The Evolution of Electronic Computer Services

By Robert L. Patrick

Written: June, 2016

CHM Reference number: R0365.2017

Author Sketch: Coder, programmer, software architect, facility manager, company founder, computer specialist (consultant)

Abstract: A trip though time (1950 to date) discussing how the demand for computing services influenced the services offered.

Focus

The ancients could read, compute, and record. If you are interested in early one-off computing efforts look elsewhere, because this piece is about computing in bulk and the several contusions some of us have lived through.

Externally Programmed Computing

Machine accounting comes first because some accounting applications have not changed much in the last 100 years. In Tiny Tim's day, accounting was done manually with bound books and ink. Then along came Watson with the punched card. IBM developed electro-mechanical machines that could:

a. Sequence those cards based on data punched in them,
b. Accumulate sums from cards passing under a read head,
c. Selectively print data from cards and the accumulating counters onto the pin-feed tab paper that is still in use today.

Prior to 1949 engineers designed on paper and wrote equations. Some of the equations were evaluated by slide rule or mechanical calculator, but mostly they bent metal to test their designs.

In 1948 an engineering team at Northrop wired several accounting machines together (one of which had tubes and could do calculations at 100 operations per minute) and had the first digital configuration that could automatically do an arbitrary sequence of calculations and print the result. Custom wired general-purpose plugboards were the predecessors to our software since they took the capabilities of the bare hardware and made it useful to a user population.

Their equivalent of an application program was a deck of punched cards which commanded the configuration to read data from cards, perform a custom series of calculations, and print the results.

The detailed design of the plug board wiring was ingenious. I copied and tested a set of three plug boards for the IBM Card Programmed Calculator (CPC) at the Bureau of Standards outpost (SWAC) on the UCLA campus in 1951 and then installed the boards on a new CPC at Edwards AFB to do flight test data reduction.

After the system had been set up and tested, I became an applications programmer. Leading edge engineering outfits had used analog computers for dynamic evaluations and prepared spreadsheets for hand calculation where numerical accuracy was required.

Every program had to be tested before being repetitively used for production calculations. Two ladies, skilled in the use of mechanical desk calculators, would be assigned, using input data prepared by an engineer, to calculate their way one or more times through the spread sheet. These ladies, respectively called computresses, would compare results after each step of the hand calculation to catch errors and deliver a perfect test case. The CPC programs had to produce the same results from the same input to be accepted.

IBM made a pair of compatible machines to accurately prepare punched cards. A lady used a Keypunch to convert handwritten input to properly perforated cards. Then a second lady repeated the operation using a Verifier that checked that the holes in the cards matched the key on the keyboard that had just been depressed. Transcription errors were rare in a deck of cards that had been professionally keypunched and verified.

The CPC did mostly linear calculations. It used polynomial forms in lieu of iterations to evaluate some math functions since it did not iterate well. Very complex functions (like the profile of the atmosphere) were done using look up tables. The CPC had a table look up ability that allowed it to scan a prepared deck of cards that held a table, select the proper value, and skip the rest. The table of pre-punched cards had to be inserted in the deck of cards before the CPC started a calc run. Very complex calculations required two passes through a CPC. In the first pass the computer produced intermediate results on punched cards. In a second pass results were printed out. All this occurred at 100 cards per minute.

Computer time was precious. Shortly after installation, a second shift was set up with written operator instructions and professional operators. The machine was not very reliable. Original test cases were saved and used to determine if the machine operated properly. An operator was always in attendance. On frequently run jobs, the operator could tell from the pattern of sounds whether everything was normal or something was amiss.

Production was the goal. The whole airbase existed to verify each manufacturer's new airplane designs. Design fixes and production contracts were hanging in the balance. The defense of the country in some way depended on the completion of a flight test report. There were no games or experiments. This was a production shop.

In 1952, a nearby earthquake caused the casters on the printer (our heaviest unit) to punch through the floor of the old WWII barracks we were using as a machine room. Other than that we tested during the day, produced during the evening, and cleaned up every night.

Began with Engineering Calculations

In 1952 IBM produced an electronic computer known as the 701. Physically each installation required quiet power and an air conditioned room about half the size of a

basket ball court. It was a binary machine designed for engineering work. It had more storage and was much faster than a CPC. It lacked in reliability.

I moved from a military environment to an industrial one. Convair-Fort Worth had produced B-24s in a mile long hanger during WWII. Post-war they built the big B-36. When I got there they were designing the supersonic B-58 Hustler. I found a home in the engineering department.

Out of the woodwork came engineers with equations to evaluate, jobs to run in production, and an overwhelming development load. In industry, big mid-year budget demands are not welcome. Our management wanted machine services but their understanding of what was required was limited. We were short handed, had a shortage of machine time, and our engineers were demanding.

We also suffered from an old internal political problem. Historically anything with IBM's initials on it was "owned" by the accounting department. While they had a CPC, only operators from the accounting department were allowed to run it and engineers had access only through them. Yet here I was, I had hands-on experience and had set up a production service. When the engineering management ordered a 701 and asserted control, I became a favored employee.
Eventually, accounting even let me directly use their machines.

IBM built nineteen 701s. Serial number 1 was installed in NYC and used for customer indoctrination and training. Convair installed serial number 7. I was on the NY machine the first month it was installed. Our salesman (later an IBM-VP) accompanied me. John Backus and an IBM team had produced an interpretive software package which was so like the CPC system I had been using, that I hardly had to read the manual. It was called SpeedCode, and it was free (i.e. Included in the price).

It took a piece of binary iron and allowed it to read and store a program punched on a deck of cards, do decimal arithmetic, and store/access intermediate results on a magnetic drum and on removable magnetic tapes. Since the 701 was a stored program electronic computer, it could do iterations and complex branching.

SpeedCode was easy to program but since it was interpretive it was slow to run. The shop also had a several independently created assemblers, dumps, and a manual library of subroutines. Some came from IBM and some from other users. SHARE had not been born yet. There was no knowledgeable management or operational concept. Each programmer used the soft tools he was familiar with and the engineering customers pressed for fast results to support design and production of the prototype B-58.

Had we known then what we know now, we should have organized to funnel all jobs initially through SpeedCode and get early results back to the engineer quickly, albeit slow in production. Then for frequently running jobs, recode them in machine language, from proven problem statements, to get efficient operation.

Instead some engineers had long waits (measured in weeks) for initial results. Further, machine room efficiency was lost since the assembly programs were used as loaders (due to the volume of changes required), and the programs were hard to maintain.

This freedom/chaos extended to machine operations. The style was "programmer present and operating". The machine room schedule was a signup sheet with 15 minute segments, and most all production was relegated to over night runs.

The programming staff consisted of a weak manager and 16 individuals. Some of us were engineers, a few had degrees in mathematics, and some had walked in off the street. We even had a female Marine veteran who kept us all in stitches at our frequent beer busts.

It was soon apparent that program development and production competed for the same (limited) machine time resources. As a result, some production standards and documentation were set up and a second shift (with extra rental dollars to IBM) was set up. The programming staff did not resist this standardization too much because it relieved each programmer of his production responsibilities (remember: programmer present and operating) and almost relieved him/her of late night phone calls when a production job crashed. I set up second shift operations and worked the first month until things settled down. The time was 1953.

The weak point in the 701 hardware was its Williams Tube (cathode ray tube) memory. The machine had a tendency to fail on long production runs. When we divorced the programmer from his creation, we had to add a new programming feature: RESTART. Without the operator being able to restart, a programmer who had successfully gotten a job into production would occasionally find a memory dump from a priority job on his desk when he/she came to work in the morning.

I tried a few experiments to get more efficient use out of our installed equipment: magnetic tape for program residence instead of punched cards, use patches through the card reader for updates to correct errors and keep the program current, use standard plug boards and standard card formats except for conversion of existing files, and get the programmers out of the machine room. These ideas I took with me when I moved to General Motors Research (GMR) in Detroit.

Machine Room Efficiency

When I arrived at GMR their research establishment had just moved into the new corporate research campus north of Detroit. The buildings were modern, with lots of glass, and connected by heated finished tunnels for coatless passage in the dead of winter. They had a CPC installed and a 701 on order. I was assigned to support a design team producing a gas turbine engine for the Fire Bird show car. I programmed engine design calculations and later reduced the data obtained from an engine test cell. As a 30 year old mechanical engineer I was in hog heaven.

When the 701 arrived (#17), we avoided the chaos I had experienced in Texas and picked one software package for each function. I continued my work with the CPC and SpeedCode on the 701. However, we still had programmers present and operating.

When the IBM 704 was ordered, we set up a project to prepare for its arrival and operation. For their big *commercial* accounts, IBM offered standalone machines to read cards and make magnetic tape, take tape and print it, and take tape and punch cards from the images. We ordered one of each. The similar devices which were directly attached to the 704 would be used only for maintenance and operator status information. We devised an operating system (perhaps the first anywhere) that allowed a stack of jobs to be loaded onto tape in a separate operation, the tape to be mounted on any unused tape drive while the previous batch was being completed, and the job stream started. Control cards imbedded in the stream controlled the decimal to binary conversion, the loading of the job for execution, monitoring the job while it ran, salvaging the job if it crashed, and formatting the output for a tape to print operation as the batch exited.

In addition, we had a locally constructed time of day clock and did job time accounting with end of job reports to the programmer telling what resources he used for this run. While much of this has been described in detail elsewhere, nothing has been written to disclose *why* we built the operating system way we did. (In 1955)

The 704 had a relatively small memory. Some of the conversion programs i.e. input and output translators, would have filled that memory and caused overlays on our typical sized jobs. If we loaded the translators once, and ran the job stream past them in a tape to tape operation, the total resources devoted to input-output were smaller. From that we had more mainframe time to use for productive applications.

Since the mainframe cost the most to rent, the more jobs we could push through it before encountering extra rental costs were important too. Thus efficiency was a watchword.

We did job time accounting and reporting so the programmer could know if his changes improved the operation or not. Further, we had the programmer estimate time required for all production runs. This was reported to the human operator when the job was loaded. This allowed the operator to recognize an infinite loop (runaway job) and provide meaningful information to the programmer when the job was purged for excessive run time.

By using control cards on the front of each job we could get a degree of uniformity on a mixed job stream that contained both checkout and production jobs and for jobs which used tape and those that executed only in memory. This uniformity allowed human operators to stay on top of the stream and salvage useful information on jobs in trouble. Further, substituting binary output for resident translators saved precious memory for larger applications.

The offline tape I-O allowed new input tapes to be hung while output tapes from the previous batch were being translated. If we had enough work queued up, the mainframe never stopped.

The concept allowed us to install a Fortran I compiler as just another input translator. This allowed us to offer compile and go in a single job. After I left the (software) project, they created simulators so the mainframe could simulate a piece of peripheral equipment that was down for maintenance.

For the short checkout jobs which occupied most of our daily schedule at GMR, we could run 600 test jobs in a 9 hour shift. During the operation I left at Convair, with scheduled 15 minute slices for programmer present operation, a full schedule resulted in only 36 jobs in a 9 hour shift.

The GM Tech Center occupied several buildings over a few acres. When the staff expanded to other buildings, the computer center provided desk-to-desk pickup and delivery: a mail person would pickup ready jobs, take them to the input hall, and later deliver the input deck and the output it produced back to the original programmer. This was in keeping with the concept that a programmer at the water cooler wasn't programming and might be keeping a colleague from his deskwork too.

The mature system provided a keypunch/verify service for new input (both jobs and data) to provide cards without transcription errors, a record of exactly what ran (the card deck), and up to 4 test shots a day if you could find and fix your programming errors fast enough. I had been assigned to a high priority missile job and became a user of the system I had architected. On one occasion late in our testing, I took four copies of our run deck, made minor modifications for conditions we needed to test, and ran four jobs in one batch. That level of test would be hard to do with another operating system concept.

The operating system *concept* we pioneered at GMR was used by some really smart guys who built the SHARE Operating System (SOS) for the 709, and by IBM in IBSYS for the 7090. After another GM assignment in the missile business, I left to run a commercial service bureau in Washington (CEIR) before I became one of the founders of the Computer Sciences Corporation (CSC). After a short while at CSC I started consulting (1959) and one early assignment I got back into software and computer operations with the Direct Couple project at Aerospace Corp. in El Segundo, CA.

<u>More Operational Efficiency</u>

IBM announced the 7040 (in 1961) and it was almost compatible with the 7090 Aerospace had installed. One of the Systems Programming team noticed that the cost of two 7090 channels was about the same as a 7040. (Smart people never accept gifts at face value.) He realized that with proper software the 7040 could handle all the I-O functions leaving the 7090 to do the heavy mathematics.

The Aerospace Corporation was the heart of the USAF Space Division and deeply involved with launches from Cape Canaveral. Their workload consisted of the jobs you'd expect from a bunch of engineers plus occasional demanding pre-launch calculations.

It was the pre-launch aspect that was the stinker. Here was a shop smoothly running several hundred batch jobs a day from central and remote sources and then a demand for 100% of the computing capacity had to be honored. To abruptly shut down all the job queues and dedicate the whole system to pre-launch was simple, picking up the pieces and restarting the big batch operation was the challenge.

Fortunately Aerospace had a competent crew of system programmers and an outstanding team leader. A project was considered to extend IBM's IBSYS to run on a Jr-Sr configuration. First, standard IBSYS was patched to gather some detailed job statistics. We measured non-overlapped I-O as this was the minimum amount of CPU cycles on the 7090 (later a 7094) that would be freed up by moving all the primary I-O to the 7040.

Then we established a rigid project goal: none of the changes to the operating system would be allowed to affect the applications programming community, i.e. if you had a job in production, the job would continue to run without change. After that we felt free to improve operations and efficiency. We plunged ahead with our system programming project and ordered a 7040 and a Channel to Channel attachment to connect the two machines.

The resulting system had several nice scheduling features. The queue of incoming jobs was completely handled by the Jr machine. New jobs were read and stored on disk. A list of available jobs and their descriptions was built. If tapes were needed they were called for on the tape vault terminal. The chief operator also had a terminal which allowed the computer to report status to him and allowed him to set key parameters for operation. The job queue was a table handled in priority order. The chief set the mode of operation. Normally, the computer would select the next job according to priority and time in the system. If that job required a tape that was still in the vault, a non-setup job would be run while the tape was obtained and mounted.

If a high priority job was entered, it was next. If an ultimate priority run was required, the mode was changed to complete or terminate the jobs running and focus on pre-launch. After the launch calculations were performed, a software module would assess the status and restart the normal queue.

The extensions to IBSYS provided efficiency and smooth operation. The NASA Space Center was part of the project and installed a similar system. IBM turned it into a product and also included a similar product in their S/360 collection.

Note: These scheduling niceties removed some of the longstanding objections to the batch mode of operation.

Management Difficulties

In the beginning computing was a service provided by one piece of the corporate organization to another. This was a closed-shop service. However, long service delays chafed those who had important work to do.

In the early 1960s, many customers were educated as good (or better) than those within the IT organization. Some picked up a little programming knowledge from their colleagues, some from car pools, some rubbed off from the full time programmers. Then when faced by provoking delays, they wanted to program on their own. About this time a wave of Fortran books appeared and the open- shop closed-shop debate was on.

Management did not understand the issues and stood with the customer set since they were the producers, were usually senior individuals, and had the most political clout. "How could you deny them computer access and still hold them to their project schedule?" So the floodgates were opened.

At first the customers did the little jobs and the IT "professionals" did the serious work. When new graduates with university programming experience were hired, some customer organizations set up programming teams that rivaled those in the closed-shop organizations.

However, while they looked equivalent on paper, there were important differences between the two. The career path of the open-shop programmers was with their home organization, which was seldom computing. They attended conferences and read technical literature corresponding to their primary interest. The closed-shop programmers tended to be computer freaks and were interested (more or less) in software, operations, and all things digital. There was some cross over, but the two groups had different interests and did not club together.

Some gorgeous goofs were the result. In one classic case an open-shop data processing program sorted the big file into the same order as the little file before an update pass. In another case, one of the mathematical variables used as a denominator stepped through zero without being recognized until the research report was printed.

Neither side was faultless. Programming accuracy and quality were not formally addressed in either organization. Training was catch as catch can, and many of the college produced graduates had experience with one-off orphan software that looked good on a resume, but was of little use in industry.

And there was one quirk that was never mentioned, at least if you were consulting and wanted to stay employed. Somehow seeing a PhD pecking away at a keyboard seemed to be demeaning and inefficient. Usually he was not composing, he/she had

equations written out on paper and was simply inputting them into the computer. When word processing made the scene, the ratio of secretaries to engineers went down. We substituted professional labor for what had previously been overhead support. This may have provided a service to expensive people who would otherwise have been just waiting, but accounting-wise it "looked" more beneficial than managing the support problem in the first place. The hardware manufacturers were delighted in the increased sales. I never saw a trade-off analysis of the costs involved.

However, the population demanding computer services exploded.

Early Data Base

Some commercial computer systems maintained large files of current data on disk. Today we would call these databases. Many industries needed immediate access to their data in the normal course of business. In the 1950s a salesman in the field would call the home office on the phone to see if there was enough stock in the warehouse before he accepted an order.

In the 1960s big manufacturers ordered IBM devices from a menu and setup "point of sale" systems with an online computer disk file, a dedicated computer, a lot of copper wire, and special purpose dedicated terminals for inquiry and status reporting.

One of these systems was installed at Rockwell to support the construction of the Apollo space capsule. As you can imagine, a hoard of engineers working on a complex product on a tight schedule had a tendency to run all over each other. If one engineer made a change it could affect one or more mating components. Weight was continuous concern. The solution was Change Control.

The management elected to install a computer (an IBM 7010) to maintain a database of current part numbers. Whenever a change was to be made to any component its part number and change level had to be checked to be sure that the drawing obtained by the engineer was the most current version. Further, the corresponding entry in the file had to be marked so all owners of mating components were alerted that a change was in-process.

In a big facility with a few thousand engineers, an online network of terminals to access the current file was set up. In 1967, lessons from this existing system shaped a software package eventually known as IMS/360. The functions were so appealing that the project team consisted of programmers from Rockwell, IBM, Caterpillar, and one consultant (me).

The system was online during the business day and ran batch over night. It was implemented by the central IT staff as a utility. The users had no more access to the internals than they did to the telephone system. The concept was so useful that IBM brought out CICS for their smaller customers.

There was only one known disadvantage: During WWII, and for a few decades thereafter, an engineer in an large airplane factory who needed a current drawing, ordered one from central records by phone and a comely girl on roller skates delivered it to him. With online systems, skate delivery was no longer necessary.

The growth and expansion of the computer-electronics industry allowed many smaller populations and smaller industries to achieve similar systems without the high expense and long delivery time these pioneers paid. Three simultaneous technological developments matured to make this possible. They spawned two new computer services.

Moore's Law and Other Improvements

In the 1950s, at the beginning of the computing era, the telephone system offered mediocre voice service. I had one client that had a plug board (with elderly female operators) into the late 1960s. Whenever anybody located a site for a big computer center their checklist made them evaluate the telephone service before signing the lease. Some telephone companies delivered such poor service based on obsolete technology that we avoided doing business in their service areas.

When we did establish a site (or connect to an existing one) we had to negotiate leased line service with the phone system. This introduced flags on the circuit in all central offices the leased line went through. Otherwise, the line would occasionally get unplugged or tampered with.

Now the phone system has improved. The rules have changed and we have new competitors. Competitors now offer modern service that is end to end digital.

Basic computer technology has also improved remarkably. As often reported against Moore's Law, the speed of computer circuits has been doubling every two years or so and the density of circuits on a chip has improved apace. The net result has been smaller, more reliable, devices. The modem that serves my house has one coax input and three outputs: Telephone, TV, and Internet. It is powered by 120v ac, hangs on the side of a cabinet, requires no air conditioning or fan, and occupies 8x8x2= 128 cubic inches. At one time this would have occupied a tabletop or shelf space in a rack.

Last, but not least, storage technology has improved so much that desktop PCs can contain a reliable disk file with storage measured in megabytes. Without these (mostly unsung) storage device improvements many of our online services could not be offered.

Given these three technological forces, Apple, Microsoft, IBM, and a number of lesser firms offered personal desktop computers. After achieving reliable operation and adequate performance, they sold machines with onboard modems and things started to get connected.

Back in the early1960s, out in California, RAND and the System Development Corporation (both of whom had spare (obsolete) computers) wrote custom software that allowed users on attached terminals to obtain computer services from a shared central computer. They called it timeshare.

After a slow start, timesharing software allowed anyone with a terminal and a phone line to have computer access. Some users needed part of a fast computing machine for a short time. Others were part of an association/corporation that needed access to an online file of data.

A large and very vocal center for timesharing was on the east coast at MIT, they had ARPA money. Some of the manufacturers became interested. GE even designed, built, and marketed a special computer that provided service simultaneously to dozens of terminals. Other companies were formed and offered new computer architectures for timesharing work. IBM, whose bread and butter was big iron running high volumes with batch software, even made a couple of attempts at building a timesharing business.

There were debates in the computer press, and at least one was face-to-face at an Eastern Joint Computer Conference (EJCC). They discussed which was better: online or batch. There was little appreciation that these two approaches served two different user populations with vastly different job/session characteristics.

About the same time the student population, whose workload at the time was unique, needed service. Stanford, Cal Berkeley, and Dartmouth innovated and developed software. Only Dartmouth lasted. At the time, timesharing was just too expensive with the then current state of the three key technologies.

Business wanted service between 10 and 4. Students wanted service from 6 to midnight. Even if the peak hour costs were borne by the user populations, the machines went idle over night with nobody to pay the bills. Several companies were setup to provide timesharing service. The economics killed them.

Italians and PCs

Back in 1962, when I was swimming in shops full of Big Iron, Olivetti assembled the first personal computer (says Wiki). Then there were trickles of activity among the hobbyists in this country. I even had lunch with a pair of devotees on one of my consulting assignments, and frankly they were boring - - all talk about chip speed and nothing about what it was good for (applications).

I guess that was about the status until IBM established a PC project outside of the corporate culture (ending in 1981). The popular literature talks about IBM blessing the desktop, giving it credence, etc. but what IBM really guaranteed was reliability. With IBM behind it, it worked as intended.

One of my regular clients plunged gung-ho into the desktop culture and loaned me one. It was a Compaq “portable”. It worked all right, and allowed me to work on documents

either in my office or theirs, but it was heavy (28#). In its carry-case it weighed as much as a large sewing machine. My right arm is still longer than my left from lugging that thing across a parking lot to my car.

From our primitive beginnings, Word was a breakthrough. Almost as important, but little heralded, was data base on desktops (dBASE). For the first time, a user could manage his data, present it intelligently, and bring a little order into his/her professional life. The early operating systems provided most of the function required, the later "improvements" have all been to promote entertainment and the mass market.

Future

Today (2016) we are back into timesharing, only with advanced technology and cheaper solutions. Many customers already have a desktop computer that can be used as a terminal. The Internet, and the modern methods of charging for communications services, eliminates one significant expense. IBM, and other big computer companies with their backs against the wall, are offering Cloud computing, a form of shared computing.

The world is continuing to change. The need is there. The customers are ready. The online security problem needs to get solved. The centers offering online service may be robust enough (at least none has failed big-time and in public). However, most major recent technological developments have been focused on digital entertainment (where the big money is).

