Software as Labor Process

Nathan Ensmenger

University of Pennsylvania
Philadelphia
USA
nathanen@sas.upenn.edu

&

William Aspray

Computing Research Association
1100 17th St. NW # 507
Washington, D.C. 20036
USA
aspray@cra.org

The Software Crisis and the "Labor Problem" in Programming …

For almost as long as there has been software, there has been a software crisis.[1] Laments about the inability of software developers to produce products on time, within budget, and of acceptable quality and reliability have been a staple of industry literature from the early decades of commercial computing to the present. In an industry characterized by rapid change and innovation, the rhetoric of the crisis has proven remarkably persistent. The acute shortage of programmers that caused "software turmoil" in the early 1960s has reappeared as a "world-wide shortage of information technology workers"[2] in the 1990s. Thirty years after the first NATO Conference on Software Engineering, advocates of an 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."[3] Corporate managers and government officials release ominous warnings about the desperate state of the software industry with almost ritualistic regularity. The Y2K crisis is only the most recent manifestation of the software industry's apparent predilection for apocalyptic rhetoric.

[1] The Oxford English Dictionary identifies the first use of the word "software" in a 1960 article in the Communications of the ACM. By early 1962 Daniel McCracken was already lamenting the "software turmoil" that threatened to "set the software art back several years." ("The Software Turmoil: Nine Predictions for '62," *Datamation* 8/1 (1962)). References to the "Gap in Programming Support" appear even earlier (Robert Patrick, "The Gap in Programming Support," *Datamation* 7/5 (1961)).

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

[3] W. Gibbs, "Software's Chronic Crisis," *Scientific American*, September 1994.

References to the chronic "software crisis" are so ubiquitous that it is possible to lose sight of their historical origins and significance. Specific claims about the nature and extent of the crisis can be used, however, as a lens through which to examine broader issues in the history of software. In this paper we discuss the historical construction of the software crisis as a crisis of programming labor. We argue that many of the crucial innovations in modern software development—high-level programming languages, structured programming techniques, and software engineering methodologies, for example—reflect corporate concerns about the supply, training, and management of programmers. We suggest that the labor crisis in software threatened the viability of software as an economic activity; that it originated in the failure of software as a reliable artifact; that it stimulated efforts to establish the discipline of software engineering; and that it undermined the legitimacy of software as scientific and engineering practice.

Because the labor crisis in programming has been so widely referred to and written about, it serves as an ideal launching pad for an exploration of other, less familiar issues in the labor history of software. In our paper, for example, we reexamine the perennial debate about programming training and management in terms of contemporary debates about socially constructed notions of "skill," "knowledge," and "productivity." We argue that the changing role of women in software reflects larger developments in the professional fortunes and occupational identity of programmers. We describe the role of institutions such as unions, professional associations, and the government in the shaping of software development practices. Our goal is to suggest some directions for further scholarship in what we regard as an essential element of the history of computing.

It should be noted, however, that the study of labor processes presents serious methodological challenges to historians. Conventional interpretations of the software crisis are often based on the software management literature, which is typically biased towards the perspective of employers and managers. This literature also tends to reflect an ideal rather than reality. The voice of the worker is rarely represented in the types of sources that we as historians are accustomed to dealing with. We know very little about the experiences and attitudes of the typical software developer, or about the craft practices and "shop floor" activities of programmers.[4] There is almost no secondary literature available on this subject. Our discussion of software as labor process is therefore more historical than historiographical, to exemplify some of the historical issues that deserve further attention. To repeat, we try to show some of the historical questions mainly by sketching some of the history. In an attempt to counter the traditional bias towards management perspectives, we deliberately chose to construct our narrative around an eclectic sampling of sources and perspectives. The ongoing debate about the software labor crisis has been passionate, contentious, and replete with ambiguities and self-contradictions. The fact that the community of software workers included both former theoretical physicists and Helen Gurley Brown's "Cosmo Girls" is not

[4] Michael Mahoney has more fully described these difficulties in his "The History of Computing in the History of Technology," *Annals of the History of Computing* 10/2 (1988): 113-25.

an incidental curiosity; it is an essential element of the labor history of software.[5] In this paper we hope to convey the sense of excitement and drama experienced by early software workers. Our hope is that in doing so we will encourage historians to explore the rich history of software labor, and perhaps to make use of some previously undiscovered resources. We understand that the story we are telling is an entirely American story, and that when one looks at the international scene, the relevant issues may be different.

The Acute Shortage of Programmers …

Historically, the software crisis has often been portrayed explicitly as a problem of programming labor. In 1962 the industry journal *Datamation* warned of a "gap in programming support" that threatened to "get worse in the next several years before it gets better."[6] Several decades later Bruce Webster declared that the "heart of the real software crisis … [is that] there is more software to be developed than there are capable developers to do it. Demand will continue to outstrip supply for the foreseeable future. Hence, more and more software will be behind schedule, over budget, underpowered, and of poor quality—and there's nothing we can do about it."[7] The problem was not so much a lack of programmers *per se*; what the industry was really worried about was a shortage of experienced, *capable* developers. That there was little agreement within the software community about who exactly qualified as an experienced, capable developer only served to emphasize their real or perceived rarity.

The potential shortage of programmers materialized as early as 1954, when the first-ever Conference on Training Personnel for the Computing Machine Field was held at Wayne University.[8] At the time it was generally felt that mathematical knowledge was an essential component of programming expertise. Several speakers noted that although in 1951 there were only 2,000 Ph.D. mathematicians in the nation, there were already 2,000-4,000 jobs available in computing, and the annual demand for programmers was expected to double.[9] E. P. Little of Wayne University warned that "estimates of manpower needs for computer applications … [are] astounding compared to the facilities for training people for this work." [10] W. H. Wilson of General Motors observed "a universal feeling that there is a definite

[5] Helen Gurley Brown was the controversial editor of *Cosmopolitan Magazine* and the author of the 1962 *Sex and the Single Woman*. Her "Cosmo Girls" were modern, hard working, and sexuality aggressive.

[6] Robert Patrick, "The Gap in Programming Support," *Datamation* 7/5 (1961).

[7] Bruce Webster, "The Real Software Crisis," *Byte Magazine* 21/1 (1996).

[8] Wayne University had an active, early university computing program, strengthened by its partnerships with the local Detroit industries; thus it was a logical choice to host this training conference. The conference provides a good snapshot of the supply of computing workers at the time.

[9] *Manpower Resources in Mathematics*. National Science Foundation and the Department of Labor, Bureau of Labor Statistics, 1951.

[10] Arvid W. Jacobson, ed., *Proceedings of the First Conference on Training Personnel for the Computing Machine Field,* held at Wayne University, Detroit, Michigan, June 22 and 23, 1954 (Detroit, 1955), 79.

shortage of technically trained people in the computer field."[11] There was little hope current production could meet expected demand.

The largest employer of programmers in this period was the System Development Corporation (SDC), the RAND Corporation spin-off responsible for developing the SAGE missile defense system. SDC employed seven hundred programmers in the late 1950s, and several thousand by the early 1960s. Like many large software development companies in this period, SDC was forced to train most of its own programmers. One manager at SDC noted proudly that, although it was estimated in the 1954 that all of the computer manufacturers *combined* could only provide 2500 student weeks of instruction annually, three years later "during a comparable period, SDC devoted more than 10,000 student weeks to instructing its own personnel to program."[12] Between 1956 and 1961 the company trained 7,000 programmers and systems analysts.

Not only did SDC train more programmers than anyone else in this period ("We trained the industry!")[13], it also propagated its own systems-oriented approach to software development. The SAGE project was unusual in that it was a large, monolithic effort involving thousands of programmers and mission-critical systems. The only other projects of comparable size and complexity at this time were being undertaken at IBM. Most other commercial software developers were working on smaller, more self-contained efforts requiring far fewer programmers. Programmers at these installations worked on multiple projects involving a diverse range of business problems. They often participated in every aspect of system development, from requirements gathering to system design to implementation; consequently they experienced more intellectual stimulation and satisfaction from their work.[14] Large government-oriented employers like SDC and IBM may have trained the majority of programmers in the 1950s, but they had difficulty keeping them around.

As the market for commercial computers expanded in the 1960s, the demand for experienced programmers increased rapidly. In 1962 the editors of *Datamation* declared that "first on anyone's checklist of professional problems is the manpower shortage of both trained and even untrained programmers, operators, logical designers and engineers in a variety of flavors."[15] Five years later, "one of the prime areas of concern" to electronic data processing (edp) managers was still "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."[16] A widely quoted AFIPS study from 1967 noted that although there were already 100,000 programmers, there was an immediate need for at least 50,000 more.[17] "Competition for programmers has driven salaries up so fast," warned a contemporary article in *Fortune* magazine, "that programming has become proba-

[11] Arvid W. Jacobson, 21.

[12] T. C. Rowan, "The Recruiting and Training of Programmers," *Datamation* 4/3 (1958).

[13] Claude Baum, *The Systems Builders: The Story of SDC* (Santa Monica, 1981), 47.

[14] 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), 89.

[15] Editorial, "Editor's Readout: A Long View of a Myopic Problem," *Datamation* 8/5 (1962).

[16] Richard Tanaka, "Fee or Free Software," *Datamation* 13/10 (1967).

[17] Quoted in Edward Markham, "Selecting a Private EDP School," *Datamation* 14/5 (1968).

bly the country's highest paid technological occupation. ... Even so, some companies can't find experienced programmers at any price."[18] At one point the so-called "population problem" in software became so desperate that service bureaus in New York farmed out programming work to inmates at the nearby Sing-Sing prison, promising them permanent positions pending their release![19]

The acute shortage of programming labor was not entirely alleviated by the increased production of programmers. In fact, a 1968 study by the ACM SIGCPR (Special Interest Group on Computer Personnel Research) warned of a growing *oversupply* of a certain undesirable species of software specialist: "The ranks of the computer world are being swelled by growing hordes of programmers, systems analysts and related personnel. Educational, performance and professional standards are virtually nonexistent and confusion growths rampant in selecting, training, and assigning people to do jobs."[20] It quickly became apparent that certain programmers were much more productive than others. An early study at IBM suggested that exceptional programmers were ten times more efficient than their merely average colleagues.[21] The alleged 10:1 performance ratio quickly became firmly embedded in the cultural wisdom of the industry. And so the fundamental question facing employers was not so much "where can I hire a programmer" as "where can I hire an exceptional programmer." This of course begs the question of what exactly constituted exceptional programming ability, but we will return to that issue. For the time being it is enough to point out that, like it or not, many large software corporations in this period were forced to underwrite "full scale training efforts, not because they want to do it, but because they have found it to be an absolute necessary adjunct to the operation of their business."[22]

Many employers were anxious to produce better standards for training and curriculum, but it was unclear to whom they should turn for guidance. In the late 1940s and early 1950s, computers were generally used as mathematical instruments. It was not inappropriate, therefore, to require of programmers formal mathematical training and a university education. By the middle of the 1950s, as commercial computing emerged, it was increasingly business-oriented. The results of a survey presented at the 1954 Wayne University conference reflect this fundamental shift: although only 5% of the computers in operation at that time were used in business, when the machines on order were considered the number rose to 16%.[23] The university computer training programs that focused on formal logic and numerical analysis became increasingly out-of-touch with the needs of business. The authors of a 1959 Price-Waterhouse study on "Business Experience with Electronic Computing" suggested that mathematics training had little to do with programming ability:

[18] Gene Bylinsky, "Help Wanted: 50,000 Programmers," Fortune (March 1967), 141.

[19] News Brief, "First Programmer Class at Sing Sing Graduates," *Datamation* 14/6 (1968).

[20] H. Sackman, "Conference on Personnel Research," *Datamation* 14/7 (1968).

[21] H. Sackman, W. J. Erikson, and E. E. Grant, "Exploratory Experimental Studies Comparing Online and Offline Programming Performance," *Communications of the ACM* 11/1 (1968): 3-11.

[22] James Saxon, "Programming Training: A Workable Approach," *Datamation* 9/12 (1963): 48.

[23] William Aspray, "The Supply of Information Technology Workers, Higher Education, and Computing Research: A History of Policy and Practice in the United States," in *The International History of Information Technology Policy*, ed. Richard Coopey (Oxford, forthcoming).

> Because the background of the early programmers was acquired mainly in mathematics or other scientific fields, they were used to dealing with well-formulated problems and they delighted in a sophisticated approach to coding their solutions. … When they applied their talents to the more sprawling problems of business, they often tended to underestimate the complexities and many of their solutions turned out to be oversimplifications. Most people connected with electronic computers in the early days will remember the one or two page flow charts which were supposed to cover the intricacies of the accounting aspects of a company's operations.[24]

The mismatch between university training and the needs of the corporation was in part a function of the institutional history of university computing centers. Most of these were originally housed in engineering departments—and were therefore more machine-oriented than programming proficient—or functioned as service bureaus for traditional academic departments. These service bureaus generally focused on scientific applications, heavily dependent on mathematics and generally coded in a scientific programming language such as FORTRAN. Students educated in this environment tended to absorb the academicians' traditional disdain for practical application—an attitude probably not often well received by potential corporate employers. As Richard Hamming pointed out in his 1968 Turing Award Lecture, "Their experience is that graduates in our programs seem to be mainly interested in playing games, making fancy programs that really do not work, writing trick programs, etc., and are unable to discipline their own efforts so that what they say they will do gets done on time and in practical form."[25] The tension between the theoretical orientation of academic computer specialists and the practical demands of industry employers served to exacerbate the perceived shortage of experienced business programmers.[26]

The relatively small number of colleges and universities that did offer some form of practical programming experience were unable to provide trained programmers in anywhere near the quantities required by industry. As a result, aspiring software personnel often pursued alternative forms of vocational training. Some were recruited for in-house instruction programs provided by their employers. IBM provided programming training services to many of its clients. Others enrolled in the numerous private edp training schools that began to appear in the mid-1960s. These schools were generally profit-oriented enterprises more interested in quantity than quality. For many of them the "only meaningful entrance requirements are a high school diploma, 18 years of age ... and the ability to pay."[27] The more legitimate schools oriented their curricula towards the requirements of industry. The vocational schools suffered from many of the same problems that plagued the universities: a shortage of experienced instructors, the lack of established standards and curricula, and general uncertainty about what skills and aptitudes made for a qualified programmer. "Could you answer for me the question as to what in the eyes of industry constitutes a 'qualified' programmer?"

[24] Conway (n. 14 above), 82.

[25] Richard Hamming, "One Man's View of Computer Science," in *ACM Turing Award Lectures: The First Twenty Years, 1966-1985* (New York, 1987), 207-18.

[26] This seems to be as true in the 1990s as it was in the 1960s. See Gibbs (n. 3 above).

[27] Edward Markham, "EDP Schools—An Inside View," *Datamation* 14/4 (1968): 22.

pleaded one *Datamation* reader: "What education, experience, etc. are considered to satisfy the 'qualified' status?"[28] The problem was not only that the universities and vocational schools could not provide the type of educational experience that interested corporate employers; the real issue was that most corporations were simply not at all sure what they were looking for.

Wrapped up in all of the debates about the labor shortage in software are a series of fascinating questions about the essential nature of programming expertise. Is programming aptitude an innate ability or can it be acquired? What skills and abilities distinguish the exceptional programmer from his merely average colleagues? Is it more important for the programmer and analyst to understand the business or the technology? Questions like these inspired a series of psychological and personnel studies aimed at understanding the minds and motivations of programmers.

In the late 1950s and early 1960s it was not uncommon for programmers to refer to what they did as more of an art than a science. Many would have agreed with Carl Reynolds, the president of the Computer Usage Development Corporation, when he declared that "There isn't an ideal programmer any more than there is an ideal writer. All sorts of people, from divinity to mathematics students to music and romance-language majors have gravitated to programming."[29] Employers were frustrated by the inability of standard selection mechanisms to tangibly assist in the recruitment and training of programmers.[30] What they wanted was a litmus test for programming aptitude. Anecdotal evidence suggested that there must be some psychological or intelligence factors that correlated with programming ability. When this turned out not to be related to mathematics (or chess or musical ability, the other popular candidates[31]), employers turned to industrial psychologists for alternative measures. The IBM Programmer Aptitude Test (PAT), developed in 1955, correlated performance in training programs with subsequent performance ratings by project managers and served for many years as a *de facto* industry standard. Although many personnel departments used the IBM PAT as a sort of primitive filtering method, for the most part these early attempts at empirical research proved remarkably inconclusive. A review of the 1950s literature on the selection of computer programmers identified only those skills and characteristics that would have been assets in any white-collar occupation: the ability to think logically, to work under pressure, and to get along with people; a retentive memory, the desire to see a problem through to completion; careful attention to detail. The only surprising result was that "majoring in mathematics was not found to be significantly related to performance as a programmer!"[32] Gerald Weinberg, the outspoken author of *The Psychology of Computer Programming*, spoke for many when he argued that "nobody has ever been able to demonstrate

[28] John Callahan, "Letter to the editor," *Datamation* 7/3 (1961).

[29] Carl Reynolds, quoted in Bylinsky (n. 18 above), 143.

[30] John Hanke, William Boast, and John Fellers, "Education and Training of a Business Programmer," *Journal of Data Management* 3/6 (1965).

[31] Joseph O'Shields, "Selection of EDP Personnel," *Personnel Journal* 44/9 (1965); Dean Dauw, "Vocational Interests of Highly Creative Computer Personnel," *Personnel Journal* 46/10 (1967).

[32] W. J. McNamara and J. L. Hughes, "A Review of Research on the Selection of Computer Programmers," *Personnel Psychology* 14/1 (Spring 1961), 41-2.

that any of the various ‘programmer’s aptitude’ tests was worth the money it cost for printing.”[33] More than four decades after the first Conference on Training Personnel for the Computing Machine Field, one project manager confessed that “The conclusion I have reluctantly come to after more than 20 years of software development is this: Excellent developers, like excellent musicians and artists, are born, not made.”[34] Although at this point we know very little about the historical construction of notions of programmer skill and ability, it seems clear that these are issues of interest not only to historians, but also to contemporary observers and participants.

Programmers as Professionals …

Many software personnel were keenly aware of the crisis of labor and the tension it was producing for their industry and profession, as well as for their own individual careers. Calling computer programmers the “Cosa Nostra” of data processing, industry pundit Herbert Grosch accused software professionals (himself included) of being “at once the most unmanageable and the most poorly managed specialism in our society. Actors and artists pale by comparison. Only pure mathematicians are as cantankerous, and it’s a calamity that so many of them get recruited [as programmers] by simplistic personnel men.”[35] Although computer specialists in general were appreciative of the short-term benefits of the software labor shortage (in terms of above average salaries and plentiful opportunities for occupational mobility), many believed that a continued crisis threatened the long-term stability and reputation of their industry and profession. “With a mounting tide of inexperienced programmers, new-born consultants, and the untutored outer circle of controllers and accountants all assuming greater technical responsibility, a need for qualification of competence is clearly apparent.”[36] The inability of the software community to provide its own solution to certification problem within edp, warned some observers, “will result in a solution imposed from without. In several fields, the lack of professional and industrial standards has prompted the government to establish standards.”[37]

Computer programmers in particular were worried that an influx of the kind of “narrow, semi-literate technicians”[38] put out by vocational schools and junior colleges would undermine their claims to professional legitimacy. The lack of established certification standards rankled some aspiring software professionals. “As long as anyone with ten dollars can join the ACM (Association for Computing Machinery) and proclaim himself a professional computer expert,” it would be impossible to “guarantee the public a minimum level of competence in anyone

[33] Gerald Weinberg, *The Psychology of Computer Programming* (New York, 1971); William Ledbetter, “Programming Aptitude: How Significant is It?” *Personnel Journal* 54/3 (1975).

[34] Bruce Webster, “The Real Software Crisis,” *Byte Magazine* 21/1 (1996).

[35] Herb Grosch, “Programmers: The Industry’s Cosa Nostra,” *Datamation* 12/10 (1966).

[36] Editorial, “Editor’s Readout: The Certified Public Programmer,” *Datamation* 8/3 (1962).

[37] David Ross, “Certification and Accreditation,” *Datamation* 14/9 (1968).

[38] L. Fulkerson, “Should there be a CS Undergraduate Program? (letter to editor),” *Communications of the ACM* 10/3 (1967).

who is permitted to claim membership in the profession."[39] Others worried about incursions by other, more established professions into what software workers regarded as their own proprietary occupational territory: "We can wait for the CPA types to find out the tricks of our trade, train a substantial number of their younger sub-alterns in machines and programming languages, and take over the task. Or we can establish a parallel license, team up with the CPA's for accounting and auditing tasks, and work in other directions independently."[40] In the sociological literature of the era, jurisdictional control over training and certification was presented as the *sine qua non* of professionalism.[41] Like many white-collar workers in this period, software personnel self-consciously attempted to replicate the institutional structures of the established professions.

One of the obstacles faced by the various certification committees that were established in the 1960s, however, was the general lack of agreement about what made for a good programmer: "At present, there is no established mechanism to qualify even the qualifiers."[42] A second obstacle was the great diversity of background within the existing software community. "Professional programming is fortunately wide open. In what other field are you likely to find a Ph.D. and a person whose education stopped at the high school level working as equals on the same difficult technical problem?"[43] No single certification program seemed able to reflect the diverse needs of the software community. When the National Machine Accountants Association announced its first business data processing certificate program in 1962, its efforts were greeted with deafening silence by more academically oriented groups such as the ACM.[44] In a similar manner, programs that required college-level degrees or formal mathematical training were rejected by the thousands of otherwise qualified and experienced programmers who would thereby be disqualified from working in their chosen profession. For whatever reason, despite numerous attempts by various groups to impose standard criteria for the education and certification of programmers, software specialists were never able to establish effective control over entry into their profession. In the words of one cynical observer, the lack of established certification standards unfortunately indicated that none of the "industry's widely publicized upcoming incompetents would find their accession to financial stardom impeded by the need for specific qualification such as the passing of a reasonable test of competency."[45]

Concerns about the future of their occupation weighed heavily on the minds of many programmers. What was the appropriate career path for a software worker? "There is a tendency," suggested the ACM SIGCPR, "for programming to be a 'dead-end' profession for many individuals, who, no matter how good they are as programmers, will never make the transition into a supervisory slot. And, in too

[39] Daniel McCracken, "The Human Side of Computing," *Datamation* 7/1 (1961): 10.

[40] Herb Grosch, "Computer People and their Culture," *Datamation* 7/10 (1961): 51.

[41] Harold Wilensky, "The Professionalization of Everyone?" *American Journal of Sociology* 70/2 (1964).

[42] Datamation Editorial (n. 36 above).

[43] Alex Orden, "The Emergence of a Profession," *Communications of the ACM* 10/3 (1967): 146.

[44] Datamation Report, "DP Certification Program Announced by NMAA," *Datamation* 8/3 (1962); David Ross, "Certification and Accreditation," *Datamation* 14/9 (1968).

[45] Editorial, "Editor's Readout: The Certified Public Programmer," *Datamation* 8/3 (1962).

many instances this is the only road to advancement."[46] Whereas traditional engineers were often able (and in fact expected) to climb the corporate ladder into management positions, programmers were often denied this opportunity.[47] It was not clear to many corporate employers how the skills possessed by programmers would map onto the skills required for management. Part of the problem was the lack of a uniform programmer "profile." There was no "typical" programmer. The educational and occupational experience of programmers varied dramatically from individual to individual and workplace to workplace. There was vast gulf, for example, "between the systems programmers—who must tame the beast the computer designers build—and the applications programmers—who must then train the tamed beast to perform for the users."[48] It was possible for two programmers sitting side by side—and managed by the same data processing manager and hired by the same personnel administrator—to be working on entirely different types of project each requiring distinctly different sets of skills and experience.

In the 1950s many programming recruits were migrants from other more traditional scientific and engineering disciplines. For many of these well educated "converts," the move to a new career posed personal and professional challenges. They were fascinated by computers but were wary of abandoning established careers for an uncertain and immature industry. Edsger Dikjstra, in his 1972 ACM Turing Award Lecture entitled "The Humble Programmer," described the dilemmas he faced while deciding to transition from theoretical physics to professional programming:

> ... I had to make up my mind, either to stop programming and become a real, respectable theoretical physicist, or to carry my study of physics to formal completion only, with a minimum of effort, and to become … what? A programmer? But was that a respectable profession? After all what was programming? Where was the sound body of knowledge that could support it as an intellectually respectable discipline? I remember quite vividly how I envied my hardware colleagues, who, when asked about their professional competence, could at least point out that they knew everything about vacuum tubes, amplifiers and the rest, whereas I felt that, when faced with that questions, I would stand empty-handed.[49]

Dijkstra and his fellow erstwhile engineers and scientists formed the vanguard of the nascent programming profession. They possessed many of the skills and credentials required for corporate advancement. It was not difficult for these men to imagine themselves following a career path similar to that of their more traditional colleagues. It was this first generation of university-trained programmers who felt particularly threatened by the hordes of new software personnel who entered the profession in the 1960s. The composition of the programming workforce was changing, and was becoming more specialized and diverse. Gone were the days "when programmers taken as a group were overpaid … programming in general, and for a user company in particular, is a dead-end proposition, unless there is true

[46] Datamation Report, "The Computer Personnel Research Group," *Datamation* 9/1 (1963): 38.

[47] Louis Kaufman and Richard Smith, "Let's Get Computer Personnel on the Management Team," *Training and Development Journal* (December 1966).

[48] Christopher Shaw, "Programming Schisms," *Datamation* 8/9 (1962).

[49] Edsger Dijkstra, "The Humble Programmer," in *ACM Turing Award Lectures: The First Twenty Years, 1966-1985* (New York, 1987), 17-32.

incentive and genuine advancement to be had in other areas upon completion of the dp [data processing] requirement."[50] A hierarchy developed within the software professions, as the more broadly educated "systems analysts" attempted to distinguish themselves from the narrowly technical "coders" and keypunch operators. The programmers sat somewhere in between these two extremes. Systems analysis was portrayed as a more abstract and transferable form of problem solving than mere programming, and therefore suggested wider applicability.[51] "To rise to the ranks of the systems analysts, the elite of the profession, a man not only has to master the technique of translating detailed instructions into a machine code, he must also be able to grasp concepts and to define the over-all, organized, systemic approach to the solution of a problem, or series of problem. And if he's to work with scientific or technical problems, he has to have the background to cope with the subject matter. … Men with such qualifications aren't easy to come by."[52] Systems analysts were more likely than programmers to rise to the level of upper management.[53]

Many of the job advertisements in the late 1960s and early 1970s reflected the concerns that programmers had regarding their occupational future and longevity. "At Xerox, we look at programmers … and see managers."[54] "Working your way towards obsolescence? At MITRE professional growth is limited only by your ability."[55] "Is your programming career in a closed loop? Create a loop exit for yourself at [the Bendix Corporation]."[56] Like their counterparts in the 1990s, programmers in this period were worried about burning out by age forty. Corporations struggled to retain the employees that they had invested so much time and money in recruiting and training. The average annual turnover rate in the industry approached 25%, and at one edp installation turnover reached more than 10% *per month*. Poor management, long hours, and easy mobility "too often made an already mobile workforce absolutely liquid."[57] One problem was a labor market that provided plentiful opportunities for experienced developers: "Once a man is taught the skills, he may be hard to keep. Companies that use their computers for unromantic commercial purposes risk losing their programmers to more glamorous fields such as space exploration."[58] Managers attributed excessive employer

[50] N. Rings, "Programmers and Longevity," *Datamation* 12/12 (1965).

[51] Scott Overton, "Programmer/Analyst: The Merger of Diverse Skills," *Personnel Journal* 52/7 (1972).

[52] Bylinsky (n. 18 above), 168.

[53] Frank Greenwood, "Education for Systems Analysis: Part One," *Systems & Procedures Journal* (January/February 1966).

[54] Xerox Corp., "At Xerox, we look at programmers and see managers (ad)," *Datamation* 14/4 (1968).

[55] Mitre Corp., "Are You Working Your Way Toward Obsolescence (ad)," *Datamation* 12/6 (1966).

[56] Bendix Computers, "Is Your Programming Career in a Closed Loop (ad)?" *Datamation* 8/9 (1962).

[57] "EDP's Wailing Wall," *Datamation* 13/7 (1967). While this high level of turnover was no doubt disruptive, it hardly compares to that experienced in certain traditional manufacturing industries. During the Ford Motor Company's 'labor crisis' of 1914, annual employee turnover reached 380%. Turnover in the contemporary software industry still averages 19% (based on the 11th Annual Salary Survey, *Computerworld*, 1 September 1997).

[58] Bylinsky (n. 18 above), 168.

turnover to the tight labor market, unscrupulous "body snatchers and other recruiting vultures,"[59] and the inherent fickleness of over-paid, prima donna programmers. Interestingly enough, however, a 1971 study of job satisfaction and computer specialists suggested that the majority of programmers valued the psychological benefits of their work—in terms of self-development, recognition, and responsibility—over its financial rewards.[60] What programmers disliked was the imposition of the "ultra-strict industrial engineering and accounting type controls"[61] aimed at limiting their professional autonomy.

Despite their concerns about the status and future of their profession, software developers in this period seemed to hold the position of power in the labor/management relationship. Programmers were able to vote with their feet on many crucial aspects of the terms and condition of their employment. Large government projects had difficulty attracting qualified programmers, in part because of salary considerations but mostly because they were seen as being boring and rigid. As one contemporary organizational sociologist suggested, programmers appeared to be "one group of specialists whose work seems ideally structured to provide job satisfaction."[62] What is curious, however, is that programmers on the whole do not seem to have translated their monopoly of the software labor market into stable long-term career prospects. They were unable to establish many of the institutional structures and supports traditionally associated with the professions. Although starting salaries were high and individual programmers were able to move with relative ease horizontally throughout the industry, there were precious few opportunities for vertical advancement.[63] Many programmers worried about becoming obsolete, and felt pressure to constantly upgrade their technical skills.[64] Most significantly, however, they faced the open hostility of managers. It was no secret that many corporate managers in this period were only too eager to impose new technologies and development methodologies that promised to eliminate what they saw as a dangerous dependency on programmer labor.[65]

Programmers and Managers: the Routinization of Labor …

The labor crisis in software has always been about much more than a mere disparity between supply and demand. By the end of the 1960s software development costs dominated the budget of most computer installations, and labor costs dominated the production of software. Managers quickly turned their sights on the programmers. Only the proper management of software personnel could save the

[59] John Fike, "Vultures Indeed," *Datamation* 13/5 (1967).

[60] Enid Mumford, *Job Satisfaction: A study of computer specialists* (London, 1972), 93.

[61] Robert Head, "Controlling Programming Costs," *Datamation* 13/7 (1967).

[62] Mumford, 175.

[63] James Jenks, "Starting Salaries of Engineers are Deceptively High," *Datamation* 13/1 (1967).

[64] "Learning a Trade," *Datamation* 12/10 (1966).

[65] Avner Porat and James Vaughan, "Computer Personnel: The New Theocracy—or Industrial Carpetbaggers," *Personnel Journal* 48/6 (1968).

software projects from a descent into "unprogrammed and devastating chaos."[66] Computer programmers often served as the symbolic representation of all that was wrong with the industry. They soon developed a reputation, deserved or otherwise, for being careless, unprofessional, and difficult to manage. As one senior vice-president of a Fortune 50 company, speaking of edp personnel, expressed it, "They don't exercise enough initiative in identifying problems and designing solutions for them. ... They are impatient with my lack of knowledge of their tools, techniques, and methodology—their mystique; and sometimes their impatience settles into arrogance. ... These technologists just don't seem to understand what I need to make decisions."[67] Many of the technological, managerial, and economic woes of the software industry became wrapped up in the crisis of software management.

Even when the software crisis was not explicitly articulated as a problem of programmer management, the relationship was often implied in the recommended "silver bullet" solution. When a prominent adherent of object-oriented programming 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,"[68] 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. The solutions to the "software crisis" that most frequently recommended—among them the elimination of rule-of-thumb methods (i.e. the "black art" of programming), the scientific selection and training of programmers, the development of new forms of management, and the efficient division of labor—are not fundamentally different from the four principles of scientific management espoused by Frederick Taylor in an earlier era.[69]

In his 1977 book *Programmers and Managers*, the labor historian Philip Kraft described what he called the "routinization of programming."[70] Building on the work of Karl Marx and Harry Braverman, Kraft situated the history of programming in one of the grand conceptual structures of labor history: the ongoing struggle between labor and the forces of capital. In his *Labor and Monopoly Capital: The Degradation of Work in the Twentieth Century*, Braverman argued that the basic social function of engineers and managers was to oversee the fragmentation, routinization, and mechanization of labor. Cloaked in the language of progress and efficiency, the process of routinization was envisioned primarily as a means of disciplining and controlling a recalcitrant work force. The ultimate result was the

[66] Robert Boguslaw and Warren Pelton, "Steps: A Management Game for Programming Supervisors," *Datamation* 5/6 (1959).

[67] Editorial, "The Thoughtless Information Technologist," *Datamation* 12/8 (1966).

[68] Brad Cox, "There is a Silver Bullet," *Byte Magazine* 15/10 (1990).

[69] Taylor's four principles of scientific management can easily be mapped on the software management literature of this and other periods. In brief, his four principles were: 1) develop a science for each element of work to replace traditional rule-of-thumb methods; 2) scientifically select, train, and develop the workers, rather than let them define their own work practices; 3) cooperate with the workers to insure adherence to the new scientific principles; 4) establish an equal division of the work and the responsibility between management and labor, with management taking over all the tasks for which they are better suited.

[70] Philip Kraft, *Programmers and Managers: The Routinization of Computer Programming in the United States* (New York, 1977).

deskilling and degradation of the worker. David Noble described the institutional foundations of the deskilling process in *America By Design* (1977) and its specific application in the numerically controlled machine tool industry in *Forces of Production* (1984).[71] His fellow Braverman disciple Philip Kraft applied the argument to computer programmers and the software industry:

> 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. … Elaborate efforts are being made to develop ways of gradually eliminating programmers, or at least reduce their average skill levels, required training, experience, and so on. … Most of the people that we call programmers, in short, have been relegated largely to subsidiary and subordinate roles in the production process. … While a few of them sit at the side of managers, counseling and providing expert's advice, most simply carry out what someone else has assigned them.[72]

Kraft suggests that managers have generally been successful in imposing structures on programmers that have eliminated their creativity and autonomy. His analysis is remarkably comprehensive, covering such issues as training and education, structured programming techniques ("the software manager's answer to the conveyor belt"), the social organization of the workplace (aimed at reinforcing the fragmentation between "head" planning and "hand" labor), and careers, pay, and professionalism (encouraged by managers as a means of discouraging unions). Although Kraft's conclusions may be controversial, his research addresses an essential aspect of the history of software as labor: attempts by corporate managers to address the software crisis by developing new methodologies of project management and process control.

There is no lack of evidence of pervasive management dissatisfaction with both programmers and the programming process. We have already described the enormous expenses incurred in the training, recruitment, and retention of software specialists. And since labor costs comprised almost the entire cost of any software development project, any increases in programmer efficiency or reductions in personnel immediately impacted the bottom line. In addition, software specialists had acquired a negative reputation in the eyes of corporate managers as being intractable and individualistic. According to one unflattering depiction, a programmer "doesn't want to be questioned, doesn't want to account accurately and in detail for his time. … He doesn't want to be supervised ... doesn't want to supervise. Says he wants responsibilities, but gripes if they're assigned to him. … The computer was acquired for him, not for operating results. … It's "not a pretty profile ..."[73] A widely quoted psychological study that identified as a "striking characteristic of programmers … their disinterest in people,"[74] reinforced the managers' contention that programmers were insufficiently concerned with the larger interests of the company. The apparent unwillingness of programmers to abandon

[71] David Noble, *America by Design: Science, Technology, and the Rise of Corporate Capitalism* (New York, 1977); David Noble, *Forces of Production: A Social History of Industrial Automation* (New York, 1984).

[72] Kraft, 26-8.

[73] Editorial, "Checklist for Oblivion," *Datamation* 10/9 (1964).

[74] Dallis Perry and William Cannon, "Vocational Interests of Computer Programmers," *Journal of Applied Psychology* 51/1 (1967).

the "black art of programming" for the "science" of software engineering was interpreted as a deliberate affront to managerial authority: "The technologists more 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."[75] The reinterpretation of the software crisis as a product of poor programming technique and insufficient managerial controls suggested that the software industry, like the more traditional manufacturing industries of the early twentieth century, was drastically in need of a managerial overhaul.[76]

The 1968 NATO Conference on Software Engineering is perhaps the earliest and best-known attempt to rationalize the production of software development along the lines of traditional industrial manufacturing. Comparing software writers unfavorably to hardware developers ("they are the industrialists and we are the crofters"), one speaker criticized the software industry for appearing "in the scale of industrialization somewhere below the more backward construction agencies."[77] Other conference participants echoed this call for the adoption of "mass-production techniques" of software production. The NATO conference stimulated further interest in the software engineering approach to system development, and was succeeded by a lengthy series of conferences, proposals, methodologies, and technological innovations aimed at eliminating corporate dependence on the craft knowledge of individual programmers. It would not be inaccurate to characterize much of the history of software as an ongoing and determined effort to develop what Frederick Brooks referred to as a "silver bullet" capable of slaying the werewolf monster of "missed schedules, blown budgets, and flawed products."[78] These efforts have typically belonged to one of three general categories: **procedural structures** aimed at disciplining both the labor force and the process of software development; **professional structures** intended to assure standard levels of programmer ability and product; and **technological structures** meant to reduce the number and required skill level of software personnel.

Procedural Structures for Managing Programmers

In the late 1950s, computer programming was often considered to be a uniquely creative activity—genuine "'brain business,' often an agonizingly difficult intellectual effort"[79]—and therefore almost impossible to manage using conventional methods. The limitations of early computers often demanded 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 ma-

[75] Editorial, "The Thoughtless Information Technologist," *Datamation* 12/8 (1966).

[76] H. V. Reid, "Problems in Managing the Data Processing Department," *Journal of Systems Management* (May 1970).

[77] M. D. McIlroy, quoted in Peter Naur, Brian Randall, and J. N. Buxton, *Software Engineering: Proceedings of the NATO Conferences*(New York, 1976), 5.

[78] Frederick P. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering," *IEEE Computer*, April 1987.

[79] Bylinsky (n. 18 above), 141.

chine'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. As John Backus would later describe the situation, "programming in the 1950s was a black art, a private arcane matter … each problem required a unique beginning at square one, and the success of a program depended primarily on the programmer's private techniques and inventions."[80]

By the middle of the 1960s, a perceptible shift in the relative costs of hardware and software had occurred. 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. The pressing problems for software developers were now more managerial than technical. 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 his review of Charles Lecht's *The Management of Computer Programmers*[81] Numerous potential solutions to the problem of programming management were suggested over the next several decades. In a presentation to the Fall Eastern Joint Computer Conference in 1965, J. Presper Eckert argued that programming would become manageable only when it could be referred to as "software engineering."[82] A few years later structured programming was advocated as the ideal tool for reducing the "vagaries of individual personality and 'style'."[83] In 1973 Terry Baker and Harlan Mills outlined their "chief programmer team" system, which they claimed would redefine software development as a "true professional discipline with a recognized, standard methodology."[84] Others recommended the virtues of rapid-prototyping and the iterative-spiral system of project management. In the late 1980s object-oriented programming (OOP) took over as the methodology *du jour*. "There is a silver bullet," claimed OOP advocate Brad Cox, suggesting that the adoption of OOP methods would finally bring about the long-awaited "software industrial revolution … that will alter the software universe as surely as the industrial revolution changed manufacturing."[85] The point is that although the particular man-

[80] Nick Metropolis, J. Howlett, and Gian-Carlo Rota, eds., *A history of computing in the twentieth century a collection of essays*(New York, 1980), 126.

[81] Robert Gordon, "Review of Charles Lecht, The Management of Computer Programmers," *Datamation* 14/4 (1968).

[82] Eckert quote reprinted in Gordon.

[83] Libellator, "Programming Personalities in Europe," *Datamation* 12/9 (1966): 28.

[84] Terry Baker and Harlan Mills, "Chief Programmer Teams," *Datamation* 19/12 (1973): 58.

[85] Brad Cox, "There is a Silver Bullet," *Byte Magazine* 15/10 (1990).

agement methodologies changed over time, the underlying message remained the same: “This time it will be different. This time it will work. This time we will be able to successfully impose the methods of ‘scientific’ management on an unruly and intractable workforce.”

Professional Structures for Managing Programmers

In addition to these procedural solutions to the problem of programmer management, corporate employers also encouraged software specialists to pursue their own professional development. “Professionalism instead of expertise can wipe out idealistic schedules and platitudinous projections and allow the data processing system group to do a realistic, efficient job.”[86] If the programmers could regulate themselves and certify standard levels of education and competence, then companies would need to spend less money on training and oversight.[87] They would also have a more reliable basis for making hiring decisions and evaluating productivity and performance. Codes of professional ethics were suggested as a means of encouraging high standards of performance and behavior.[88] Although the literature is replete with calls for the establishment of such codes, historians know little about how the various professions associations responded or to what effect. In any case, it appears that many companies honestly believed that enabling programmers to think of themselves as professionals “would be highly beneficial in the eventual progression of the industry toward well-ordered maturity.”[89] Several scholars have studied role of the corporation in the development of the engineering professions; there is a great need for similar work in the history of the software.[90]

Technological Structures for Managing Programmers

Perhaps the most clearly aggressive attempts to eliminate corporate dependence on expensive and unreliable labor involved the adoption of new “automatic programming” devices. These are the technologies that Philip Kraft accuses managers of using to “break down, simplify, routinize, and standardize … work so that it, too, can be done by machines rather than people.”[91] We are using the term “automatic programming” to refer not to any one specific technology but rather the managerial ideal of ordered, assembly line software development. A number of computer manufacturers did produce “automatic programming” systems intended to reduce that need for experienced programmers. The G-WIZ compiler from General Electric, for example, claimed that it would eliminate the need for pro-

[86] Jay Wesoff, “The Systems People Blues,” *Datamation* 14/6 (1968).

[87] Editorial, “Professionalism Termed Key to Computer Personnel Situation,” *Personnel Journal* 51/2 (February 1971).

[88] RAND Symposium, “Defining the Problem, Part II,” *Datamation* 11/9 (1965).

[89] Editorial, “Editor’s Readout: The Certified Public Programmer,” *Datamation* 8/3 (1962).

[90] For example, see David Noble, *Forces of Production* (n. 71 above); Edwin Layton, *The Revolt of the Engineers: Social Responsibility and the American Engineering Profession* (Baltimore, 1971); Robert Zussman, *Mechanics of the Middle Class: Work and Politics Among American Engineers* (Los Angeles, 1985).

[91] Kraft (n. 70 above), 26.

grammers by allowing managers to do their own programming.[92] Similar claims were made for FORTRAN and COBOL. More recently the Department of Defense-sponsored ADA programming language has been trumpeted as "a means of replacing the idiosyncratic 'artistic' ethos that has long governed software writing with a more efficient, cost-effective engineering mind-set."[93] The effectiveness of these systems, both past and present, was over-sold in the marketing literature.[94] What is important is the obvious appeal that these systems and languages held for corporate employers. In its "Meet Susie Meyers" 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. If pretty little Susie Meyers, with her spunky miniskirt and utter lack of programming experience, could develop software effectively in PL/1, so could just about anyone.

How should we then understand the claims of Kraft and others that the history of programming in the recent decades has been one of continual discipline, deskilling, and degradation? An uncritical reading of the management literature on software development, with its confident claims about the value and efficacy of various performance metrics, development methodologies, and programming languages, would suggest that Kraft and his associates were correct. In fact, many of these methodologies 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."[95] Their authors would be the first to admit it. By taking these claims at face value, Kraft is able to provide a comprehensive interpretation of a wide variety of developments and phenomena: the fragmentation of the workforce, the appeal of structured programming, rising levels of job turnover and employee dissatisfaction, the increased use of foreign laborers. Joan Greenbaum, a contemporary of Kraft and intellectual "fellow traveler," has recently reaffirmed her belief in the Braverman deskilling hypothesis: "If we strip away the spin words used today like 'knowledge' worker, 'flexible' work, and 'high tech' work, and if we insert the word 'information system' for 'machinery,' we are still talking about management attempts to control and coordinate labor processes."[96]

A more critical reading suggests that the claims of the management literature represent imagined ideals more than current reality. Writing in 1971, the occupational sociologist Enid Mumford actually lauded data processing as an "area where the philosophy of job reducers and job simplifiers—the followers of Tay-

[92] The G-WIZ compiler is described in the RAND Symposium, "On Programming Languages: Part II," *Datamation* 8/11 (1962).

[93] David Morrison, "Software Crisis," *Defense* 21/2 (1989).

[94] RAND Symposium, "On Programming Languages: Part II," *Datamation* 8,11 (1962).

[95] Kraft (n. 70 above), 26.

[96] Joan Greenbaum, "On twenty-five years with Braverman's 'Labor and Monopoly Capital.' (Or, how did control and coordination of labor get into the software so quickly?)," *Monthly Review* 50/8 (1999).

lor—has not been accepted."[97] The fact that the software crisis has survived a half-century of supposed 'silver bullet' solutions suggests that Kraft may have overlooked a crucial component of this history. What is missing from his analysis is the perspective on the software labor process provided by the many companies who recognized that computer programming was, at least to a certain extent, a creative and intellectual demanding occupation, and who, in their management of software personnel stressed "the importance of a judicious balance between control and individual freedom."[98] In the words of an astute contemporary observer:

> 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. … 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.[99]

Although Kraft accurately describes the features of a specific managerial response to the software crisis, he misses its larger historical significance. Attempts to 'resolve' the crisis, by either pronouncing it over or suggesting particular solutions, are typically either a historical or just plain uninteresting. It is the persistence of the crisis that makes it so fascinating to the historian. In what ways have popular perceptions of the software crisis been politically influenced and socially constructed? What does the perpetual crisis of programming labor tell us about the unique characteristics of the software industry and the complex and controversial relationship between the "art of programming" and the "science" of software engineering? How can we explain the failure of traditional labor market mechanisms to alleviate the ongoing shortage of programmers? How can we relate the history of software to larger themes in social and labor history? We have only hinted at some possibilities—the opportunities for further significant research are enormous.

Women in Software

In recent years labor historians have devoted considerable attention to issues of race and gender in the history of labor-management relations and the dynamics of the workplace environment. The conventional wisdom argues that corporate managers often use women and minorities as low-wage, low-skill replacements for skilled white male laborers. Occupations tend to become sex-typed as being either male or female, depending on their relative position in the wage and status hierarchy. An influx of women and/or minorities into an occupation is usually considered to indicate that routinization, degradation, and deskilling has occurred. Women have rarely held high positions within the scientific or engineering community in significant numbers, at least until fairly recently.

There is evidence that the story of gender and software labor is a little less clear-cut. As a number of scholars have suggested, women have played an impor-

[97] Mumford (n. 60 above), 175.

[98] Robert Head, "Controlling Programming Costs," *Datamation* 13/7 (1967): 141.

[99] Willis Ware, "As I See It: A Guest Editorial," *Datamation* 11/5 (1965): 27.

tant role in the history of software development. The first ENIAC programmers were women, and Jennifer Light has argued that these women significantly influenced early computing and programming practice.[100] The Association for Computing Machinery's first "Man of the Year" was a woman.[101] Women have not only held a greater percentage of jobs in software than might otherwise have been expected, they were also able to advance farther and faster than there peers in other high-tech industries. Clearly there is something interesting going on in the history of the software professions that deserves further scholarly examination.

What do we know about women and software? Women were the very first programmers, or 'coders' as they were called in the earliest years of computing. The intended role of these women was clearly articulated in the three volumes on "Planning and Coding of Problems for an Electronic Computing Instrument," written by Herman Goldstine and John von Neumann in the years between 1947 and 1949.[102] These three volumes served as the principal textbooks on the programming process at least until the early 1950s. The Goldstine/von Neumann method assumed that the computer would be used for complex scientific computation, and the division of labor in the programming task seems to have been based on the practices used in programming the ENIAC.

Goldstine and von Neumann spelled out a six-step programming process: (1) conceptualize the problem mathematically and physically, (2) select a numerical algorithm, (3) do a numerical analysis to determine precision requirements and evaluate potential problems with approximation errors, (4) determine scale factors so that the mathematical expressions stay within the fixed range of the computer throughout the computation, (5) do the dynamic analysis to understand how the machine will execute jumps and substitutions during the course of a computation, and (6) do the static coding. The first five of these tasks were to be done bythe "planner" who was typically the scientific user and overwhelmingly often was male; the sixth task was to be carried out by "coders"—almost always female (on the ENIAC project). Coding was regarded as a "static" process by Goldstine and von Neumann, one that involved writing out steps of a computation in a form that could be read by the machine, such as punching cards, or in the case of ENIAC in plugging cables and setting switches. Thus there was a division of labor envisioned that gave the most skilled work to the high-status male scientists and the lowest skilled work to the low-status female coders.

It turns out that the coders on the ENIAC project ended up doing many more tasks than envisioned. 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 dynamic analysis; and with their growing skills, some scientific users left many or all six of the programming stages to the coders. In order to debug their programs and to distinguish hardware glitches from software errors, they developed an intimate

[100] Jennifer Light, "When Computers Were Women," *Technology & Culture* 40/3 (1999).

[101] Admiral Grace Hopper received her "Man of the Year" award in 1962. Needless to say, it was extremely unusual for an association of technical professionals to grant its highest honor to a woman, especially in the early 1960s!

[102] These technical reports are most easily found today in reprint form in William Aspray and Arthur Burks, eds., *The Papers of John von Neumann on Computing and Computer Theory* (Cambridge, Mass. and Los Angeles, 1987).

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."[103] Thus what was supposed to have been a low-skill, "static" activity prepared these women coders well for careers as programmers—and indeed, those who did pursue professional careers in computing often became programmers and did well at it. A few women, Grace Hopper and Betty Holberton of UNIVAC and Ida Rhodes and Gertrude Blanche of the National Bureau of Standards in particular, continued to serve as leaders in the programming profession.[104]

However, during the 1950s, business applications began to surpass scientific applications; a computer manufacturing industry grew up to service the rapidly expanding need for computers for business applications; and a tremendous demand grew up for programmers. The number of new programmers, most of whom were male at first, swamped the number of female coders who had become programmers. Programming quickly became primarily a man's job.

If the Braverman/Kraft thesis about the deskilling of programming labor were correct, we would expect to see the employment of women in software increase as the occupation became less skilled and more routine. In a 1964 survey, 76 percent of the respondents expected to see the ratio of women in programming increase: "The only limitation is the number of qualified applicants," stated one manufacturer.[105] There are indications that certain types of female employees were seen, at least in the 1960s, as being more stable and reliable than their male counterparts, based upon some typical sexual stereotyping: "Women are less aggressive and more content in one position ... Women … are more prone to stay on the job if they are content, regardless of a lack of advancement. They also … are less willing to travel or change job locations, particularly if they are married or engaged. For these reasons there is a considerably lower turnover rate in women programmers and as a result, the initial investment in training pays a greater dividend for their employees."[106] Employers were warned away, however, from hiring "the most undesirable category of programmer," the female "about 21 years old and unmarried," who was likely to marry, become pregnant, or waste precious energy worrying about her social commitments for the weekend.[107]

There is no doubt that some male programmers were threatened by a perceived incursion of females into their profession. For many of these men, women were associated with low-skill clerical labor, even though many of the ENIAC 'girls' had actually possessed college degrees in mathematics. The new generation of female programmers was being recruited from the ranks of keypunch operators or

[103] W. Barkley Fritz, "The Women of Eniac," *Annals of the History of Computing* 18/3 (1996): 20.

[104] Frances Elizabeth ("Betty") Snyder Holberton was awarded the Association for Women in Computing's Ada Lovelace Award in 1997. Grace Hopper described her as being "the best programmer that she had known during her long career." (Fritz, 20).

[105] Report, "Advanced Programmers, Women Employment Seen Rising," *Datamation* 10/2 (1964).

[106] Valerie Rockmael, "The Woman Programmer," *Datamation* 9/1 (1963): 41.

[107] William Paschell, Automation and employment opportunities for office workers; a report on the effect of electronic computers on employment of clerical workers (Washington, D.C., 1958); also Rockmael, 41.

‘coders.’ In an era when programmers were anxious to distinguish programming as a creative intellectual activity from coding as manual and narrowly technical labor, these women represented the lowest rungs of the occupational hierarchy (“There’s nothing lower than a coder”[108]). An influx of low-skill, low-wage labor threatened both the professional self-identify of the programmers and their superior bargaining position in the labor market for software workers. It is hard to imagine, therefore, that they would have been pleased or flattered by Helen Gurley Brown’s exhortation to the readers of *Cosmopolitan* that they go out and get jobs as programmers making $15,000 after five years.[109] Many of the advertisements for “automatic programming” languages and systems used women as a proxy for less expensive, more tractable labor. If you could teach your secretary to program in COBOL, there was no need to pay for expensive programming talent.

There are other historical questions to be asked about gender and software labor. Recent statistics on computer science enrollments and software industry employment indicate that the number of women in computing has been dropping since the early 1980s. Why? It has been argued that many women perceive computer careers as being overly competitive, incompatible with a well-rounded family oriented lifestyle, and solitary rather than social.[110] Writers such as Sherry Turkle and Tracy Kidder have described the various ways in which the programmer subculture emphasizes culturally masculine traits such as competitiveness, practical joke playing, and aggressive hacking and cracking.[111] How and why did this masculine subculture develop? How does it relate to the perpetual software labor crisis? Anecdotal evidence suggests that women are attracted to programs in information systems, rather than computer science or computer engineering, because “information systems is perceived as more people-oriented and more attuned to the uses of information technology.”[112] What does this tell us about the historical and social construction of computer knowledge and specialties? In what ways has the absence of women from the programming profession been used to emphasize its rational, “scientific” qualities? Labor historians have developed an extensive literature on work and gender; historians of software should make use of their expertise and experience.

Other Major Players …

The bulk of this paper has focused on specific issues in the history of software as a labor process. It seems appropriate at this point to step back and briefly situate these issues in the larger context of post-war social and technological developments. Let us begin with a discussion of other major players in late-twentieth

[108] "Checklist for Oblivion,” *Datamation* 10/9 (1964).

[109] The quote from Helen Gurley Brown appears in an advertisement for the Computer Sciences Corporation, “In case you missed our first test,” *Datamation* 13/9 (1967).

[110] Peter Freeman and William Aspray, *The Supply of Information Technology Workers in the United States* (Washington, D.C., 1999), 113.

[111] Sherry Turkle, *The Second Self: Computers and the Human Spirit* (New York, 1984); Tracy Kidder, *The Soul of a New Machine* (New York, 1984).

[112] Freeman and Aspray, 111.

century labor and technology: labor unions, the defense community, and other government agencies.

Labor Unions

Formal labor organizations have played almost no role in the history of the software industry. In one respect this is not entirely unexpected, since whitecollar professionals have traditionally resisted unionization. Employers tend to encourage their software workers to think of themselves as professionals, at least in regard to this particular issue. There is recent evidence to suggest that this situation may be changing, however. The Washington Alliance of Technology recently won important concessions from Microsoft over its treatment of so-called "permatemp" employees.[113] Although these high-tech consultants are often paid relatively high hourly wages, they typically do not receive health-care benefits, vacation time, stock options, pension plans, or overtime. Whereas so-called "free agents" like these make up only ten percent of the overall workforce in the United States, they comprise almost half of all software employees.[114] Like many high-tech workers in the late twentieth-century, software specialists straddle the border between the professional and the technician. In the past, programmers resisted association with hourly workers and other wage laborers. They prided themselves on being salaried professionals on par with other engineers and managers. It may be that changes in the labor market, the rise of overseas competition, and an influx of foreign laborers may foreshadow an increased presence of organized labor in the software industry.

Defense Community

There are few technology industries in the late twentieth century that have been unaffected by Cold War politics and the imperatives of the military-industrial complex. The software industry is presumably no exception to this general rule. It is unfortunate, therefore, that historians know so little about the influence of the Cold War and the military on the production of software. James Tomayko, who has written widely about both history of computing in aerospace and the historical development of software engineering, has argued that the NASA software development efforts, like the SAGE System and the IBM OS/360 operating system, were "major software projects that directly contributed to the evolution of software engineering."[115] Philip Kraft argues that it was the Korean War that "provided the incentive to organize the training of programmers in the same manner as other engineering occupations," and, not surprisingly, "the military which provided both the means and the setting to do so."[116] He suggests that a Cold War mentality entered the programming profession through the RAND Corporation

[113] "Microsoft Moves to End Permatemping," *The Washington Alliance of Technology Workers*, September 2, 1999 (http://www.washtech.org/roundup/contract/ms_conversion.html).

[114] Austin Bunn, "No-Collar Workers: Is there room for unions in the New Media world?" *The Village Voice*, January 13, 1999.

[115] James Tomayko, *Computers in Spaceflight: The NASA Experience* (Linthicum Heights, MD, 1998).

[116] Kraft (n. 70 above), 37.

and its association with the SAGE project. Paul Edwards' more recent argument that the highly centralized SAGE system "provided the technical underpinnings for an emerging dominance of military managers over a traditional experience- and responsibility-based authority system" perhaps applies equally well to the software as well as the military professions.[117] There is some evidence to support this opinion. In the late 1950s the SAGE project did indeed serve as "the training ground for an industry." Many SAGE veterans went on to hold prominent positions in the software community. The sheer size and complexity of the SAGE project, along with its particularly sensitive nature, did encourage a modular, hierarchical approach to software production. It may be that this did have a strong influence on later developments. The truth is that we just do not know. There is a strong need for further research in this area.

Other Government Agencies

Generally, the federal government has not established direct labor policy for information technology workers, including software workers. Instead, this policy in the postwar period has been embedded in policies for science, education, public welfare, economics, and business. Through the 1970s, IT labor policy was mainly the result of legislation related to science and education policy concerning the National Science Foundation. Some unknown number of programmers were trained in the formal higher educational system under the provisions of the National Defense Education Act, which was stimulated by the Russian launch of Sputnik. NSF provided an important computer facilities program from 1957 to 1973, which helped some 500 U.S. universities acquire their first computers. These computers were used to train a generation of computer professionals, including many programmers. DARPA opened its computer science program in 1962, and NSF opened an Office for Computing Activities in 1967. Although funding from these programs often went to support research projects, these projects were the training ground and means of financial support for many graduate students, some of whom became software professionals. White papers written by the National Academy of Science and the NSF led to substantially increased support for campus computing programs for both research and education.[118]

Federal budget trimming to pay for the Vietnam War and the Mansfield Amendment to the 1972 Military Procurement Authorization, which narrowed the scope of research that the military could support, significantly harmed academic computer science. Support for computer facilities was suspended, research and education funds for computer science dwindled, and universities turned increasingly to theoretical research projects rather than large-team, empirical studies. The

[117] Paul Edwards, *The Closed World: Computers and the Politics of Discourse in Cold War America* (Cambridge, 1996), 104.

[118] For further information on the role of the federal government on United States information technology policy see William Aspray, "The Early History of IT Worker Policy," *Computing Research News*, September 1999; "The Recent History of IT Worker Policy," *Computing Research News*, November 1999; "The Supply of Information Technology Workers, Higher Education, and Computing Research: A History of Policy and Practice in the United States," in *The International History of Information Technology Policy*, ed. Richard Coopey (Oxford, forthcoming).

universities were becoming progressively less interesting places for computing research, and faculty members and graduate students took flight to industry. Undergraduate enrollments in computer science were burgeoning at the same time, and there was widespread concern that industry pull was eating the seed corn of potential faculty available to train the next generation of students. Moreover, the people coming out of university programs were not all that attractive to industry. A response that involved NSF, the universities, the information technology (IT) industry, and the professional societies slowly overcame these problems in the 1980s.

Computing had first hit the federal radar screen in the 1960s. During the 1980s it became of serious policy interest for the first time. NSF and the National Research Council formed major organizations to deal with computer science. Computing was seen as having an important role in national economic competitiveness. This was one of the reasons behind 1991 legislation that established the High Performance Computing and Communications Initiative, funded in the billion dollars range. Academic research and educational programs received strong financial support under this legislation.

The Immigration Act of 1990 shifted the balanced of immigration somewhat away from family-based immigration and more towards career-based education. This enabled the number of IT workers on permanent visas to increase, but the numbers remained small. Even with the legislative change, fewer than 2,000 permanent visas were awarded per year to mathematicians and computer scientists. Under pressure from industry who needed more IT workers, the H-1B temporary visa program was implemented; and new legislation was passed in 1998 to greatly increase the number of these visas awarded annually. The issue of foreign workers and temporary visas remains a hot political topic today. Other recent issues that have been subject to federal policy are another round of seed-corn problems in university computer science programs, the under-representation of women and minorities in the computing field, and industry demand for tax credits to companies to provide training for their workers. The increasingly important role of the computer and the Internet in the economy and everyday life has been noticed in Washington, and interventionist policies directed at computing technologies are now more common and likely to increase in the future. Unlike some of the general education and science legislation of past decades, which had only indirect bearing on software labor, issues concerning the number and training of programmers are of direct policy interest today.

Conclusions

In the previous sections of this paper, we have identified issues in the labor history of software by briefing telling aspects of the history. In this section, westand back from the history and identify several key areas deserving further historical attention. So we simply close by asking a list of questions. Work in these areas will support the scholarship of historians of computing aiming to get a more complete picture of their subject, as well as labor historians who want to draw examples from this important technical area. These are questions for studying the U.S. sofware labor situation; perhaps the international situation will require a different set of questions.

Training, Education, and Identification of High-Quality Workers

The general understanding of what a computer is, and what it is for, has changed significantly over time. As the computer transitioned from a scientific instrument to an information processor to a communications device, how has the software industry met the ever-increasing demand for programmers (or perhaps more specifically, a certain *type* of programmer)? What qualifications and character traits did managers seek in their software laborers? How have the skills required to do programming work effectively, as well as the aspirations and background of the workers, evolved over time? Who was attracted to these jobs? How were they educated, recruited, and trained? What made for the big differences in the productivity levels across individual programmers?

Professionalization, Certification, Career Development, and Occupational Identity

Labor historians have shown that many workers are concerned not only with the material conditions of their work (such as safety, pay, hours, etc.), but also with less tangible issues of status, personal development, and identity. Programmers have long had an interest being perceived as professionals, rather than technicians. How have the role and occupational identity of software personnel changed over time? To what extent has professionalism been encouraged through the creation of barriers to entry such as certification, accreditation, and standardized curricula? How have programmers worked to establish an occupational or professional identity through the construction of programming as an engineering or scientific discipline, or through the elevation of the status and visibility of programming within the corporation? Have programmers managed to successfully establish themselves as professionals? In what ways is this profession like and unlike those that have traditionally been studied by labor historians?

Structures Imposed by Management on Labor, and by Labor on Management

The "Taylorization" of work in the twentieth century is a (and perhaps *the*) major theme in contemporary labor history. To what degree has there been an attempt to apply to programming the scientific management techniques that seemed to have worked so well in the traditional manufacturing industries? To what degree has management been able to define programming skill and practice and therefore assure themselves of a standardized worker and product? How has management attempted to fragment, routinize, and deskill software work? Have these attempts generally been successful? Given that labor seems to have the upper hand over management today and for much of the past because of the laws of supply and demand, in what ways and to what extent has labor been able to shape the work environment?

Gender, Race, and the Culture of the Workplace

Why has the participation of women and minorities been so low in this field, especially in the last two decades during which other science and engineering groups have experienced improvements in the participation of such underrepresented groups? To what degree has there been deskilling and gender typing in the sofware field, and how has this varied over time? What has been the effect of under representation of these groups on those seeking to produce software products?

Government Regulation and Government Programs that Affect Software Labor

In what ways, if at all, have government programs in the United States to provide support for research and education in the universities shaped the development of software workers? In what ways has defense needs for computing technology shaped either the demand for software labor or the way that it is organized and managed? How has immigration law affected the supply for software labor and the ways in which software workers are employed and relate to their employers? Have tax training and other corporate incentives from government changed the nature of the people who have been hired to do software work or the career path of software workers?

As we begin to answer some of these questions, we will enrich our understanding both of software history and of the history of labor.