Making Programming Masculine

Nathan Ensmenger
University of Pennsylvania
July 18, 2008

Draft copy. Please do not circulate or cite.

I The Computer Girls

In the April 1967 issue of Cosmopolitan magazine, sandwiched between such conventional women’s magazine fare as “The Bachelor Girls of Japan” and “A Dog Speaks: Why a Girl Should Own a Pooch,” appeared a curious little essay entitled simply “The Computer Girls.” The “computer girls,” as the article explained, were the female “computer programmers” who taught the dazzling new “miracle machines” called computers “what to do and how to do it.” There were already more than 20,000 women working as computer programmers in the United States, argued the article’s author, Lois Mandel, and there was an immediate demand for 20,000 more. Not only could a talented “computer girl” command as much $20,000 a year, Mandel confidently declared, but the opportunities for women in computing were effectively “unlimited.” The rapid expansion of the computer industry meant that “sex discrimination in hiring” was unheard of, and anyone with aptitude – male or female, college-educated or not – could rise to the top of the field. And not only were women in computing treated as equals, but they actually had many advantages over their male colleagues. Programming was “just like planning a dinner,” Mandel quoted the noted computer scientist Dr. Grace Hopper as saying. “You have to plan ahead and schedule everything so it’s ready when you need it. Programming requires patience and the ability to handle detail. Women are ‘naturals’ at computer programming.”

It would be easy to dismiss “The Computer Girls” as a fluff piece, a half-hearted attempt by Cosmopolitan to capitalize on contemporary interest in the computer revolution. And indeed, there are elements of the article that would support this conclusion.

---

To modern readers, of course, the very language of the “computer girl” appears condescending and sexist. The analogy that is made between computer programming and recipe creation seems forced and superficial, even coming as it does from the well-known and highly respected computing pioneer Grace Hopper. At times the article descends into what seems almost a parody of formulaic Cosmopolitanism, such as when Sally Brown, “a redhead from South Bend, Indiana” confesses that she doesn’t mind working late because there is often “a nice male programmer to take a girl home …” About another female programmer – a Ms. Diane Johnson – it is noted that she “could easily have been a high-fashion model.” At one point the author speculates, seemingly without irony, about the “the chances of meeting men in computer work.” (The conclusion she comes to is that these are “very good,” as the field was currently “overrun” with men.) The last word of the article text comes from a male programmer, rather than one of the “computer girls,” and is patronizing and dismissive: of course “we like having the girls around,” he declares, “they’re prettier than the rest of us.” And finally, in true Cosmopolitan style, the article concludes with a characteristic “Cosmo Quiz”: by answering a few simple questions, any Cosmo girl could allegedly determine whether she too had what it took to be a professional computer programmer making “$15,000 after five years.”

But underneath its seemingly frivolous exterior, “The Computer Girls” represents an insightful overview of the gender dynamics of computer work at one of the most critical periods in its history. It reflects very accurately the confusing – and often contradictory – messages about the proper role of women in the computing fields that were floating around in the formative decades of electronic computing. On the one hand, women did play a critical role in early computing, particularly in computer programming. Compared to most technical professions, computer programming did remain unusually open to females. But on the other hand, in the late 1960s the computer programming community was also actively making itself masculine, pursuing a strategy of professional development that would eventually make it one of the most stereotypically male professions, inhospitable to all but the most adventurous and unconventional women.

Let’s begin with the what the Cosmopolitan article gets right:

First of all, it is entirely accurate in its claim that there were an exceptional large number of women working in computer programming. In fact, if anything it underestimates the percentage of women working as programmers in this period. Mandel suggests that one out of every nine working programmers was female. This is probably overly conservative. The exact percentage of female programmers is difficult to pin

---

1Ibid.
down with any accuracy – even figuring out the total number of programmers in this period is difficult – but other reliable contemporary observers suggest that it was closer to 30, or even 50, percent.\textsuperscript{3} The first government statistics on the programming profession do not appear until 1970, when it was calculated that 22.5 percent of all programmers were women – an estimate that, although itself probably conservative, was more than twice as high as Mandel's.\textsuperscript{4}

Of course, even with more complete census data it would be dangerous to make too many conclusions about the presence of women in computing. Computing itself is a very broad term covering a multitude of occupational categories, including high status disciplines like computer programming and systems analysis as well as low-status jobs such as keypunch operator. Women tended to congregate in the lower end of the occupational pool in computing. Even within computer programming there were different roles differentially available to men and women. But as the Cosmopolitan article rightly points out, compared to most of the traditional professions computer programming was remarkably receptive to females. It cited as evidence the experience of programmer Helene Carlson, who had previously earned an MA degree in astrophysics at Harvard. Carlson had early on discovered that “there wasn’t much a woman could do in astronomy.” In computing, however, Carlson felt that she had been “fully accepted as a professional.”\textsuperscript{5}

Again, there is evidence to suggest is that Carlson (and Cosmo) was absolutely spot-on in regard to the vertical mobility available to women in computer programming. Compared to other technical disciplines, computer programming was not highly stratified along gender lines. Not only were women able to break into the entry levels of the profession, but they were often able to claw their way to its highest pinnacles. In 1969, for example, the Data Processing Management Association recognized Grace Hopper with its very first “man of the year” award in the computer sciences. That an emerging professional society with grand aspirations for technical and managerial leadership would even consider giving its first major award to a woman is really quite remarkable. Although Hopper was unusual in that she possessed both a Ph.D. and a commission in the United States Navy (at that time as a Lieutenant Commander), she was not entirely sui generis: other women, including Betty Snyder Holberton, Jean Sammet, and Beatrice Helen Worsley, all came to occupy influential positions within the computing community.\textsuperscript{6}

\textsuperscript{5}Mandel, op. cit.
In addition to accurately representing the state of the contemporary labor market in programming, the Cosmopolitan article also does a reasonable job of explaining its unique characteristics. In large part, it argued, the unusual freedom of opportunity available to women in computing was simply an outgrowth of the rapid growth of the commercial computer industry. An industry that was doubling in size every year or two simply could not afford to discriminate against women. “Every company that makes or uses computers hires women to program them,” the article noted matter-of-factly, “If a girl is qualified, she's got the job.” And since the meaning of “qualified” in this period was still being negotiated (more on this point later), there was no particular reason for firms to privilege men over women.7

It would be difficult to over-emphasize the degree to which the programmer labor shortage of the late 1960s dominated contemporary discussions of the health and future of the computer industry. Since the early 1950s industry employers had been warning of an imminent shortage of computer programmers. By the early 1960s these warnings seemed to have been realized. The “gap in programming support” caused by the ever-worsening “population problem” threatened to wreak havoc with the industry.8 In 1962 the editors of the powerful industry journal 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.”9 In 1966 the “personnel crisis” had developed into a full blown “software crisis,” according to Business Week magazine.10 An informal 1967 survey of MIS (management information systems) managers identified as the primary hurdle “handicapping the progress of MIS” to be “the shortage of good, experienced people.”11 One widely quoted study released that same year noted that although there were already 100,000 programmers working in the United States, there was an immediate need for at least 50,000 more.12 Estimates of the number of programmers that would be required by 1970 ranged as high as 650,000.13 “Competition for programmers has driven salaries up so fast,” warned Fortune Magazine in 1967, “that programming has become probably the country’s highest paid technological occupation … Even so, some companies can’t find experienced programmers at any price.”14 The ongoing “shortage of capable programmers,” argued the industry journal Datamation, “had profound implications, not only for the computer industry

---

7 Mandel, “The Computer Girls”.
as it is now, but for how it can be in the future.”

In the face of this perpetual shortage of programmers, employers turned to extraordinary measures. Recruitment companies scoured local community centers and YMCAs for potential programmer trainees, administering programming aptitude tests to almost warm body they could find. In 1968 one computer service bureau in New York City even began testing inmates at the nearby Sing-Sing prison, promising them permanent positions pending their release. Given that employers were willing to hire prisoners as programmers, their appeal to Cosmopolitan readers is perhaps more comprehensible. As in the case of other severe labor shortages – wartime, for example – women were able to move into fields from which they might otherwise have been excluded.

But it was not only the desperate need for programmers that allowed women to unique opportunities within the profession. Although in the late 1960s programming was generally considered highly-skilled labor – as one observer declared, “generating software is ‘brain business,’ often an agonizingly difficult intellectual effort” – the exact nature of that intellectual effort was not yet clearly defined. Programming was “not yet a science,” argued the same observer, “but an art that lacks standards, definitions, agreement on theories and approaches.” The lack of a fully established scientific or engineering identity left space open for women. Although the possession of a college degree in mathematics was still considered a necessity in scientific computing (which tipped the scales demographically in favor of males), business computing – the most rapidly growing segment of the commercial computer programming industry – required an entirely different set of skills. What these skills were no-one quite knew, and so many firms relied on aptitude tests to determine which employees had the most potential for programming. Aptitude was everything; you either had it or you didn’t. And since there was no particular reason that these aptitude tests were gender-specific (again, more on this later), there was also no reason that men would be more likely than women to be selected as programmer trainees. In addition, as the Cosmo article also correctly noted, since most firms preferred to hire train programmers from within, and therefore often tested all of their employees for programming aptitude, this meant that even women working in such highly feminized (and low-status) occupations as stenography had a chance at becoming a programmer. The trick was getting some initial experience: as one employment counselor cited by Mandel argued, “a girl’s best

---

17“First Programmer Class at Sing Sing Graduates”. In: Datamation 14.6 (1968). Pp. 97–98.
18Bylinsky, loc. cit.
19Ibid., p. 141.
bet is to get a spot anywhere in the computer department, using skills like filing or typing or accounting, with the plan in mind to get on the firm’s programmer-trainee list from the inside.” There were outside vocational schools that claimed to prepare people for careers in programming, but as one of the “girls” quoted in the article declared, “I’d never consider paying for my own training when I can get someone else to pay for it.” The combination of low-barriers to entry and subsidized technical education made programming powerfully appealing to many women who might otherwise be trapped in traditionally female occupations.

It is worth noting as well that, given this context, the quiz provided at the end of “The Computer Girls” article was no superfluous or silly afterthought. The quiz included real questions from the aptitude test developed by NCR to test for programming aptitude. Similar tests, most notably the IBM Programmer Aptitude Test (PAT), were used by eighty percent of all employers to select for programmer trainees. In 1967 alone, the PAT was administered to more than 700,000 individuals.

In any case, after noting a few other reasons why programming might be an appealing profession for women – including that at least some programming work could be done at home (while children were napping) – the Cosmo article concluded by suggesting that it was largely a lack of knowledge about the field that kept women from entering it in greater numbers. Since programming was thought to be vaguely mathematical in nature (incorrectly, the article concludes), and since female students were often discouraged from pursuing any fields involving science or mathematics, they too often missed out on the exciting opportunities available in programming. This was unfortunate. “I don’t know of any other field, outside of teaching, where there’s as much opportunity for a woman,” the article quoted the director of education for the Association of Computing Machinery, James Adams, as saying. “Soon, mothers will be telling their daughters: ‘Now study your arithmetic so you can become a computer girl.” Although the overall positive tone of the article is undercut somewhat at the very end by the aforementioned snark from a male programmer about the desirability of recruiting “prettier” colleagues, “The Computer Girls” nevertheless represents a smart, optimistic, and insightful glimpse at the potentially bright future of women in computing.

beginning in the mid-to-late 1990s, historians of computing, both popular and

academic, began to recognize the crucial contributions that women have made to the development of the electronic computer. Like many such (re)discoveries of the previously unrecognized contributions of women to the development of science and technology, this one had both historical and contemporary significance. Given that computing was generally considered to be a particularly hyper-masculine environment (even when compared to the traditionally male-dominated engineering disciplines), the surprisingly large presence of women in early computing seemed to turn on its head conventional assumptions about the lack of female participation in contemporary computing. It wasn’t that women were uninterested in computing, or unprepared or constitutionally disinclined to participate, the historical evidence seemed to suggest, but rather that their participation had been systematically ignored or under-reported. In light of contemporary debates about low (and declining) female enrollments in departments of computer science, this seemed a significant and empowering discovery. 

The focus of most of this literature has been, understandably enough, on what Judy Wacjman, among others, has called the “hidden history” of women in technology. The goal was to explore what the history of women in computing had to say about women – about their contributions, experiences, and abilities. 

This essay will address instead the flip side of this question: namely, what has the history of women in computing had to say about computing. In the broader history of technology, a large-scale reliance on female labor has traditionally been seen as a either a reflection or harbinger of industrialization, of the routinization and degradation of labor, and of a loss of status and autonomy. Because of the modern association of computer work – particularly computer programming – with high-status males, we tend to assume that such work has always been masculine, and that the presence of women is therefore exceptional. My argument is that most computer work – again, particularly computer programming – began as women’s work. It had to be made masculine. This process of masculinization was closely associated with the development of the professional structures of the discipline: formal programs in computer science, professional journals and societies, certification programs and standardized development methodologies. Seen from the perspective of aspiring computer professionals (primarily male), “The Computer Girls” represented not a celebration of the openness and opportunity inherent in their industry, but an indictment of everything that was
II  In The Beginning Were The Women …

The most prominent case study in the history of women in early computing is that of the “Women of ENIAC” (or, as they were better known to contemporaries, the “ENIAC Girls”). These women – Kathleen McNulty, Frances Bilas, Betty Jean Jennings, Elizabeth Snyder, Ruth Lichterman, and Marlyn Wescoff – were female “human computers” recruited by the male ENIAC engineers/managers to “set-up” the general-purpose ENIAC machine to perform the specific “plans of computation” required to solve particular real-world problems. Although the idea of the computer “program” had not yet been developed, the women of ENIAC are nevertheless widely celebrated as the world’s first computer programmers. And not only was the pioneering work that they did on the ENIAC historically significant, many went on to subsequent careers – often at the highest levels – in electronic computing.

In his 1996 article based on interviews with the ENIAC programmers, Barkley Fritz highlights the substantial contributions of that these women made to the operation – and particularly the troubleshooting – of the prototype ENIAC computer. According to Betty Jean Jennings, for example, the “ENIAC girls” were trained to understand the internal wiring diagrams of the ENIAC machine, and “as a result we could diagnose troubles almost down to the individual vacuum tube. Since we knew both the application and the machine, we learned to diagnose troubles as well as, if not better than, the engineer.”

In a few cases the local craft knowledge that these female programmers accumulated significantly affected the design of the ENIAC and subsequent computers. ENIAC programmer Betty Holberton recalled one particularly significant episode when she convinced John von Neumann to include a “stop instruction” in the machine: although initially dismissive, von Neumann eventually recognized the programmer’s legitimate need for such an instruction. Other accounts by participants and observers echo the critically important – but generally unanticipated – role that the ENIAC programmers played in facilitating the successful launch of one of the world’s most famous early electronic computers. But as Jennifer Light has convincingly demonstrated, the contributions of these women was subsequently systematically eliminated from the historical record.

There is no question that the work of the ENIAC women was disregarded in large part simply because they were women. But almost as significant as their gender was their subordinate position as “software” workers in a hardware oriented development.

---

19 Light, op. cit.
project. Obviously the two are closely related. Of course, the use of the word “software” in this context is anachronistic – the word itself would not be introduced until 1958 – but hierarchical distinctions and gender connotations it embodies – between “hard” technical mastery and the “softer,” more social (and implicitly, of secondary importance) aspects of computer work – are applicable even in the earliest of electronic computing development projects. In the status hierarchy of the ENIAC project, it was clearly the male computer engineers who were significant. The ENIAC women, the computer “programmers,” as they would later be known, were expected to simply adapt the “plans of computation” already widely used in human computing projects to the new technology of the electronic computer. These “plans of computation” were themselves highly gendered, having been traditionally developed by women for women (human computing had been largely feminized by the 1940s). The ENIAC women would simply “set-up” the machine to perform these pre-determined plans: that this work would be turn out to difficult and to require radically innovative thinking was completely unanticipated. The telephone switchboard-like appearance of the ENIAC programming cable-and-plug panels reinforced the notion that programmers were mere machine operators, that programming was more handicraft than science, more feminine than masculine, more mechanical than intellectual.

The idea that the development of hardware was the real business of computing, and that software was at best secondary, persisted throughout the 1940s and early 1950s. In the very first textbooks on computing published in the United States, for example, John von Neumann and Herman Goldstine outlined a clear division of labor in computing – presumably based on their experience with the ENIAC project – that clearly distinguished between the “head-work” of the (male) scientist, or “planner,” and “hand-work” of the (largely female) “coder.” In the Goldstine/von Neumann schema, the “planner” did the intellectual work of analysis, and the “coder” simply translated this work into a form that a computer could understand. “Coding” was, according to von Neumann and Goldstine, a “static” process, one that could be performed by a low-level of clerical worker. “Coding” implied manual labor, and mechanical translation or rote transcription; “coders” were obviously low on the intellectual and professional status hierarchy. It was not unreasonable to expect that, as was the case in the ENIAC project, that most of these “coders” would be women.

An early manuscript version of the UNIVAC “Introduction to Programming” manual mirrored this distinction between “planner” and “coder,” analysis and implementation. In this instance the term “programmer” was used, somewhat unconventionally, in place of “planner”, but the distinction between the analytical “programmer” (the

---

person who "studies the problem, determines the appropriate method of solution, and prepares the flow chart") and the clerical "coder" (who "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") remains the same. In the UNIVAC manual, like the Goldstine/von Neumann textbook, the real business of programming was analysis: the actual coding aspect of programming was trivial and mechanical.

It was not until the early 1950s that the now commonplace title (and meaning) of "programmer" was widely adopted within the computing community. As David Grier has suggested, the verb "to program," with its military connotations of "to assemble" or "to organize," suggested a more thoughtful and system oriented activity. But even as the now commonplace designation "programmer" was increasingly adopted within the computing community, software workers would struggle to distance themselves from the status (and gender) connotations suggested by the older designation "coder." The accusation that programmers were "mere coders" (the adjective "mere" almost always being used to reinforce the low status of "coders") was used throughout the 1950s and 1960s (and indeed, up until the present) by those who wanted to counter the influence of "uppity" software workers. The noted computer scientist John Backus, for example, argued that the adoption of the title "programmer" by former "coders" happened "for the same reason that janitors are now called 'custodians': "Programmer was considered a higher class enterprise that 'coder, and things have a tendency to move in that direction."

The conflation of programming and coding, and the association of both with low-status clerical labor, suggested the ways in which early software workers were gendered female. In the ENIAC project, of course, the programmers actually were women. In this respect programming inherited the gender identity of the human computing projects in which it originated. But the suggestion that "coding" was low-status clerical work also implied an additional association with female labor. As Major Davies, Sharon Strom, and Elyce Rotella have described, clerical work had, by the second decade of the 20th century, become largely feminized. This was particularly true of clerical occupations that were characterized by the rigid division of labor and the introduction of new technologies. Some of these occupations carried over directly into the computer era: the job of keypunch operator, for example, had been thoroughly feminized long

---

14Grier, op. cit., p. 52.
before it became associated with electronic data processing. And although today we do not associate the work of key-punchers with the work of the computer programmer, in the 1950s and 1960s the differentiation between keypunch operator and other forms of computer work was not always clear. The Cosmopolitan article, for example, lumped keypunch operators in among the “computer girls,” and other contemporary sources identified keypunch operators as an obvious source of programmer trainees. In any case, the historical pattern of the 19th and 20th centuries has been that low-status occupations, with the exception of those requiring certain forms of physical strength, have often become feminized.

III The “Bad Boys” of Programming

Even by the early 1950s, however, computer programming began to acquire new status and a new gender identity. The experience of the ENIAC girls had shown that electronic computing was anything but the “automated form of hand computation” that it had been anticipated to be. The neat distinction made by Goldstine and von Neumann between analysis and implementation quickly broke down in actual practice. To begin with, since the primary purpose of the earliest computers was to produce solutions to complex mathematical functions that could not be solved analytically, the programmers of these computers necessarily required skill in numerical analysis. This process of analysis was itself something of an art form: numerical solutions always involved a compromise between speed and accuracy – even when using the fastest computers. Choosing the right approximation involved balancing acceptable error against the specific limitations of a given machine – a process that required daring, creativity, and mathematical intuition.

Perhaps even more significantly, the performance and memory constraints of the first generation of electronic computers demanded that programmers cultivate a series of idiosyncratic and highly individual craft techniques designed to overcome the limitations of primitive hardware. For example, contemporary memory devices were so slow and had such little capacity that programmers had to develop ingenious techniques to fit their programs into the available memory space. In order to coax every bit of speed out of a relatively slow storage device such as a rotating memory drum, programmers would carefully organize their coded instructions in such a way as to assure that the each instruction passed by the magnetic read head in just the right

order and at just the right execution time. Only the best programmers could hope to
develop applications that worked at acceptable levels of usability and performance.

For all of these reasons and more, programming in the 1950s acquired a reputation
for being incomprehensible to all but a small set of extremely talented insiders. As John
Backus would later describe it, “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
invention.” Techniques developed for one application or installation could not be easily
adapted for other purposes. There were few useful or widely applicable tools available
to programmers, and certainly no “science” of programming. Programmers often
worked in relative isolation, and had few opportunities for formal or even informal
education. They generally perceived little value in the work going on at other firms or
laboratories, as it was equally haphazard and idiosyncratic. They placed great emphasis
on local knowledge and individual ability.

The heady combination of mathematics, engineering “tinkering,” and arcane tech-
nique attracted a certain kind of male to the formerly feminized world of computer
programming. Some had abandoned careers in more established scientific disciplines
to pursue adventure in the emerging field of electronic computing. Others drifted
in from more closely related fields such as electrical engineering, or from careers in
business or data processing. A few, such as the physicist-turned-programmer Edsger
Dijkstra, worried about the lack of a “sound body of knowledge that could support it
[programming] as an intellectually respectable discipline.” The popular notion that
programmers were idiosyncratic geniuses, and that “a really competent programmer
should be puzzle-minded and very fond of clever tricks” was an pernicious anachro-
nism, Dijkstra would later argue, that encouraged a short-sighted, “tinkering” approach
to software development. Academically-minded programmers like Dijkstra felt that
too many of their colleagues regarded their work as temporary solutions to local
problems, rather than as an opportunity to develop a more permanent body of knowl-
dge and technique. What computing needed to realize its true revolutionary potential,
Dijkstra argued, was a more rigorous approach to programming, one modeled after the
science of applied mathematics. But most programmers accepted – and some reveled
in – the conventional belief that, at least for the conceivable future, programming
would remain the exclusive domain of the select few who possessed the “right stuff.”

This perception of programming as an idiosyncratic arcane discipline – and by
extension, its practitioners a “long-haired programming priesthood” – was reinforced
by a series of aptitude tests and personality profiles that suggested that computer pro-

39 Edsger Dijkstra. “Programming as a Discipline of Mathematical Nature”. In: American Mathematical
grammers, like chess masters or virtuoso musicians, were endowed with a uniquely creative ability. By the middle of the 1960s the majority of companies (80%) were using such tests and profiles as their primary tool for identifying programmer-trainees. "Creativity is a major attribute of technically oriented people," suggested one representative profile: "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." Many of the advertisements for programmers in this period reference chess-playing, musical ability, and mathematics very specifically. In 1956 IBM launched an advertisement for programmers that led to the hiring of such notable chessmen as Arthur Bisquier, the U.S. Open Chess champion, Alex Bernstein, a U.S. Collegiate champion, and Sid Noble, the self-proclaimed "chess champion of the French Riviera." (It should be noted, however, that the same campaign also netted an Oxford trained crystallographer, an English Ph.D. candidate from Columbia University, an ex-fashion model [female], and a "proto-hippie," so obviously chess-playing ability was not the sole criteria by which IBM evaluated its candidates.) In any case, good programming was believed to be dependent on uniquely qualified individuals, and that what defined these uniquely individuals was some indescribable, impalpable quality—a "twinkle in the eye," an "indefinable enthusiasm," or what one interviewer described as "the programming bug that meant ...we're going to take a chance on him despite his background."

In addition, great disparities were discovered between the productivity of individual programmers, with one widely cited IBM study suggesting that code produced by a truly excellent programmer was twenty-six times more efficient than that produced by his merely average colleagues. 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. "When a programmer is good, he is very, very good. But when he is bad, he is horrid," the study declared, reinforcing the notion that skilled programmers were thought to be effectively irreplaceable, and were to be treated and compensated accordingly. Programmers were to be selected for their intellectual gifts and aptitudes, rather than their business knowledge or managerial savvy.

---

The notion that programming was a “black art” pervades the literature from the early decades of computing. Even today, more than a half-century after the invention of the first electronic computers, the notion that the computer programming still retains an essentially “artistic” character is still widely accepted. Whether or not this is desirable is an entirely different question – one which remains a subject of considerable and contentious debate. What is important is that by characterizing the work that they did as “artistic,” programmers could lay claim to the autonomy and authority that came with being an artist. Note that the appeal here is to the tradition of the artisan, or craftsman, which is a masculine identity, than it is to the potentially effeminate “artsy” type.

The widespread perception that programming ability was an innate ability, rather than an acquired skill or the product of a particular form of technical education, could be seen as gender-neutral, or even female-friendly. After all, the aptitude tests for programming ability were widely distributed among female employees, including clerical workers and secretaries. And, according to one 1968 study, it was found that a successful team of computer specialists included an “ex-farmer, a former tabulating machine operator, an ex-key punch operator, a girl who had done secretarial work, a musician and a graduate in mathematics.” The last [the mathematician] “was considered the least competent.” As hiring practices went, aptitude testing at least had the virtue of being impersonal and seemingly objective. Being a member of the “old boys club” does do much for one’s scores on a standardized exam.

But the aptitude tests and personality profiles did embody and privilege masculine characteristics. For example, despite the growing consensus within the industry (particularly in business data processing) that mathematical training was irrelevant to the performance of most commercial programming tasks, popular aptitude tests such as the IBM PAT still emphasized mathematical ability. Some of the mathematical questions tested only logical thinking and pattern recognition, but others required formal training in mathematics – a fact that even *Cosmopolitan* recognized as discriminating against women.

---


48 Except to the extent that fraternities and other male social organizations served as clearinghouses for stolen copies of popular aptitude tests such as the IBM PAT. Such theft and other forms of cheating were rampant in the industry, and taking the test more than once was almost certain to lead to a passing grade.

Even worse were the personality profiles. The use of personality profiles to identify programmers began, as with other industry-standard recruiting practices, at the System Development Corporation (SDC), the RAND Corporation spin-off charged with the development of the software for the SAGE air-defense system. Faced with the need to train computer programmers in unprecedented numbers – in 1956 SDC employed 700 programmers, almost three-fifths of the total number of programmers available world-wide, and by the beginning of the 1960s had trained 7,000 more – SDC relied extensively on aptitude testing and personality profiling. By the beginning of the 1960s, however, SDC psychologists had developed more sophisticated models based on the extensive employment data the company had collected over the previous decade, as well as surveys of members of the Association of Computer Machinery and the Data Processing Management Association. In a series of papers published in serious academic journals such as the *Journal of Applied Psychology* and *Personnel Psychology*, SDC psychologists Dallis Perry and William Cannon provided a detailed profile of the “vocational interests of computer programmers.” The scientific basis for their profile was the Strong Vocational Interest Bank (SVIB), which had been widely used in vocational testing since the late 1920s.

The basic SVIB in this period consisted of four hundred questions aimed at eliciting an emotional response (“like”, “dislike”, or “indifferent”) to specific occupations, work and recreational activities, types of people, and personality types. By the 1960s, more than fifty statistically significant collections of preferences (“keys”) had been developed for such occupations as artist, mathematician, policeman, and airplane pilot. The assumption behind the use of such profiles was that candidates who had interests in common with those individuals who were successful in a given occupation were themselves also likely to achieve similar success.

Many of the traits that Perry and Cannon attributed to successful programmers were unremarkable: for the most part programmers enjoyed their work, disliked routine and regimentation, and were especially interested in problem and puzzle-solving activities. The programmer key they developed bore some resemblance to the existing keys for engineering and chemistry, but not to those of physics or mathematics, which Perry and Cannon saw as contradicting the traditional focus on mathematics training in programmer recruitment. Otherwise, programmers resembled other white-collar professionals in such diverse fields as optometry, public administration, accounting, and personnel management.

In fact, there was only one really “striking characteristic” about programmers that the Perry and Cannon study identified. This was “their disinterest in people.” Compared

---

11 Ibid.
with other professional men, “programmers dislike activities involving close personal interaction. They prefer to work with things rather than people.”52 In a subsequent study, Perry and Cannon demonstrated this to be true of female programmers as well.53

The idea that computer programmers lacked “people skills” quickly became part of the lore of the computer industry. The influential industry analyst Richard Brandon argued that this was in part a reflection of the selection process itself, with its emphasis on mathematics and logic. The “Darwinian selection” mechanism of personnel profiling, Brandon suggested, selected for personality traits that performed well in the artificial isolation of the testing environment, but which proved dysfunctional in the

---

52Ibid.
more complex social environment of a corporate development project. Programmers were “excessively independent,” argued Brandon, often to the point of mild paranoia. The programmer type is “often egocentric, slightly neurotic, and he borders upon a limited schizophrenia. The incidence of beards, sandals, and other symptoms of rugged individualism or nonconformity are notably greater among this demographic group. Stories about programmers and their attitudes and peculiarities are legion, and do not bear repeating here.”

![Figure 2: Datamation Cartoon, 1963](image)

Needless to say, these psychological profiles embodied a preference for stereotypically masculine characteristics. A 1970 review of the psychometric literature noted that computer programmers received unusually high masculinity and low femininity scores. In fact, only four occupational groups received higher masculinity scores (unfortunately, the review does not mention which four). “These consistent results [high masculinity scores] define one characteristic of the people in data processing jobs,” the

---

review concluded – namely, their masculine self-identity.55

The idea that “detached” (read male) individuals made good programmers was embodied, in the form of the psychological profile, into the hiring practices of the industry.56 Possibly this was a legacy of the murky origins of programming as a fringe discipline in the early 1950s; perhaps it was self-fulfilling prophecy. Nevertheless, the idea of the programmer as being particularly ill-equipped for or uninterested in social interaction did become part of the conventional wisdom of the industry. Although the short-term effects of this particular occupational stereotype was negligible, it would later come back to haunt the programming community as it attempted to professionalize later in the decade. In any case, it effectively excluded most women from the discipline. According to one survey of Canadian employers, more than two-thirds used a combination of aptitude and general intelligence tests, personality profiles, and interest surveys in their selection processes.57

One interpretation of the male bias embedded in these aptitude tests and personality profiles is that such tests are, in fact, an accurate reflection of the mental or emotional characteristics that make for a good programmer – logical, detached, anti-social – and that these traits just happen to be more predominant in males. This is the essentialist argument: gender discrimination as a function of biology. Even in the 1960s and 1970s there seemed little evidence for such reductionist explanations.58

Another interpretation is that programming ability has no correlation at all with biologically-determined predispositions, but that the widespread use of gender-biased testing regimes by industry employers nevertheless did create a feedback cycle that ultimately selected for programmer with stereotypically masculine characteristics. The primary selection mechanism used by the industry selected for anti-social, mathematically inclined males, and therefore anti-social, mathematically inclined males were over-represented in the programmer population; this in turn reinforced the popular perception that programmers ought to be anti-social and mathematically-inclined (and therefore male), and so on ad infinitum. This would be an historically continent argument: gender discrimination as a function of historical accident.

A third interpretation is that the tests were developed deliberately to exclude women from an increasingly high-status, lucrative, and therefore male-dominated profession. This is the conspiratorial, or at least inadvertently hegemonic, argument.

My own view is that, in the case of aptitude testing and personality profiling, that

the privileging of masculine characteristics is the result of some combination of laziness, ambiguity, and traditional male privilege. There was widespread evidence, even in the late 1960s, that psychometric testing was inaccurate, unscientific, had been widely compromised, and was a poor predictors of future performance. Nevertheless, these methods continued to be used simply because they were convenient. The rapid expansion the commercial computer industry in the early 1960s demanded the recruitment of large armies of new professional programmers. At the same time, the general lack of consensus about what constituted relevant knowledge or experience in the computer fields undermined attempts to systematize the production of programmers. Commercial programming schools were seen as being too lax in their standards, the emerging academic discipline of computer science too stringent. Neither was believed to be a reliable short-term solution to the burgeoning labor shortage in programming. In the face of such uncertainty and ambiguity, aptitude testing and personality profiling promised at least the illusion of managerial control. To borrow a phrase from contemporary computer industry parlance, aptitude testing was a solution that scaled efficiently. That is to say, the costs of aptitude testing grew in a predictable, linear relationship to the number of applicants (as opposed to other recruitment methods such as personal interviews, whose costs in time and money grew rapidly.) Put even more simply, it was possible to administer aptitude tests quickly and inexpensively to thousands of aspiring programmers. Compared to such time-consuming and expensive alternatives such as individual interviews or formal educational requirements, aptitude testing was a cheap and easy solution. And since the contemporary emphasis on individual genius over experience or education meant that a star programmer was as likely to come from the secretarial pool as the engineering department, the ability to screen large numbers of potential trainees was preeminent.

But the kinds of questions that could be easily tested using multiple choice aptitude tests and mass-administered personality profiles necessarily focused on mathematical trivia, logic puzzles, and word games. The test format simply did not allow for any more nuanced or meaningful or context-specific problem solving. And, in the 1950s and 1960s at least, such questions did privilege the typical male educational experience. Again, this bias towards male programmers was not so much deliberate as it was convenient. The fact that the use of lazy screening practices inadvertently excluded large number of potential female trainees was simply never considered. But the increasing assumption that the average programmer was also male did play a key role in the establishment of a highly masculine programming subculture.
IV Professionalization = Masculinization

The process of making programming masculine did not begin – or end – with the transformation of the staid, feminized clerical work of "coding" into the highly masculine, seat-of-the-pants "black art" of programming of the 1950s, not even with the embodiment of certain masculine values into the hiring procedures of the industry. To begin with, this transformation was never fully complete. There were still aspects of the programming process that remained rote, mechanical, and low-status. It was also not clear that the frontier mentality of programming culture in the 1950s was anything but a function of the immaturity of the industry. The influx of new programmer trainees and vocational school graduates into the software labor market had only exacerbated an already bad labor situation. The market was flooded with aspiring programmers with little training and no practical experience. As one study by the Association for Computing Machinery’s Special Interest Group on Computer Personnel Research (SIGCPR) warned, by 1968 there was 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,” the SIGCPR argued. “Educational, performance and professional standards are virtually nonexistent and confusion growths rampant in selecting, training, and assigning people to do jobs.”

At the same time that the demand for skilled programmers was increasing dramatically (and seemingly without limit), when salaries and opportunities for occupational mobility were at their peak, many programmers were plagued with uncertainty about the status and future of their discipline.

There were tangible reasons for this uncertainty. The increasing capabilities and reliability of second generation hardware meant that the baroque “work arounds” and optimizations so prized by programmer-tinkerers were no longer necessary. In addition, the development of so-called “automatic programming systems” threatened to make programmers obsolete altogether, and to return responsibility for the “head work” involved in problem analysis back to the scientists and managers, where it belonged. The persistent lack of programmers to develop a “scientific” basis for their discipline suggested that they were at best artisans or technicians, the last vestiges of a “pre-industrial” approach to software development. The organizational tensions provoked by the increasing use of computerized systems for managerial purposes

60 The most damning critique of the “black art” of programming came from Douglas McIroy at the 1968 NATO Conference on Software Engineering: “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 considerable higher, and would like to investigate the prospects for mass-production techniques in software.”
created resentment against the perceived “abdication” of management imperatives to whiz-kid “computer boys.” These tensions reflected themselves in active attempts by managers to reassert their traditional authority over computer programmers by redefining their work as “merely” technical. Finally, the rising cost of software relative to hardware meant that firms began looking for ways to reduce costs by “rationalizing” their development practices. And according to recent experience in traditional manufacturing, such “rationalization” often meant the incorporation of a less-expensive, lower-skill (read feminized) work force.

There is considerable debate within the historical literature about the degree to which corporations, academics, and other reformers were able to rationalize the practices of computer programmers in response to the emerging “software crisis” of the late 1960s. In his 1977 book *Programmers and Managers: The Routinization of Computer Programming in the United States*, the historian Philip Kraft argued that managers had, in fact, been successful in “degrading” the work of computer specialists. "Programmers, systems analysts, and other software workers,” Kraft argued, were the victims of efforts to “break down, simplify, routinize, and standardize” their work practices. Building on the work of labor historians such as Harry Braverman (and, of course, Karl Marx), Kraft suggested that corporate managers have generally been successful in imposing structures on programmers that have eliminated their creativity and autonomy. His analysis was 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). In 1979 Joan Greenbaum echoed Kraft’s conclusions, arguing that “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.” Unlike Kraft, Greenbaum made an explicit connection between routinization and feminization. More recently, other scholars have suggested that the connection between routinization, feminization, and the increasing of foreign labor in software development (“outsourcing”).

It is questionable how successful corporate managers and other “rationalizers” were in their quest to transform software development into a controlled, industrial manufacturing process. Software development remains a uniquely artisanal form of craft

---

production, and computer programmers are, on the whole, well-paid, highly-valued, and largely autonomous professionals. But at the same time, it is also clear that many programmers in the 1960s were worried about the possibility of having their work routinized at degraded. Certainly the management literature from this period is full of confident claims about the ability of new performance metrics, development methodologies, and automatic programming languages to reduce corporate dependence on individual programmers. As Michael Cusamano has described, the vision of the "software factory" – in which hordes of low-paid, low-skill programmers cranked out mass-produced software products – was a persistent theme in this literature. It was not unreasonable, as Kraft has suggested, to assume that "elaborate efforts" were being made "to develop ways of gradually eliminating programmers, or at least reduce their average skill levels, required training, experience, and so on." The history of technology in the 20th century is replete with such stories.

One of the time-honored strategies for dealing with labor “problems” in the United States has been the use of female workers. There is a vast historical literature on this topic: from the very origins of the American industrial system women have been seen as a source of cheap, compliant, and undemanding labor. The same dynamic was a work in computer programming. In a 1963 Datamation article lauding the virtues of the female computer programmer, for example, Valerie Rockmael focused specifically on her stability, reliability, and relative docility: “Women are less aggressive and more content in one position … Women consider fringe benefits of more importance than their male peers and are more prone to stay on the job if they are content, regardless of a lack of advancement. They also maintain their original geographic roots and are less willing to travel or change job locations, particularly if they are married or engaged.” In an era in which turnover rates for programmers averaged twenty-percent annually, this was a compelling argument for employers, since their substantial initial expenditures on training "pays a greater dividend" when invested in female employees. Note that this was something of a backhanded compliment, aimed more at the needs of employers than female programmers. In fact, the "most undesirable category of programmers," Rockmael argued, was “the female about 21 years old and unmarried,” because “when

---

67 See, for example, Kasson, Civilizing the Machine: Technology and Republican Values in America, 1776-1900, (1976); Milkman, Gender at work: the dynamics of job segregation by sex during World War II, (1987); Kessler-Harris, Out to work: a history of wage-earning women in the United States, (1982); and indeed, most of the rest of the history of labor and technology
The story of how a single language answers the question, "Can a young girl with no previous programming experience find happiness handling both commercial and scientific applications, without resorting to an assembler language?"

Let's face it. The cost of programming just keeps going up. So for some time to come, how well you do your job depends on how programmers like Susie Meyer do theirs.

That's the reason for PL/I, the high-level language for both scientific and commercial applications.

With PL/I, programmers don't have to learn other high-level languages. They can concentrate more on the job, less on the language.

So think about PL/I. Not just in terms of training, but in terms of the total impact it can have on your operation.

Figure 3: IBM Corporation, Susie Meyers Meets PL/1, 1968
she would start thinking about her social commitments for the weekend, her work suffered proportionately.68

In addition to this emphasis on the compliance of female programmers, women were often used in advertisements from this period as a visual proxy for low-skill, low-wage labor. In its "Meet Susie Meyers" advertisements for its PL/1 programming language, for example, 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?" (see figure 3) 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 was non-too-subtle: 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. The suggestion that women were the least capable form of computer user is, of course, still a regular feature of the computer industry literature, where the frequently made claim that a technology “is so simple that even my grandmother could use it” continues to associate gender identity with perceived competence.

These attempts to mobilize gendered rhetoric (or visuals) in the service of what one contemporary described as the “the domestication of this once proud, wild animal” (the computer programmer) did not go unnoticed by programmers.69 The publication of "The Computer Girls" article, for example, prompted an almost immediate response in the form of an advertisement from the Computer Sciences Corporation. Although the overlying tone of the article was light-hearted – “In a recent issue of COSMOPOLITAN, Helen Gurley Brown exhorted her girl readers to become programmers and make 15,000 after five years …” – the underlying concern it expressed was also quite apparent: the suggestion that “Cosmo Girls” could make for good programmers was implicitly demeaning, and threatening to the status and future of the discipline.70

I have written extensively elsewhere about the "Question of Professionalism" as it emerged in the computer fields during the late 1960s.71 For the purposes of this paper it is enough to note that the development of the structures of a programming profession – including formal programs in academic computer science, professional journals and

---

70 Computer Sciences Corporation. "In case you missed our first test...”. In: Datamation 13.9 (1967). P. 149.
societies, and professional certification programs – became the goal of many computer programmers, and their corporate employers, as a means of addressing the perceived “software crisis” of the late 1960s.

The professionalization of programming and other computer specialties was appealing to a number of constituencies. For practitioners, professionalism offered increased social status, greater autonomy, improved opportunities for advancement, and better pay. It provided individuals with a “monopoly of competence” – the control over a valuable skill that was readily transferable from organization to organization - that provided leverage in the labor market. 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.

The 1960s were a period when many white-collar occupations were pursuing professional agendas, and the sociological literature of period seemed to provide a clear road-map to the benefits of professionalism. It appeared to many that these benefits were available to almost any occupation, assuming only that they followed the appropriate road-map.

The professionalization efforts of computer specialists were, to a certain extent, encouraged by their corporate employers. Professionalism, or at least a very specific form of corporate-friendly professionalism, provided 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 … The professional's rewards are full utilization of his talents, the continuing challenge and stimulus of new EDP situations, and an invaluable broadening of his experience base.” Insofar as it encouraged good corporate citizenship, professionalism had the potential to solve a number of pressing management problems: it might motivate staff members to improve their capabilities; it could bring about more commonality of approaches; it could be used for hiring, promotions and raises, and it could help solve the perennial question of “who is qualified.” At the very least, allowing programmers think that they were professionals would go a long way towards reducing turnover and maintaining the stability of the

data processing staff.\textsuperscript{77}

The desire to develop professional standards is an understandable, and indeed laudable, agenda for programmers to pursue. But it does carry with it certain implications for the gender dynamics of the discipline. As Margaret Rossiter and others have suggested, professionalization implies masculinization.\textsuperscript{78} The imposition of formal educational requirements, such as a college degree, can make it difficult for women – particularly women who have taken time off to raise children – to enter the profession. Similarly, certification programs or licensing requirements – such as the Data Processing Management Association’s Certificate in Data Processing Program – also erected barriers to entry that disproportionately affected women. Professionalism also suggests a certain degree of managerial authority and competence – skills and characteristics that were often seen as being masculine rather than feminine. In his book \textit{The Psychology of Computer Programming}, for example, Gerald Weinberg notes (and dismisses) the commonly held belief that female programmers were incapable of leading a group or supervising their male colleagues.\textsuperscript{79} The more than programmers were seen as potential managers (a new development that came with professionalization), the more women were excluded.

There were other, more subtle ways in which professionalization implied masculinization. As aspiring computer scientists struggled to construct for themselves a legitimate academic identity, they increasingly focused on the mathematical and theoretical aspects of their discipline, a strategy that to this day is seen as discouraging female participation.\textsuperscript{80}

Perhaps most significantly, professionalization requires segmentation and stratification. In order to elevate the overall status of their discipline, aspiring professionals had to distance themselves from those aspects of their work that were seen as low-status and routine. This work did not just disappear – it was just done by other people. The general job category of “programmer,” which from the early 1950s on had been used as a blanket term to describe a broad range of computer workers, was increasingly replaced by a complicated hierarchy of job titles: junior programmer, senior programmer, lead programmer, junior analyst, senior analysts, program manager, etc. Again, it is difficult to gather accurate statistics on who occupied what categories, but there is some evidence to suggest that women were generally confined to the lower levels of the

\textsuperscript{79}Gerald Weinberg, \textit{The Psychology of Computer Programming}, p. 85.
professional pyramid. This calls into question the more optimistic claims about the participation of women in computing: without knowing exactly what kinds of work these women were doing, it is difficult to draw any firm conclusions about the true nature of the opportunities available to women in computing. What is most important is that the stratification of the programming profession seemed to involve the demeaning of female programmers. Gerald Weinberg claims, for example, that insecure male programmers frequently belittled female colleagues, who, unlike secretaries, “did not know their place.”

Finally, there is the fact that sexist attitudes towards women formed part of the background noise of most technical professions in this period. Women were frequently represented in advertisements in the computing literature as sex-objects, low-level secretaries, “naive” users, or at best, potential distractions. One series of advertisements for optical character recognition equipment that ran in the journal Datamation throughout 1967 was particularly misogynistic: each issue played on a variation on the theme “our optical readers can do anything your [female] keypunch operators can do,” with additional qualifications such as “except get pregnant,” “or complain of being tired all the time,” or “make time on company time,” or “be social butterflies.” (see figures 4 and 5) It is impossible to estimate the effect that this persistent low-level sexism had on women hoping to make a career in programming. Computing was by no means uniquely sexist in this respect, but neither was a particularly enlightened oasis from prevailing social attitudes.

V Conclusions

No historical study of gender in computing can afford to ignore the role of women in the history of computer programming. The contributions of female programmers to this history, significant in their own right, are even more remarkable given the prevailing social norms of the era (particularly in technical occupations). But to a certain degree our perception that presence of women in programming is premised on the assumption that programming has always been an essentially masculine discipline. But in the early decades of electronic computing, the gender identity of programming was ambiguous: a field that started out feminized was only gradually – and fitfully – made masculine. Much of the social work necessary to accomplish this occupation sex-change was couched in terms of professional development. In programming, as in many occupations, professionalism was masculinization. In the decades following the

---


1960s, of course, computer programming has become a hyper-masculine profession. But the origins of this are social rather than biological, historically contingent rather than inevitable.
Our optical reader can do anything your keypunch operators do.

(Well, almost.)

It can't make time on company time. Or use the office for intimate tête-à-têtes. Or be a social butterfly. But it can read. And gobble data at the rate of 2400 typewritten characters a second. And compute while it reads. And reduce errors from a keypunch operator's one in a thousand to an efficient one in a hundred thousand.

Our machine reads upper and lower case characters in intermixed, standard type fonts. It handles intermixed sizes and weights of paper, including carbon-backed sheets.

An ordinary computer program tells our reader what to do. . . . to add, subtract, edit, check, or verify as it reads. Lets you forget format restrictions, leading and trailing zeros, skipped fields, and fixed record lengths. And our reader won't obsolete any of your present hardware because it speaks the same output language as your computer.

Our Electronic Retina Computing Reader can replace all—or almost all—of your keypunch operators. At least that's what it is doing for United Airlines.

If you have a volume input application, it can do the same for you. Tell us your problem and we'll tell you how.

RECOGNITION EQUIPMENT Incorporated
Our optical reader can do anything your keypunch operators do.

(Well, almost.)

It can’t take maternity leave. Or suffer from morning sickness. Or complain of being tired all the time. But it can read. And gobble data at the rate of 2400 typewritten (or hand printed) characters a second. And compute while it reads. And reduce errors from a keypunch operator’s one in a thousand to an efficient one in a hundred thousand.

Our machine reads upper and lower case characters in intermixed, standard type fonts. It can handle intermixed sizes and weights of paper, including carbon-backed sheets.

An ordinary computer program tells our reader what to do . . . to add, subtract, edit, check or verify as it reads. Lets you forget format restrictions, leading and trailing zeros, skipped fields, and fixed record lengths. And our reader won’t obsolete any of your present hardware because it speaks the same output language as your computer.

Our Electronic Retina Computing Reader can replace all—or almost all—of your keypunch operators. At least that’s what it is doing for American Airlines.

If you have a volume input application, it can do the same for you. Tell us your problem and we’ll tell you how.