V

The Personal Computer Phase, 1980--

Personal Computers transformed programming.
[put in topic para here]

About 1980, the personal computer recovered from its initial dark age. By this time, personal computers were available with sixty-four killobytes of memory-- enough for a journal article-- and, more important, two floppy disks, each with a capacity of a hundred kilobytes or more. This capacity was not, as such, enough for all uses, of course. However, many, and probably most, large programs could be decomposed into a series of self-contained phases or passes, each of which could run on such a machine. The size and complexity of the program which could be run was limited only by the user's willingness to repeatedly change floppy disks.

Five years later, at a time when the IBM PC AT and the Apple McIntosh had been introduced, there were floppy disks in widespread use, which held a whole megabyte, and small "winchester"-type hard disks of five megabytes or more. The Apple McIntosh's Motorolla 68000 processor was not quite comparable to an IBM 370, but it was gaining rapidly. At this stage the personal computer ceased to be the mainframe computer's poor relation. An increasing range of personal computer programs, computer language software included, were not inferior to their mainframe equivallents, but rather, superior. The most admired

language of the mainframe era had been IBM's PL/I. IBM's mature PL/I Optimizing compiler, with its libraries, had taken up about two and a half megabytes of disk space, [1]well within the capacity of a hard-drive-equipped personal computer. PL/I was not ported to the personal computer for years, and even then, the price tag was ridiculously high, in excess of $10,000. However, other programming languages, notably "C" and Pascal, filled the vacuum. Language software was available for $500 at first, and then, as the market saturated, some brands were available for $50, and finally, some public-domain languages were available for the cost of copying floppy disks. Within a few years after the introduction of the personal computer, much of its system software (operating systems and programming languages) was back within the tradition of the mainframe.

Put another way, the technique of creating software was substantially back within the mainframe tradition. The methods of managing and controlling complexity were once again valid. However, the programming was not the same. Personal computers were smaller than mainframe computers ever had been, and they

---

1. That is, 202 "tracks" on an IBM 3330 disk drive, with each track capable of holding up to more than 13,000 bytes, depending on the care and skill with which the data is packed. A track is the basic unit in which a mainframe's disk storage is allocated, the cybernetic equivallent of a railroad boxcar. We may take it as read that the IBM'ers did an efficient job of packing the compiler into as few tracks as possible.

IBM Corporation, OS PL/I Optimizing Compiler: General Information (# GC33-0001-5), 6th ed., September 1984, San Jose, California, p. 33

Spotswood D. Stoddard, Principles of Assembler Language Programming for the IBM 370, McGraw-Hill Book Company, New York, 1985, p. 517

were cheaper, of course. Personal computers were far more effectually standardized than mainframe computers had been, and it was comparatively feasible to program them in the aggregate, that is, to write a program to be run on an unknown computer with the confidence that the program would run on a strange machine without needing modification. Personal computers required less skill to operate-- unlike mainframe computers, personal computers customarily had their startup procedure built into a Read-Only Memory. Soon there were systems of automatic secret handshakes which made it a comparatively simple matter to connect up additional components. A recent standardization initiative is called Plug and Play. By 1960's standards, plug and play had been attained with the IBM PC of 1981. New types of programming systems, such as spreadsheets and databases, had deskilled many routine types of programming to the point that this programming could be done by nonprogrammers. Summing up, there was far less routine programming to do.

However, this worked both ways. There were immensely more personal computers than there had ever been mainframes. Being more adaptable, personal computers were employed for more diverse tasks than mainframes had been.

The new roles for programmers were characteristically skilled ones. They were skilled either in the sense of doing programming at a very high level, or in the sense of teaching programming and computer usage. It is difficult to determine the respective proportions of the two types-- census data does not make the

distinction, and in any case, many individuals probably were a bit of both. At any rate, the category of "Computer Systems Analysts and Scientists" went from 276,000 in 1983 to 769,000 in 1993, a nearly threefold increase, and in the process, became the single largest category of programmers.

The new highly skilled programmers were engaged in writing application programs to solve classes of problems, rather than the individual problems that earlier application programs had solved.

A small minority of programmers were engaged in developing really large programs to run on the personal computer, programs intended for general publication, such as word processors.

Other programmers of the same general type did hardware-related programming. Personal computers were much more likely to be connected up to all kinds of specialized electronic devices than mainframes. They belonged to individuals, and these individuals were free to simply install devices in their machines, in a way which could never have been permitted with large mainframes used by many different people. For each such device, there needed to be software. Sometimes there was a whole program to run with the device, and sometimes there was just a "device driver" to translate between the electronic device and some fairly standard program. Either way, the writing of such programs was a refuge for the most intricacy-loving of perfectionists. There were never very many of either of these two kinds of virtuosi. They tended, however, to enjoy personal prestige out of all proportion to their numbers.

Considerably more programmers were engaged in doing quite skilled programming to create connections between the world of mainframe software and datasets and the personal computer. This programming included modifying mainframe programs to run on personal computers; modifying mainframe programs to look like the programs which ran on personal computers; and writing programs which permitted big and little computers to talk to eachother.

One immediate task was modifying the existing inventory of mainframe software. A lot of mainframe programs no longer belonged on a mainframe at all, now that personal computers were available. The most basic litmus test was whether a program directly involved the sharing of information between two or more users. If it did not, then, personal computers were generally cheaper. In the first place, a computer terminal contained most of the components of a personal computer, and some components which a personal computer did not nesssarily need (for example, a modem). So terminals were not especially cheap. By the time the cost of a telephone connection with the big computer was taken into account, the supposed economies of scale of mainframes looked pretty hollow.

However, personal computers did not conform to the same technical standards as mainframe computers. To make a mainframe program available for use on personal computers, the program had to be translated, a more or less laborious process known as "porting."

Even if a program was staying on the mainframe, its users would be using personal computers on other occassions, and their

expectations would be raised. Personal computers were customarily much more "user-friendly" than mainframes. The first personal computer software developers had developed quite new ideas about what a computer's screen was supposed to look like, drawing on such unlikely design sources as video games and soft-drink vending machines. Users expected to push keys like the buttons on a vending machine, instead of entering command words, and they expected a screen with assorted status lights, counters, etc., and continuously displayed lists of options. Mainframe programs had to be revamped in order to catch up.

The same thing applied to programs which were being ported to the personal computer. Even if such programs had not originally qualified as systems programming, they would do so by the time they had an acceptable user interface.

Then there was what one might call bridging software to be written.[1] This software would exist in two or more parts. One part would run on the mainframe computer, and another part would run on a personal computer, and the two parts would talk to eachother. Thus only the specific operations which required data sharing would be done on the comparatively expensive mainframe computer.

All of this programming was, for the time being, highly skilled work, of the variety that had traditionally been called systems programming. The term "systems programming" was itself

---

1. An example would be what came to be termed "Client-Server" software, but not all bridging software was necessarily proper Client-Server.

falling out of use, because it no longer made a useful distinction. People who did the kind of work which might previously been called systems programming began to call themselves by other terms such as "software developer." But there was another kind of programming work emerging. This was the job of helping nonprogrammers to use computers and even to program them.

A new kind of computer professional emerged to deal with personal computers, or more precisely, with their partially skilled users. This was the "consultant," a jack-- or jill-- of all trades. In a sense, a consultant was a new twist upon the computer service bureaus, which had long provided comprehensive service to end users. But the consultant operated at the individual level rather than that of the company. Consultants helped individual computer users with their computers. A consultant would do a whole range of tasks, such as buying the customer a suitable computer and software, installing and setting up everything, teaching the customer to use the system, preparing short manuals, and even doing a certain amount of simple programming, generally in the script languages associated with mainline application programs, rather than in a recognized programming language. One mark of the consultant was his or her toolkit, a small leather zip-fastened wallet with perhaps a dozen tools required to take a personal computer apart and install accessories.

//see ditlea, 6/15/85, p.84

2. Comparatively isolated work.

Most programming work was now made up of relatively fast jobs. Software development was still by no means a small job, but its productivity had been increased considerably. The porting of programs, like any other form of translation, was a rather faster job than the original writing, especially since there were oftenprograms to do part of the translation. But probably the most important influence was that programmers were now not only producing personal computer software, but using personal computers to do it. Personal computers were not that powerful in the abstract, but they were cheap, and a programmer's personal computer was likely to be much more powerful than his or her proportionate share of a big computer.

Given all this computing power, it was possible to make the computer do still more of the work of programming. This advantage started even while the program was still being drafted. Even the least impressive word processors available on a personal computer were infinitely superior to to the crude line editors, such as On-Line Business Systems' WLYBUR,[1] which were commonly used with a mainframe computer's terminal system. Even to change or delete a single character with a line editor involved an elaborate rigamarole, as did saving a file. And the line editor was itself a vast improvement on the keypunch, which was practically comparable to a linotype in its general awkwardness.

Once the program was drafted, it was fed into the language translator, either a compiler or an interpreter. The compilers

---

1. See OBS WYLBUR User Guide, On-Line Business Systems, Inc., San Francisco, California, 6th ed., April 1980

used on personal computers were not much better than those on mainframes (and often not as good), but they were vastly more available. There was no waiting queue to use one's own compiler, as there often was on the mainframe. If one wanted a printed listing, it came off one's own printer-- immediately; instead of being printed off in its proper turn, by a giant central printer, sorted out at length by clerks, and placed in a pidgeonhole for one to retrieve and carry back to one's terminal. With a personal computer, it was possible to compile early and often, letting the computer find the errors instead of laboriously looking for them oneself.

Once the program passed the compiler, it was gramatically correct, but that was not to say that its meaning was what the programmer intended. Ultimately, the only way to find out was to run the program, and see if it did what it was supposed to. This remains the most difficult and devious part of debugging a program. Under the old mainframe regime, it was especially so. To determine what a program was doing, the programmmer had to insert additional instructions causing it to print out messages, and from these messages, the programmer would have to infer what was happening in the program's innards where he could not see. One set of additional instructions would probably not suffice, so they would have to be removed-- hopefully without inadvertantly altering the program proper-- and others substituted in their place. Well, goodbye to all that! With the luxuriance of means provided by the personal computer came the interactive source-level debugger. This program was a kind of cybernetic X-ray

machine. It could look at every intimate detail of a program, even as the program was running, and could stop the program at any indicated place and restart it again. With a source-level debuger, it was the easiest thing in the world to find an elusive and improbable error which only manifested itself after a million program steps.

Of course, in a few cases, this increased productivity went to support giant programming projects, requiring hundreds of man-years, but those were rare. More typically, what had been a project for several people became a project for one or two people.

// insert the bit about datakulture here

If tasks were comparatively small in software development, they were absolutely small in consulting.

less likely to work with other programmers, who knew their skills, and continually tested them in largely noneconomic competition, friendly or otherwise.

b. Most programmer's jobs now tended to offer greater autonomy, but also less of the protection of the group.

c. Programmers were now surrounded by nonprogrammers, who could judge them only on external qualities.

d. A much higher proportion of ordinary programming jobs were now managerial in substance.

---

3. Women programmers became sensitive to the imagery of competence or incompetence, because this imagery determined their effectiveness.

a. If a woman was trying to teach a bunch of middle managers how to create spreadsheets, and they wouldn't listen to her because she was 'a dumb broad,' them she was ineffective, no matter how much she knew. By herself, she could not possibly gather and collate all the information required to create all the spreadsheets the organization needed. She could only train and induce her middle-aged male students to do so, and if they chose to take refuge in dumb insolence, there was very little she could do about it.

b. if she could somehow surround herself with an aura of the conventionally macho (for example, by climbing mountains), she might maneuver men into accepting her as one of themselves. Hence the advertising imagery of amazonism.