PROGRAMMING--THE QUIET EVOLUTION

G. D. Brown

July 1970

P-4417

12

PROGRAMMING--THE QUIET EVOLUTION

G. D. Brown*

The RAND Corporation, Santa Monica, California

The age of computer hardware is easily measured in generations; the first characterized by vacuum tubes, the second by transistors, and the third by integrated circuits. Computer software does not advance in generations in which new developments make old ones obsolete. Programming has progressed from machine language to general-purpose high-level languages to specialized, problem-oriented languages without discarding anything. What could not be converted for a new computer was emulated.

Many changes are discernible in the still-young programming profession, but the changes have had little to do with the revolution in hardware. Programming is a very human field and its changes derive more from human needs than technological possibilities. A look at this fascinating profession is in order.

THE MOOD

Programming in the 1960s was influenced perhaps as much by the mood of the times as by the newness of the profession. The nation's economy sustained an unprecedented period of growth with no serious reverses, and we began to think and act as if affluent. The computing industry and the programming profession not only shared, but greatly exceeded this growth.

---

*Any views expressed in this Paper are those of the author. They should not be interpreted as reflecting the views of The RAND Corporation or the official opinion or policy of any of its governmental or private research sponsors. Papers are reproduced by The RAND Corporation as a courtesy to members of its staff.

The mood of the 1970s contrasts sharply with that of the 1960s. We have suddenly awakened from the euphoria of the 1960s to find ourselves facing a harsh reality. The world we knew and the values we regarded have changed.

Computer security at one time meant protecting our information from belligerent nations. It now means protecting our physical machines and data from the excesses of society. We worry about the destruction by our own citizens and less about foreign enemies. This has reduced the visibility of computers. A computer displayed prominently in an office window is no longer a status symbol; it is an insurance problem.

Many companies, particularly in the aerospace and defense industries, are experiencing sharp budget cuts, and now the gap between programmers and jobs appears narrow. Programmers and managers are learning the true meaning of cost effectiveness. Managers suddenly discover that they are free to assign programmers to tasks suited to their skills rather than to their tastes. Undoubtedly the economy will turnaround making the budget constraints less severe, but we will remember them.

The charisma of the aerospace and defense industries has faded, not because of the budget cuts, but because of the Asian war. In the early 1960s advertisements for programmers abounded in pictures of missiles, war planes, and other accouterments of destruction. Today we would find such advertisements repelling. Everyone has been disillusioned by the war; some because they believe our country to be in the wrong; others because our mighty technology and resources have proved unequal to an army of peasants--none of whom have college degrees. We can all agree that national defense is more palatable when it deters Russia than when it invades Cambodia.

The space program reached its goal of the moon, and further developments will for some time seem anticlimatic.

Past successes must now be consolidated. The taxpayer, gazing up through the smoggy air, is beginning to question whether the money might better be spent elsewhere. And so the space program will decline in popularity. In fairness it should be noted that the public, unenthusiastic about Columbus, was even less impressed by Leif Ericsson. But then he had a problem common to many programmers--he didn't document well and so his discoveries were of little value.

Research too is losing prestige. Like all professions, research is interested in perpetuating itself. Thus the stereotype research project, rather than recommending a course of action, concludes by suggesting that more research is needed.

And now ecology is making past successes look like long-term disasters. This is unsettling because we are brought up to believe that progress is sacred and yet we find that progress pollutes. We know that nature deals harshly with animals that foul their own nests. Have research and science somehow failed us? The question beginning to form is why spend all that money on research to come up with short-range benefits that lead to long-range ruin?

Research, with the public as a patron, has been a field of glamor and money--so much so that Ph.Ds are rapidly becoming a glut on the market. The researcher, shrugging off distractions to concentrate on just his thing, groused that the public did not recognize his good ideas and pay to implement them, disdaining to take the time and effort to explain why his ideas were good. The public may soon be less inclined to pay researchers just for good ideas. The researcher may not retreat to his position in the Middle Ages when he was a penniless ascetic in a long, shabby coat bending over his formulas and calculations in a dusty attic, but he may change his image.

THE FEARS

We entered the 1960s with the disturbing thought that computers might someday be able to "think." After all, if computers could prove theorems in plane geometry, play chess, compose music, and solve calculus problems, why could they not be said to think. Granted the computer could not write music like Mozart, but how many people can? This worry has since disappeared. We began to realize that it is easy for a computer to solve calculus problems just because it is a computer, and we are impressed only because it is hard for us. We soon lost interest in some of the computer's exotic capabilities because the market was already flooded with people who could write poor music and play amateur games of chess. Finally, we discovered many things that humans can do very easily, such as recognizing a face in a crowd, that we cannot conceive of a computer doing. Man enters the 1970s secure in his position as an intelligent machine.

Slowly we began to see a growing menace from an entirely different direction. An old man is suddenly told that he has a bad credit rating, but he is unable to find out why and get it changed. A housewife receives an incorrect computerized bill and is unable to get it corrected. An engineer is abruptly dismissed from his job because a computerized dossier shows his name on the attendance list of a tea sponsored by a group once considered subversive. Computers can store and manipulate vast amounts of information, but it seems as if once the computer begins to process its information, events move inexorably forward to their conclusion, justly or unjustly, without the individual being able to change them. There is no court of last resort and little human intervention to protect us from a computer that knows neither compassion nor mercy.

Little wonder we felt uneasy about completing our census reports. The computer's long memory allows our actions of years ago to remain with us always; information about an individual has no statute of limitations. The abuses of information we have seen so far have been relatively minor, with no harm intended. Will this always be so? Each of us is vulnerable to the misuse of the records we create in our daily lives. As an admittedly extreme example, how many marriages would survive if each wife was mailed an itemized accounting of all her husband's expenditures, including flowers, lunches, cocktails, and motel rooms? Would not some of our past be better forgotten?

The information revolution may have less impact on individuals than the industrial revolution, but the solution of some of the affected has been the same; machines were smashed. Programmers, inclined to treat the computer as a toy, are saddened to think of how its misuse can ruin lives. Even more sobering is the responsibility they have suddenly had thrust upon them.

THE PEOPLE

Programming in the 1960s was a field of intense excitement. New people with varied backgrounds entered this informal, unstructured field with no training and scarcely any idea of what programming was. Many had never seen a computer before becoming programmers, and they stumbled into the field only by happy chance. Almost everything they did had never been done before, and it was always a success. If it turned out badly, they simply had to call it research. Jobs were plentiful, and the field was easy to enter because no experience was necessary. Good programmers could dictate the problems they worked on, and for them the field was almost as much entertainment as it was occupation. The emphasis was largely on inventiveness, cleverness, and exotic

solutions to problems. Programmers seldom paused to worry if they had the right problem. Small wonder that compilers were built for languages that no one wanted to use. It was the ideal time to be a programmer.

Programming, whether because of the newness of the field, the nature of the work, or to meet the demands, attracted many unconventional people. Programmers tended to be an eccentric lot. Not that they were the first hippies, but they did not look like accountants. Programming was also a convenient place for the misfit college graduate who either couldn't get a job in his major field, or who was disenchanted with the field he happened to choose. Programmers had such diverse backgrounds as music, mathematics, engineering, geology, english, and some had no formal training at all. These people with their varied backgrounds contributed much to the computing profession in new, fresh ideas. They made programmers fun people to be around. In the future, this type of person will have a harder time breaking into programming because the positions will be taken by graduates with degrees in computer science. Programming will become more conservative and traditional.

Many programmers suffered a great shock in the mid-1960s with the transition from second to third generation computers. It was catastrophic for the machine-language programmer; much of his skill and knowledge, learned over a period of years, became valueless. Equally apalling was the realization that programs that had taken years to develop would never run again. How sad to see the work of years suddenly disappear overnight! Much of the loss of the computer's mystique can be traced to the conversion from second to third generation. Upper management became very disenchanted as time losses and conversion costs disrupted the entire company.

Computer programming is a completely man-made science with no natural laws. The computer's repertoire of instructions correspond to natural laws, but when the computer

changes, the entire science and body of acquired knowledge even down to the vocabulary changes also. There is some compensation in the new opportunity to learn, and the security of having to rewrite some old programs, but few programmers find the latter very stimulating. The experience has made programmers feel insecure in their knowledge, but very secure in their occupation. It has also made programmers value compatibility--known as tradition and stability in other professions. As a gauge to how much programmers have changed, the 'reset to zero' was called a technological innovation in the past decade, but it would be called planned obsolescence now.

THE APPEARANCE

The public, generally in awe of computers and the men who instruct them, had little notion of what computing was all about in the past. But computing as a glamor profession is now giving way to ecology. As computers become better known, they begin to lose their mystique. We notice how unimpressed students at the University of Illinois are with the new ILLIAC IV super-computer.

The public used to forgive programmers their errors because they thought programming such a dark science that they were amazed it could be done at all. In the future they will be held accountable for their actions. The recent California elections illustrate this. We witnessed the sad spectacle of voters waiting for ballots to be counted while a programmer finished his debugging. There was much wrong with the computerized balloting in California besides the programming, but the proceedings did little for programming prestige.

The public familiarity with computers and the consequent loss of prestige will force other changes. In the 1960s people took pride in computer-generated reports. "Look, this

was done on a computer. In the 1970s they will likely say "Look at this ugly printing!" Perhaps we shall one day see upper- and lower-case printers and keypunches, and even standard 8½ × 11 computer output. We will certainly see more human engineering.

The programmer's image will probably change as he enters new fields of responsibility. The public is very careful about the type of person that it allows to handle its money, process its tax and credit reports, and count its ballots. Perhaps programming will begin to look much like the accounting profession, complete with formal auditing procedures. The sad loss in this will be the beards and sandals.

THE NEEDS

Programming, like the accounting profession, has a strong requirement for quality. It is not too far wrong to say that if a program is not perfect, it will not run. And yet programmers have not been overly concerned with quality. Perhaps this is because our society is more attuned to quantity than to quality. This is immediately impressed upon anyone who has purchased a new car. Quantity is made a virtue even when it is unwarranted. Thus a cigarette proudly asserts the inconvenience of its longer length, and a potato chip is touted as the world's noisiest. Americans leaving the Louvre always seem vaguely disturbed. "I thought the Mona Lisa would be--well, bigger."

Our preoccupation with quantity can be explained by our history as an expanding country of vast size, but that is changing. We find that quantity pollutes and we must control it. "Nothing in excess" as the ancient Greeks said. And so our vast energies must turn slowly away from sheer quantity and direct their attention to quality. This is particularly true of programming because it is such an exact science where errors in credit ratings, counting ballots, and billing credit cards are highly visible and

annoying to the public. However, the transition will be slow and hard for everyone.

The coming emphasis will also be on producing a finished product. It is perhaps a universal constant that the last 20 percent of the apparent job takes 80 percent of the time and effort. We have all noticed buildings seemingly spring up overnight only to remain empty for a year while the finishing touches are applied. A traveler bound from Chicago to Manhattan knows that when the 747 begins circling Kennedy Airport, the grueling part of the trip has just begun. In the past, when a programmer said that he had the job all programmed with just a few more bugs to correct, the job was considered all but done and passed on to some unfortunate trainee. Now we know that only the easy part is done. The final checkout, documentation, and production engineering will take 80 percent of the total time and effort.

Programmers will take the extra time needed to give polish to a finished product. Cleaning up a system will be a necessary part of programming. Programmers will take pride and even aesthetic satisfaction in doing a competent job through to completion. Greater emphasis will be placed on production engineering computer jobs to adapt them to their environment and users. We already see such tools as IBM's System/360 Job Control Language; a complete new language for facilitating program execution. Programmers will spend much less time coding algorithms, and more time maintaining existing programs.

THE FUTURE

The action in programming during the 1960s was definitely in the scientific fields; aerospace, national defense, and research. Now there seems to be a marked drift toward commercial programming. One sees good scientific programmers migrating to the commercial world. This is partly due to its greater security, but perhaps it is also

because the commercial world presents a new challenge. As has been pointed out, solving calculus problems is easy with a computer. Conversely, report writing is hard, file management is hard, data validation is hard, documentation is hard, program maintenance is hard, and working in profit-oriented industry that pays strict attention to costs is hard.

Past predictions about computers generally glossed over software with a sentence or two to the effect that new computer languages would soon arise to solve the programming bottleneck and make programming simple for everyone. This chimera is still pursued, but programming remains expensive and time consuming. The tools developed a decade ago are still the mainstays, and debugging and documentation aids are as primitive as ever. Programs are so personalized that there is little value to a half-finished program without the programmer who wrote it.

Programming, rather than becoming easier, is getting harder. Computer software always trades power for convenience, flexibility for simplicity, and capability for compatibility. Old programming languages never die, and so a new language complicates programming by giving yet another language to learn. Computer programs have an aesthetic impact that engenders requests for changes from even the most unlikely people, but little has been done to make program modification and maintenance easier.

For the forseeable future, programming will remain exacting and require skill, patience, and training. As programming grows more complex and as it loses its novelty, fewer non-programmers will be tempted to try their hand at it. No economies of scale are yet discernable in programming, and so the general rule remains that the fewer programmers working on a task, the better its chances for success. Mass production of computer programs remains as far away as ever.

The programming profession is now developing a much more critical eye. Some now find it mildly humorous that ALGOL could be designed without any input/output. By the end of this decade, it may become a computer cliché similar to building the boat in the basement. Suggestions for developing new programming languages are beginning to be prefaced with apologies and met with dismay. We are even beginning to look critically at the industry's glamor children, time-sharing and computer graphics.

We clearly see the dangers of micro-programming and languages for writing operating systems, the two latest programming rages. Programmers will consider micro-programming a license to develop their own personal computer. If languages for writing operating systems make them easy to create, then no doubt we shall have them by the hundreds. Thoughts of long-range compatibility, already staggering from too many programming languages, would be bleak indeed.

THE CONCLUSIONS

The programming profession will become more stable and traditional. Programmers will have more formal training and will mature to become more thorough with a pride in competence and a preoccupation with quality. Some old-time programmers, finding that they no longer fit in this changed profession, will drift on to other fields.