II

Programming as an Exceptional Science of Artifice and Complexity

The plenitude of computer power was used to invent computer programming as a science of artifice and complexity. That is, on the one hand, programming was unhampered by external realities. On the other hand, computer programming was expressly constructed in such a way as to control complexity.

The most important means for the conquest of complexity is the act of partitioning a complex system into a number of simpler subsystems, with well defined interfaces, to be dealt with independently. In natural science, this contains an element of distortion, but in the artificial science of computer programming, a proposition is true if it is declared to be true. So a subsystem is independent if it is declared to be so. Thus, in computer programming, there is no limit to the construction of independent subsystems, commonly called subroutines.[1]

---

[1] Herbert A. Simon, in the The Sciences of the Artificial (2nd ed; The MIT Press, The Massachusetts Institute of Technology; Cambridge, Mass., 1969,1981), provides a general development of the idea of complexity, and of the place of artificial intelligence and computer science as sciences oriented around the conquest of complexity. As Simon points out these sciences are perfectly artificial ones; hence they allow perfect contingency and possibility of choice, unconstrained by physical reality.

Gerald Weinberg, in The Psychology of Computer Programming (Van Nostrand Reinhold Company, 1971), treats the problem of complexity, especially in the form of the linguistic issues of programming language design, as the only meaningful aspect of the psychology of programming.

The mastery of complexity, and the freedom of choice which followed from that, had the result that programming would be an outward-looking technology rather than an inward-looking one. Unresolved complexity was not required for computer programming's internal functioning, in striking contrast to the situation in engineering. So it was possible for computer programming to embrace additional complexity, reaching out to accept human institutions in all their irrationality.

Most immediately, the choice which grew out of programming's artificiality was exercised in such a way as to minimize constraints on the programmer. As impediments to programming emerged, they were systematically eliminated by developing better intermediate software. There were two aspects to this elimination of impediments. One was a collection of practical matters; the other was the controlling and reduction of complexity.

The practical matters were not individually very important, but there were a number of them. Programming is remarkably free of all kinds of physical and practical constraints which restrict the role of women. After all, it is little more than the use of language, a form of writing. Programming does not require physical strength or agility, even in an ancillary capacity. As Elizabeth Baker[1] has pointed out, even ancillary and indirect requirements for strength can greatly reduce a woman's employability. Likewise programming is not filthy work, nor is it even in any sort of contact with filthy work. Nor does it have

---

[1] Elizabeth Faulkner Baker, Technology and Women's Work, Columbia University Press, New York, 1964, pp. 44, 80-82

any connection with hazardous work. There has been very little restriction of time, place, and schedule, on the whole.[1] Before the development of computer terminals, which can be used from anywhere there is a telephone line, programming was essentially a paper process, involving a modest quantity of papers at any given time. Programmers read print-offs and manuals, and wrote on coding forms. All of these could be taken home in a briefcase. Programming is compatible with a conventionally feminine lifestyle-- or in fact, almost any other desired lifestyle.

These are all significant factors, of course, but as Baker points out, the single greatest restriction on women is their role as wife and mother. Because they might be obliged to leave work in order to meet family obligations, women have tended to receive less training, both as a result of their own decisions whether or not to invest in schooling or low-paid apprenticeships, and also as a result of the decisions other people make about whether to admit women to the more immediately remunerative training programs.[2] Complexity becomes a barrier to access, because most women do not have time to learn the complexity.

---

[1] A special case would be "hacking," that is, programming as an artist or virtuoso. Hacking often involved staying up all night in order to gain access to a computer at a time when nobody else wanted it. This, however, was only a problem for those who were a) doing unremunerative work (and could therefore not demand an allocation of computer time via normal channels) and b) using the computer in a significantly more interactive fashion than was customary at the time. Even in this case, it might well be more advantageous to work at home, in order to minimize the strain of late hours.

[2] Baker, op. cit., pp. 438-440

Here, too, programming's symbolic nature comes to the fore. As a constructed symbol system, programming has almost from the beginning been exceedingly well internally partitioned. In no other technical field does so little training yield so great a harvest of effective skilled ability. A programmer with a little training can draw on a vast store of 'canned' skill embodied in the operating system, compiler, subroutine libraries, etc. Computer software, especially in what ultimately became known as the 'object-oriented' mode, is the perfect embodiment of Isaac Newton's famous remark to the effect that 'if he saw further, it was because he stood on the shoulders of giants.' However, the giants of computer programming have not only offered their shoulders to stand on, but have also provided ladders to enable one to clamber up more easily.

A programmer needs far less knowledge to create a mundane program-- let us say, a bookkeeping program-- than an engineer needs to create an equally mundane manufactured object. In order to make a bookkeeping program, it may well be necessary to sort records. It is not a simple matter to sort in an efficient way, that is, in a way which does not become prohibitively expensive as the number of records increases. There is an elegant and devious solution, known as the 'divide-and-conquer' algorithm. It is typically packaged up as a subroutine, which one can use without understanding it. The most advanced programming languages and operating systems have tended to increase the range and variety of expertise which can be thus packaged up for use by the unskilled.[1]

Another argument, related but more debatable, grows out of Sally Hacker’s claim[2] that the role of calculus in engineering school is to function as a gatekeepers, excluding women who will not think "male." Regardless of the claim’s merits, which would cause many woman mathematicians to make rude remarks, calculus is precisely the sort of body of intercessory knowledge which, in computer software, gets submerged or substituted for. That is, calculus can be approximately described as a body of recurring mathematical problems, and the knowledge of how to solve these problems can be embodied in computer programs, or subroutines. Invoking these programs takes far less mathematical knowledge

---

1. Broadly speaking, better programming languages permit one to program in a language more natural to one; and one which contains added vocabulary appropriate to the task at hand. Better operating systems allow one to do so without worrying about undesirable interactions with other persons, programs, or data. Where interaction is necessary, the operating system manages the details of it. Of course, to further complicate the picture, some programming languages (e.g. LISP, SIMSCRIPT, PROLOG) have their own internal operating systems, for the purpose of managing interactions according to special rules peculiar to those languages.

2. Sally Hacker, <u>Pleasure, Power, and Technology: Some Tales of Gender, Engineering, and the Cooperative Workplace</u>, Unwin Hyman, Boston, 1989

Hacker reports: "I finished one calculus exam and followed a young woman out the door. She threw up in the bushes... One young man... loved mathematics as he did life itself. But he could not pass calculus tests under pressure of time. He dropped out... [A Statics professor remarked] ’If we gave the students more time, anyone could do it. The secretaries could even pass it.’" (p. 41-42)

Hacker stresses that this is an ancestral tradition in engineering school: "Mathematics teaching and testing continued to perform the weeding function suggested in earlier debates... accounting for 72 percent of mid [nineteenth] century West Point failures" (p. 66) than doing calculus unaided.

Programming imposed few constraints on the programmer. Programmers could work at a time and place, and in a manner, best suited to their needs, and they did not need to be encyclopedically trained. In all these respects, programming was quite opposite to engineering.

By contrast with programming, engineering is comparatively unpartitioned. Because the objects designed by a traditional engineer cannot dance on pinheads, there does not exist the luxury of partitioning them. The famous Rube Goldberg cartoons are a good example of what a well partitioned machine would look like. It would also be grossly inefficient, unreliable, etc. Each component contains energy; has weight; occupies space, which another object cannot simultaneously occupy. Each component is subject to friction, structural fatigue, and other deterioration (rust, corrosion, rot, or even worms, according to what material the component is made of). And so on, in a catalog of the ills physical substance is prone to. In short, each component *costs*. The comedy of a Rube Goldberg machine lies precisely in the extravagant disproportion between costs and results. It was precisely this inefficiency and unreliability which Charles Babbage encountered when trying to build a mechanical computer.

Typically, a single component of a device built by an engineer must serve several unrelated functions, in the name of efficiency. This variety of function is not universality; it is not a case of doing several things by being able to do anything, as a computer does; the multi-functionality of the typical mechanical component is a more impoverished quality. The component is designed to be fitted for several specific functions, often having to trade-off quality in one aspect for quality in another. For example, an airplane’s wing is simultaneously: an airfoil; a loadbearing structure, not unlike a bridge; a fuel tank; and an equipment locker to hold landing gear, flaps, etc. Or rather, the wing is an uneasy compromise between all of these. There is not the luxury of thinking only about structural strength, or only about aerodynamics. The same principle applies for every built object down to a transistor radio.

To create a quite mundane and conventional artifact, with no real novelty to speak of, one may require knowledge of many collateral sciences. Even those branches of engineering, such as electrical engineering, which are intellectually closer to computer programming [1] are still embedded in engineering. Electrical engineering students are required to meet the general requirements of the engineering school, in which the department of electrical engineering is embedded. That is, under the rubric of ’engineering fundamentals,’ they must cover the equivalent of an undergraduate major in physics, with great chunks of effort put into subjects like chemistry, physical chemistry, thermodynamics, hydraulics, etc., which are very remote from the

---

1. Paul Ceruzzi, in "Electronics Technology and Computer Science, 1940-1975: A Coevolution" (Annals of the History of Computing, 1989, 10[4]:257-275), lays stress on the extension of the idea of complexity from computer science into electrical engineering, and the progressive reduction of electrical engineering to a science of information processing.

new information science oriented conception of electrical engineering.

The result is that the engineering curriculum is long and difficult. Technical coursework proliferates, driving out general education. Further, unlike much of undergraduate liberal arts education, this coursework is conducted on pedagogically sound principles, with frequent oral and written examination, written assignments, minimal use of multiple choice exams, etc. These practices are not unique to engineering: they can be found in the better sort of teaching manual;[1] These are proven methods for getting even very young and immature students to work every day. What distinguishes the engineering school from the large liberal arts college is that the engineering school programmatically applies these conventional methods to all its students, not just the promising few. Some of the least promising students respond by cribbing, of course, as Sally Hacker notes,[2] but the conventional methods of pedagogy were evolved over hundreds of years to keep cribbing within tolerable bounds, and to divert it into a form of study. There is nothing in engineering school remotely comparable to the term paper purchased from a mail-order catalog. That sort of outright cheating can only happen in an instructional system primarily designed to avoid inconveniencing

---

[1] For example, see Gilbert Highet, <u>The Art of Teaching</u>, 1950., pp. 118-24. Highet reflects the standard of instruction as conducted in an English Public School and at Oxford and Cambridge. As such, he goes rather beyond the undergraduate level practice of an American engineering school.

[2] Hacker, op. cit., p. 41

the faculty. While Computer Science courses, and indeed, nearly all the mathematics and hard science courses[1] in the liberal arts college, are pedagogically sound, there are far fewer of them. A Computer Science major is free to take large numbers of freshman and sophomore-level liberal arts survey courses. And for that matter, it is by no means necessary to be a Computer Science major to get into computer programming. But engineering school does not permit that sort of compromise. An ill-partitioned discipline requires a long and difficult curriculum, which requires a systematic teaching method demanding commitment from the students.

So the most important casualty of the system of engineering education is not liberal education, but the student’s free time. By judicious use of the cult of machismo,[2] the engineering school is able to draw boys away from profitless beer-drinking, and eventually make them into "four-year-bench-engineers." The system is quite unsuccessful in attracting girls, however. Most girls are not susceptible to machismo. Considerable numbers of coeds, more than is generally admitted, have come to college to get their "Mrs." As Louise Kapp Howe points out, most women like housework, for the very good reason that it offers greater

---

[1] The typical rare exception might be a course which is stipulated in the college catalog as not counting towards a degree in the field, and which is intended for students who did not take the appropriate preparatory subjects in high school. However, by no means all of such courses are pedagogically unsound, only some of them.

[2] Hacker (op. cit., p. 43) refers to "...this appeal, which seduces largely working-class men into the Green Berets or the paratroops in similar ways."

control, choice, autonomy, etc., than all but a handful of paid employments.[1] By extension, the primary goal of many coeds is the making of a satisfactory marriage. They can be gotten to take typing and shorthand, "just in case," but they cannot be gotten to sign up for initiation into a professional cult.[2]

Another aspect of the unpartitioned nature of engineering is that artifacts are not usually manufactured by a conventional and standard process. Software is simply copied, automatically, onto tapes, disks, etc. A programmer simply does not have occasion to think about ’design for manufacturability,’ and a programmer who

---

1. Louise Kapp Howe, Pink-Collar Workers: Inside the World of Women’s Work, Avon Books, 1978, orig. pub. 1977, p. 205-209, citing Carolyn Groo Jarmon, "Relationship Between Homemakers’ Attitudes Towards Specific Household Tasks and Family Composition, Other Situational Variables, and Time Allocation," Unpublished Master’s Dissertation, Cornell University, 1972

2. The same kind of logic applied to the more advanced levels of training in Computer Science. While the percentage of women among recipients of Bachelor’s degrees in the 1980’s was commensurate with the percentage of women programmers, the percentage for Master’s degrees was somewhat lower, and that for Doctorates was much lower, even when the foreign born had been separated out.

Karen A. Frenkel, in "Women and Computing," (Communications of the ACM, Nov. 1990 [vol. 33, num. 11], pp. 34-36) gives figures for proportions of degrees in Computer Science granted to women in 1980 and 1986-89 (p. 38). The figure for bachelor’s degrees fluctuated from 30% to 35%; the figure for masters was 20.9% in 1980, and fluctuated between 27% and 30% later in the decade; the figure for doctorates fluctuated between 9% and 18%; and that for doctorates awarded to Americans fluctuated between 12% to 21%. Finer distinctions over such a short time scale are probably not significant, especially given the small number of doctoral recipients.

However, these figured do not necessarily represent simple attrition over additional years of study. The recipients of graduate degrees in computer science did not always take their undergraduate degrees in that subject. An undetermined number of engineers (typically from "old-line" fields like mechanical, civil, or chemical engineering) may have chosen to "retread" themselves with masters’s degrees in Computer Science.

did by some accident think about it would discard it as meaningless.

Engineers, by contrast, must think about the process whereby their designs are turned into artifacts. Large numbers of engineers are employed in supervising the manufacturing process. There is, in fact, a whole branch of engineering, Industrial Engineering, concerned with this supervision, but the manufacturing process tends to infiltrate other fields as well, as they deposit certain of their members in the factory with directions to see that what is manufactured is what was designed, and that what is designed can be manufactured.

This can be seen in the clearest way in the oldest branch of engineering, Civil Engineering. Large numbers of civil engineers are employed on construction sites. Others are employed in survey work, for much of the complexity of civil engineering is the complexity of the earth’s surface. This surveying shades over into construction by way of such activities as test boring and drilling. Thus civil engineering is integrally linked to the work of construction. A civil engineer makes his way in the world not by drawing on paper, but by going out into the field. The most ambitious civil engineers make their way not merely into the field, but into the most remote wildernesses of the most backwards countries, where none have gone before.[1]

The civil engineer in the field finds himself in the center of a group of men, such as construction workers, miners, etc., who are employed in intrinsically strenuous and hazardous occupations. If civil engineering is an overwhelmingly male occupation, the surrounding occupations are still more so. In 1993, 9.4% of civil engineers were women, but that was five times the percentage of women construction workers or women miners.[2]

The conditions of 'roughneck' work define an ethos of their own. The contractual model of society, which underpins much feminist thinking, simply does not apply under such conditions. Small numbers of women have penetrated construction work, but they have not done so on a conventionally feminist basis. They have done so by accommodating themselves to roughneck culture.[3]

Summing up, engineering is largely closed to women because its internal structure and imperatives require it to demand a measure of commitment which most women, with other life goals, are not

---

1. For example, see Richard L. Meehan, Getting Sued and Other Tales of Engineering Life, The MIT Press, Cambridge, Mass, 1981, pp. 89-148. for an account of how one young civil engineer, in 1963, quit a laboratory job testing soil samples, in order to go to upcountry Thailand to do onsite soil testing for a dam that was being built with American AID funds. Once there, he happily went more or less native, and immersed himself in a world of peasant villages, construction by elephant, and, as a spare-time recreation, hunting in the jungle.

2. 9.4% of Civil Engineers in 1993, versus 1.9% for Construction Trades, 1.8% for Extractive Occupations, per: Statistical Abstract of the United States, 114th Edition, 1994, U. S. Department of Commerce, Table 637, "Employed Civilians, By Occupation, Sex, Race, and Hispanic Origin," pp. 407-09

3. [Link in the blue collar women book]

prepared to make.

If engineering was closed to all but the dedicated few, computer programming was open. If engineering educators were continually attempting to make prospective engineers into eighteen-year-old graduate students, the developers of programming tools were continually attempting to diminish the skills required to program a computer.

Unlike engineering, computer programming did not place extravagant demands on its practitioners’ ability to cope with complexity, and what demands there were, grew less with time. The practitioners’ surplus ability therefore went into coping with the complexity of the world outside computer programming. This was the work of systems analysis: reducing the disorderly real world to programmable terms.

In opposition to the school culture of computer science, there existed, almost from the beginning, a shop culture of computerization. Computerization was in essence the act of going out and collecting complexity from the real world in order to reduce it to programs. For most practical purposes, this complexity existed in a work organization. Part of the complexity would be in the form of poorly cataloged paper files, procedure manuals, etc., but much of it would be in the form of knowledge possessed by workers, and even collective agreements and understandings between workers.

It is effectively impossible to capture an accurate and sufficient description of a traditional work process without the consent and assistance of the people who presently run it. If computerization does not have their blessing, they can easily sabotage it by selectively providing misinformation, often in so subtle a form that they cannot be said to have lied. They merely chose not to take any particular pains to ensure that the systems analyst understood them correctly.

Thus, in its highest form, systems analysis necessarily ventured into the sociological realm. In contrast to the arid economics and decision theory of even an enlightened school culture representative such as Herbert Simon, systems analysis had to deal with the actual experiences of particular social classes. Michael Rose, in a representative handbook on computerization,[1] deals with such issues as the way of life and diminishing social status of clerks, the probable effect of the computer upon promotion ladders, the company politics of computerization, etc.

The distinction between school culture and shop culture is not absolute. Between computer science and total reliance on practical experience lay the Information Science curricula of business schools. The Information Science curriculum merged computer programming with a thorough grounding in all the major elements of business administration.[2]

The personal qualities required to collect complexity differ

---

[1] Michael Rose, <u>Computers, Managers, and Society</u>, Penguin Books, Ltd., Harmondsworth, Middlesex, England, 1969

[2] For example, see J. Daniel Couger's draft curriculum of 1968, apparently produced in response to the Association for Computing Machinery's "Curriculum '68." Couger's proposal was published as "Business DP Degree Programs: A Deficiency," <u>DATAMATION</u>, July, 1968, pp. 49-51.

from those required by the more purely technical sorts of work within Computer Science proper. There is little need for the more advanced types of systems programming, and there is rather more need for an ability to cope with human complexities. Systems analysts rise and fall, ultimately, not on the basis of their capacity to perform feats of logic, but rather on the basis of their ability to establish rapport with the people whose real-world knowledge they must employ. In this context, the very strengths of the engineer (and other technical virtuosi such as hackers) became liabilities. Young men of superlative technical training are often absurdly arrogant about nontechnical matters, or even about technical fields other than their own.

Women brought to computerization qualities which filled the gap left by the male arch-technician. They were more likely to be broadly educated than men, if not so deeply. Even granted that the stereotypical sorority girl did not work very hard in introductory sociology, she was at least there; she did attend the lectures, and did remember some of their content. That was considerably more than could be said for her male engineering-student contemporary, who never found the time to sign up for sociology at all. Or, if forced to take sociology, and forced to actually attend, the engineering student, on the rebound from an ’all-nighter’ in his chosen discipline, still managed to make up a considerable part of his lost sleep. So women were predisposed to think more broadly than men.

The whole art of conquering complexity which computer programming offered generated a new kind of technician. This new technician was not an engineer, but almost an anti-engineer. Unlike engineers, computer programmers were not forced into a definite mold by the requirements of their technology. They were free to be versatile. Computer programmers were not compelled to turn inwards on the esoterica of their craft as engineers did. They could look outward, using computer programming as a means to reinterpret the external world. The style of computer programming was lighter than that of engineering, with less premium on sheer tenacity of the bulldog variety, and more emphasis on receptiveness and imagination. Computer programming drew on all kinds of abilities traditionally cultivated by women, and discounted the combative tendencies which boys learned on the football field or in the boxing ring. By making feminine characteristics into virtues instead of vices, programming evolved as a technical field uniquely receptive to women.