions should look like. Was the idealized computer programmer a routinized laborer in a Taylorized "software factory" or a skilled, autonomous professional? Should programmers base their occupational identity on the model of the engineer/scientist or the certified public accountant? Should they emphasize craft technique or abstract knowledge? Did they need to be college educated or simply a vocational school graduate? Should they be male or female? The answers to each of these questions had significant implications for the role of electronic computing – and of computing professionals – in modern corporate and academic organizations. It is no wonder, therefore, that they were not readily resolved in this, or for that matter any other, period in the history of computing.

EPILOGUE

In the years since the 1960s the software industry has only continued to expand. A recent study by the Bureau of Labor Statistics shows that, since 1972, employment in the computer services industries (which includes

software and associated services) has grown 300 per cent.[92] There are now at least 1.9 million computer services workers in the United States alone. Even discounting the recent boom (and subsequent bust) in the information technology sector, software and its associated services remains one of the largest and fastest growing industries in the United States.

The obvious importance of the industry to the national and global economy suggests that computer programmers are a worthy object of continued study. The recent debate over the information technology worker supply, which has implications for a wide range of funding, education, and immigration policy issues, revealed a surprising lack of basic information about the size and structure of the information technology labor market.[93] It is clear, however, that many of the institutional structures that continue to inform our understanding of information technology and information technology workers – academic, professional, and technological – took their shape in the period discussed in this paper. In many ways the basic framework of the debate about the software crisis has remained essentially unchanged in the decades since the 1960s. In an industry characterized by change, the rhetoric of crisis has proven remarkably persistent.

In recent years the debate about the software crisis has gone global. Competition from Asia, both in terms of an influx of Asian programmers entering the United States on H1-B and L1 visas, as well as the movement of software development projects to offshore "software factories", has created new tensions within the computing community. Many of the questions about certification, professionalization, and workplace control that dominated discussions about software workers in the 1960s have re-emerged, this time around couched in terms of fears of foreign competition and national security. These are discussions that have occurred in the past, but always in regard to blue-collar manufacturing jobs, not skilled white-collar occupations. Many of the organizational tensions associated with computerization projects have been further complicated by questions of race and nationalism. There has been a renewed interest in unionization among information technology workers, a development that had previously been strongly resisted by both employers and aspiring technical professionals.[94]

Because computer programmers confound so many of the traditional

92. William Goodman, "The Software and Engineering Industries: Threatened by Technological Change?", unpublished technical report (1996).

93. Peter Freeman and William Aspray, *The Supply of Information Technology Workers in the United States* (Washington DC, 1999).

94. From the 1960s onward, movements to unionize computer programmers emerge periodically, but none acquired any significant momentum. Like many aspiring professionals, programmers generally resisted unionization efforts, and the constantly expanding demand for new programmers discouraged any attempts to erect barriers of entry to the occupation.

categories of historical and sociological analysis, they suggest new ways to re-evaluate the role of the computer in late twentieth-century society. Much of the literature on the computer, from the earliest days of computing to the present, has focused on its "revolutionary" potential. And yet, more than thirty years after the first NATO Conference on Software Engineering, advocates of a more industrial approach to software development still complain that the "vast majority of computer code is still handcrafted from raw programming languages by artisans using techniques they neither measure nor are able to repeat consistently".[95] The study of the computer in the context of work practices, occupational conflict, and organizational politics allows us to explore not only change but continuity, and to link the history of the computer to a larger body of labor and social history, as well as to contemporary issues of concern to a broad range of audiences.

95. W. Gibbs, "Software's Chronic Crisis", *Scientific American*, (September 1994), p. 86.