The Evolution
of
The First Commercial Time-Sharing Company

Interactive Data Corporation emerged from the collaboration of a group of dedicated and talented computer professionals in the late 1960s.

Interactive Data Corp. Headquarters, Waltham MA

"We were part of a special time, where a special group of people gathered together to create a special company and product. We should all feel blessed to have been a part of it."

--- Ruth Belvin, 1977

Dedication

This collective memory is dedicated to Frank Belvin whose leadership, vision and patience were paramount in creating the technology of IDC from the collection of talent he recruited.

A master at “herding cats”.

In the 1960s, Jack Arnow was the manager of the MIT Lincoln Laboratory Computer Center. (Lincoln Labs continues to be a cooperative exchange between MIT and the Dept. of Defense; MIT provides the brain-power, DOD provides the majority of the funding.)

Jack Arnow, President

Documents show that Jack wanted to increase the computer throughput on the S360/67 for the users of the Lincoln Labs installation. As delivered by IBM, running under OS PCP (Primary Control Program), the S360/67 used punched card input, and supported punched card and printer output. Data storage was kept on magnetic tapes and 2311/2314 disk drives. The bottleneck was the limitation of PCP for receiving input and generating output.

To speed up the turn-around time for jobs on the S360/67, Jack assigned one of his top programmers, Frank Belvin, to create a new operating system for an in-house IBM 1620. This OS came to be known as LLMPS (Lincoln Lab Multi Programming System). LLMPS had a specific purpose: Increase the task start and reporting speed of programs on the larger IBM S360/67 processer. This, in turn, would allow more tasks/hour to run on the S360/67.

Frank Belvin

The fastest mobile media available at that time was magnetic tape. To speed up the task start and reporting phase, the S360/67 was configured to receive task input from magnetic tape (no card reader), and to send task output to magnetic tape (no card punch, no printer). LLMPS was designed to allow the IBM 1620 to process "card-to-tape", "tape-to-card" and "tape-to-print" operations. Furthermore, LLMPS was capable of doing all three operations simultaneously. This type of multiprocessing itself was somewhat revolutionary for the 1960s, and it greatly enhanced task throughput at Lincoln.

But Jack wanted more. When Jack had originally negotiated in the early 1960s with IBM for a S360/65, he actually obtained the S360/67, with a full-duplex configuration. Unfortunately, there was no OS capable of running in full-duplex mode at that time or with the associated Dynamic Address Translate (DAT) box, so part of the deal was to run the system in half-duplex mode: One-half would be devoted to S360/65 processing to support Lincoln's production tasks using the IBM PCP OS; the other half would be available to IBM for the development of their TSS/360 Time Sharing System. The long term goal was to eventually run TSS in full-duplex mode with the DAT box enabled on both processors, and use it as the single OS at Lincoln Labs.

By the summer of 1966, IBM's progress on TSS development supported about one user, with an MTF (mean time to failure) of about five minutes. Jack was getting frustrated.

It is not clear, historically, whether Jack heard about the OS research taking place at the IBM Cambridge Scientific Center (CSC) or if the management at CSC heard of Jack's frustration first. Either way, CSC needed a full-duplex S360/67 (with a real DAT box) to continue its work, and Jack needed a reliable time sharing system for Lincoln Labs. In the winter of 1966-67, Jack banished the TSS crew from Lincoln and entered into a joint study agreement with CSC for the development of an OS, then known as CP67/CMS. By the summer of 1967, the jointly developed system was supporting 6 to 10 users with an MFT of over an hour! Frank Belvin was the lead programmer for Lincoln in the CP67 component, and Hal Feinleib was the lead programmer in the CMS component. Progress was moving quickly.

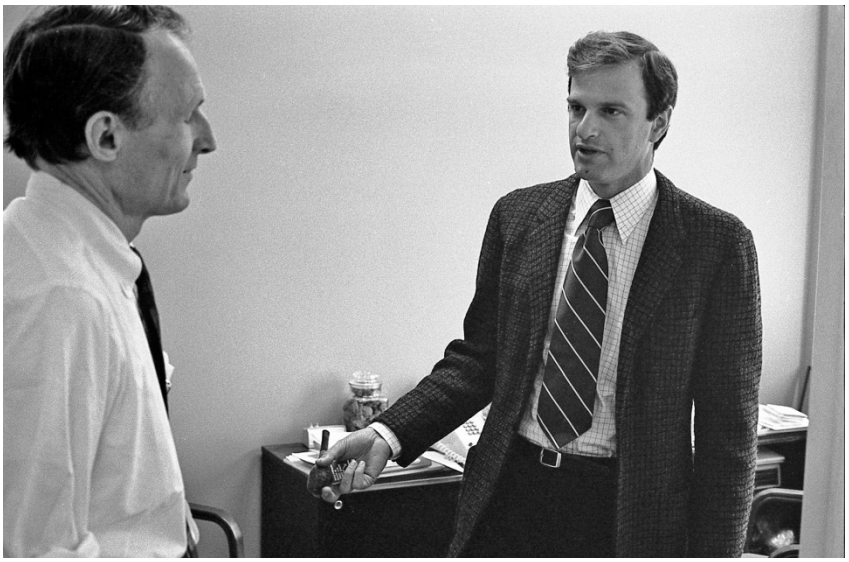
In September, 1967, Jim March joined the systems support team at Lincoln. His initial task was to support the installation and maintenance of the IBM OS PCP, and to assist users in debugging their programs that caused system abends with full memory dumps. Lincoln was his first exposure to S360 architecture. However, since most of his previous work was in IBM 7040 assembler language, his assimilation into S360 assembler was fairly rapid.

After about two months of S360 work, Hal Feinleib pulled Jim aside one afternoon and showed him a CMS terminal. Jim played on that terminal far into the evening and got hooked! Thereafter, whenever Jim had spare time from his PCP support role, he was on a CMS terminal, cramming into his head how it all worked. By January, 1968, Jim made it known that he wanted to transfer to full-time development of CMS, and was allowed to do so.

Jim and Hal worked together until the summer of 1968, when the National CSS[1] company was formed and hired Hal to be their CMS developer. Jim was left as the CMS developer at Lincoln.

Jim March

[1] National CSS was the other main CP67/CMS based commercial timesharing company to form at this time. NCSS was based in Stamford, CT, and their rise and business plans have been documented by interviews with the Computer History Museum in Fremont, CA. (https://www.computerhistory.org/collections/catalog/102702883)



Norm Daggett & Frank Belvin

Simultaneous to the creation of NCSS, Jack Arnow began forming another timesharing company based on the CP67/CMS operating system. He immediately pirated Frank Belvin and Francis Mahoney (Hardware Operations support) to join the company, followed a month later with an offer to Jim March to join in forming the Computer Communications Center (CCC).

The initial business plan for CCC was to locate clients who would benefit from having remote access to large scale computing with personalized applications. CCC would hire a programmer to be solely associated with each client to develop the applications to support its business within a CMS environment. That plan survived about 1-2 months. As has been ably documented elsewhere, Joe Gal of White Weld Banking, with his collection of financial databases, was looking for a new and more powerful computer environment for selling access to his data. Jack and Joe were introduced to each another, and IDC was born before year-end of 1968.

Frank Mahoney

Joe Gal, CEO & Chairman

Bob Seawright

* * * * *

The goal of this document is not to repeat what has been previously noted elsewhere about the business path of IDC, but to focus on some of the technical innovations developed at IDC *in its first six years of operation* to support its client base. Subsequent evolution is left to other authors.

As the reader is already aware, CP67/CMS survived within IBM to eventually be marketed as VM/370. Its targeted user base, from the perspective of IBM, required a different set of software evolutions than that of a commercial timesharing company. Both NCSS and IDC responded, independently, to each set of needs.

From September, 1968, thru January, 1969, IDC did not have its own computer, so IDC arranged for Jim to work

in the CSC building and develop CMS code on their S360/40+ (a S360/40 with a home-grown DAT box – the only such system to ever exist!). In return, the code would be shared with CSC to be absorbed into their main CMS system, if they wanted it.

In recognition that some of the IDC early clients may want access to time on an IBM PCP OS, Bob Seawright was brought on board in the fall of 1968. At the time, Bob was an employee of Union Carbide on loan to CSC to work on running the IBM PCP OS in a virtual machine. Among other hats, Bob was the "go-to guy" for any questions involving the PCP OS, and the face of IDC to clients wanting to run Programs under the PCP OS on a virtual machine.

In early 1969, Norm Daggett was brought on board. Norm's responsibility was to plan and maintain the entire terminal communications network that fed into the IDC computer(s). Without his knowledge and skills, IDC would never have become the nation-wide provider and later the international provider of time sharing services.

Initial Tasks

The first major task on our plate was system reliability. Jack said it best: "We want to be a data utility. You turn it on as you need it, and it stays on until you turn it off."

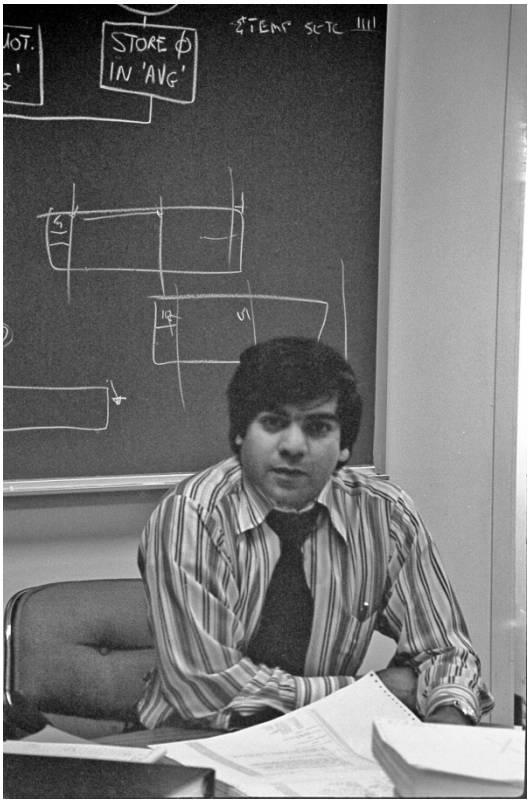
On the CP67 side, this meant solving the CP67 abends. On the CMS side, it meant minimizing, or preventing, data loss across a CMS or CP abend. At that time, CMS was notorious for the loss of the user's entire P-disk storage if an abend happened for even a brief period of time during file system updates. Jim was charged with fixing the problem, and the solution came to be known as the "Dual Directory File System".

The crux of the file system solution was to maintain two parallel file directories on any given mini-disk and maintain a 16-bit block number in an unused portion of the label block (block 3) that pointed to the Master File Directory (MFD) of the "current" directory. We also maintained a second pointer referring to the "current" bit-map block allocation table. At IPL time, the P-disk (and any other defined minidisks) current file directory and its associated allocation bit-map would be read into a portion of the CMS free memory. Then a set of disk blocks large enough for the disk directory and bit-map were immediately allocated for any directory updates, but not used until the directory update was needed.

When any file on the disk was written in a manner to cause expansion or contraction of the file, including file deletion, at file close-time the entire modified file directory resident in memory, and the revised block allocation bit-map in memory, would be written into the pre-allocated blocks of the disk. Finally, a last disk I/O was a rewrite of the label block (block 3) with the revised pointers to the new file directory MFD and bit-map that were already written back to the disk. This sequence guaranteed that the user's disk would be intact with either the old or new MFD, but not parts of both. It should be noted that the modified block allocation mask was revised to show the release of the "old" file directory blocks, and "old" bit-map.

The second major task on Frank & Jack's list concerned an enhanced method of associating FORTRAN file numbers with specific external files. While Jim was working on the dual file system, we became aware of the FILEDEF command NCSS had created. It was clear that this feature was paramount not only to FORTRAN, but could be expanded into a generalized support of the IBM DD card statement. To avoid any copyright infringements, we renamed the command from FILEDEF to FILDEF. The rest is history…

IDC went its own way with FILDEF features. Eventually, it became the basis for supporting all major language processers (compilers and assemblers) under IDC's version of CMS. This included support for access methods BSAM, QSAM, BDAM, BPAM and ISAM. Further support allowed device reference to not only disk files, but tapes, console and the virtual card reader/punch and printer. The result was to take almost any program written for IBM PCP environment and drop it, unmodified, into the CMS environment, replace the DD cards with FILDEF commands, and have it run successfully.



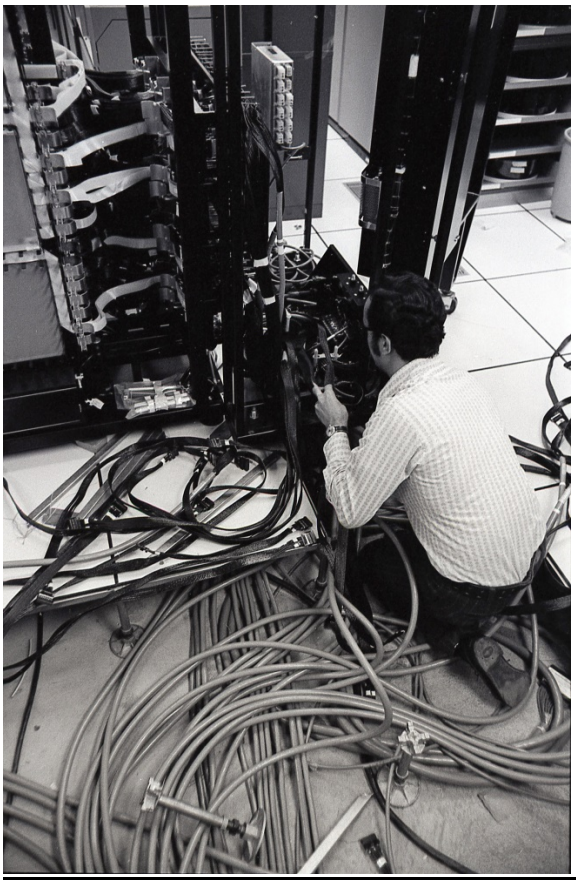
Harit Nanavati

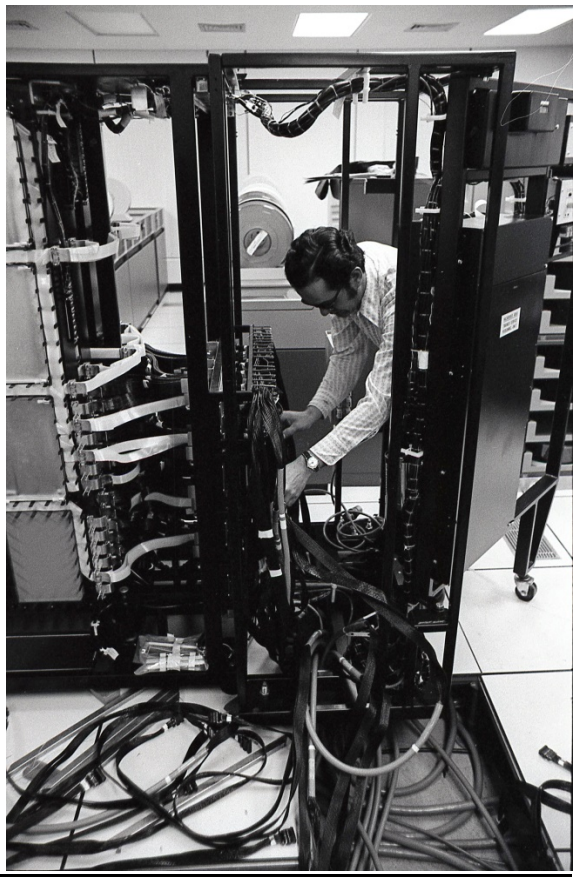
Ellen Wax

By January, 1969, with the arrival of our own hardware, IDC hired Harit Nanavati from CSC to spearhead the CP67 support and enhancements. Harit, in turn, over the coming year was joined by Ellen Wax, Don Allen, and others to assist in CP work.

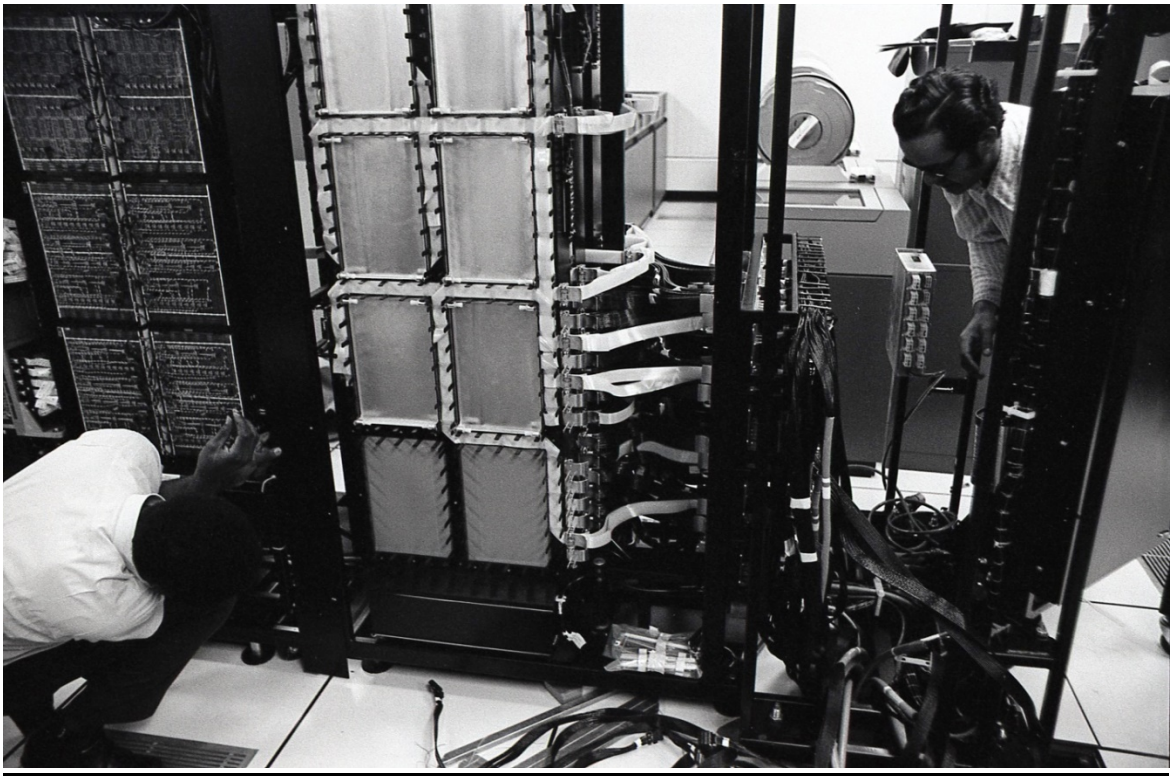
First Computer Room & Setup

Almost ready for system delivery



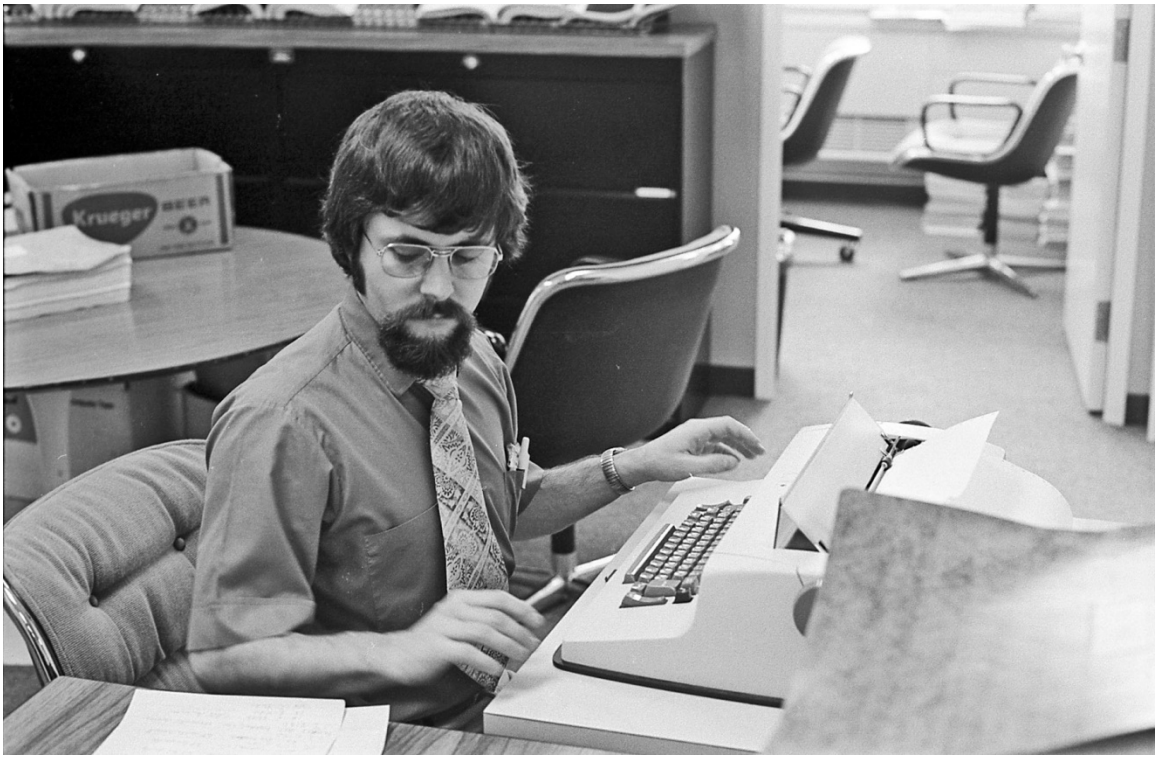


IBM FE's working hard at installation



Ready for system IPL

Frank also hired John Rainier to work with Jim on CMS.



John Rainier

At this time, to further distance ourselves from IBM, IDC renamed CP67 to CS (Control System), and CMS became ES (Executive System). Additional development began under these names – and development there was!

Joan Carlton

A basic accounting system had been built into CP67 for tracking both Problem State and Supervisor State time used by each user. From IBM's point of view, this should satisfy their VM customers, since normally these customers viewed the cost of hardware and software as a fixed overhead to the company. User-time can be internally allocated to that user's portion of the total time and cost of the facility. Initially, CP would automatically punch a card, whenever a user logged off the system, with all the relevant timing information associated with that user session. Jack Arnow wrote the initial billing program (in Fortran) using those cards for our customer monthly billing cycle (thus the need for the Fortran file number support!).[2]
Soon after, Jack Arnow and Frank Belvin hired Joan Carlton to rework Jack's programming on

[2] An interesting sidelight issue for a commercial timesharing company is, what happens to customer accounting during a CP abend? Needless to say, one of the first tasks of CP dump analysis was to extract accumulated accounting information for each client present at abend (manually), punch a card, and add it to the stack of input to Jack's billing program. One day, CP went into a loop and would not dump! Frank Belvin came to the rescue by

the client accounting system. Before Joan, our only client billing was the simple Fortran program that Jack had created in his spare time!

A commercial timesharing facility has different accounting challenges than a single company utilizing its own in-house computer system. First, if we charged strictly based on time, then there is a maximum aggregate time that can be charged, thus a maximum income to be obtained from our clients. Second, we needed to maximize the number of clients that can coexist on the facility, as well as provide an apparent service speed to keep all clients happy with their response time. These issues required solutions on several different levels.

Frank, Harit & Joan began by measuring other CS resources used by each client. These included connect time, time-of-day connection, page working-set size[3], I/O operations per second and per session, paging operations per second, etc. They then developed an algorithm to create an SRU (System Resource Unit) that took into account all the various measurements, and created a monthly invoice in roughly the same dollars as when charged strictly on time used.

This, in turn, allowed us to focus on finding ways to reduce the time a user spent on the CPU while still getting the same (or better) response time and throughput, and simultaneously make space on the real hardware for more users and their applications. The end result was an increase in revenue from more clients with the same investment in real hardware resources.

Along with the increased sophistication of performance and the accounting, the most critical part of billing was to make sure clients kept using the system so they could be billed! CS abend solutions filled the waking hours of both Harit and Ellen. While the Mean-Time-to-Failure (MTF) was steadily increasing, it was still an unusual day if we went an entire operations shift without an abend. Without dump analysis software, memory dumps automatically went to paper, with each dump occupying a paper stack 8-12 inches high. The hallways outside the offices sometimes were stacked waist-high with multiple piles of dumps.

It was probably about a year before the MTF began to regularly reach about 12 hours. It was then that one of the funniest abends was discovered. It seems that all the timing events in CS, by design in CP, utilized the CPU Timer. Thus, when CS was IPLed, the real CPU Interval Timer was set to maximum value, and then forgotten about! After about 12 hours, when the real CPU Interval Timer expired, it created an interrupt that no one had ever coded for: Abend time. But also quickly fixed.

---

sitting at the Operator's console, and began locating all the current accounting data using the addressing toggles and display registers on the front of the S360/67 console! Customer billing data was NOT to be lost.

[3] A working-set is the collection of pages needed for a user's virtual machine (VM) to operate throughout its time-slice without incurring a virtual memory page fault. It was usually calculated when a VM left its running state to enter into an I/O wait-state or time-slice end.

Significant ES Enhancements

As the SRU was being implemented in CS, John and Jim began finding ways to make ES use fewer resources while giving the same results. The major break-through came with these two enhancements:

- Making the entire (or close to entire) ES nucleus re-entrant would reduce the working-set size needed to run any single user, since the re-entrant pages could be in a shared segment.

- Eliminating the use of SVC 202 for intra-nucleus communications would eliminate an SVC reflection into CS and the associated CS overhead.

John and Jim proceeded to check every nucleus module and move each internal variable reference to either a Page-0 reference or a DSECT reference that was automatically allocated from virtual FREE storage at module entry time (and released at module exit time). The DSECT block also contained the register save areas and forward/backward linkage. Furthermore, all internal virtual SVC calls were replaced with BALR links to the desired function entry point.

Another area of concern, in both CS and ES, was "memory leaks"[4]. To combat this problem, the FREE/FRET routines were modified to record the name of the module requesting FREE storage. Subsequent dump analysis quickly showed which modules were allocating storage and not returning it in a timely fashion. Memory leaks solved!

One other change was added to ES to minimize the nucleus working-set during any given time slot. We traced the ES nucleus reference disk-seeks in groups of three. We then did a frequency analysis on the entries that showed which pages in the nucleus were more likely to reference each other during normal operations. We were then able to rearrange the nucleus modules so that those most likely to reference each other were placed into the same pages, if possible. This, again, reduced the working-set size needed to support a VM during its time-slot.

Once these performance enhancements were complete, attention could be turned to adding new features to the ES environment. Perhaps the most significant of these was the Integrated Symbolic Debugger (ISD).

The ISD was the brain-child of Bob Frankston and Mike Wyman[5]. The overall effort to create this feature included time of both men over a two year period. In the end, the achievement was a major ES enhancement requiring its own full manual to describe all its features.

[4] A memory leak was defined as a block of memory requested for allocation, but not released when the requesting program completes. Eventually, all available memory for dynamic allocation gets used up, and the associated OS must abend.

[5] Both Mike Wyman and Bob Frankston joined the IDC ranks as part of the merger of CCC and White Weld into IDC. Bob was still a high school student when he was employed at White Weld, and was a student at MIT at the time of the merger with CCC. He continued as an IDC employee on into his undergraduate years at MIT. Ultimately, after

Bob Frankston

Mike Wyman

As the name implies, the ISD allowed program developers to perform debugging operations, such as breakpoints, variable displays and modifications, code insertion and/or deletion, and execution tracing - all using the syntax and formats of the native language of the program. Normally, such debugging was done at the Machine Language level, or the Assembler Language level, often utilizing the CS Break and Display commands. ISD supported FORTRAN and COBOL native languages, as well as BAL. Now, a user could switch between language types while in the debugger.

Following that effort, Mike created the XDMS database product. This was all before IBM standardized SQL as the preferred common database language. So Mike had a "blank slate" on which to create a useful time-sharing database language. The result was, and is, fully described in its own set of documents.

The porting of the ANALYSTICS product and the First Financial Language (FFL) from the SDS 940 computer (originally part of the White Weld Interactive Data Services product line) to the CMS environment was spearheaded by Alan Dziejma. Both of these products were complete subsystems within the CMS environment, and were fully documented in their own operational manuals.

---

leaving IDC and starting Software Arts, he was the main implementor behind VISICALC, arguably the first spreadsheet application to appear in the PC market.

The Executive Sandbox

Software development was not the only important concern of the budding ES group. Morale under the demanding schedules was also a concern.

While John Rainier & Jim March were busy with their coding activities, they were wondering what to get for Frank Belvin's Christmas present. At the time, they saw a mail order catalog advertising an oversized ashtray, shaped like a kids' sandbox, filled with sand and promoted as the ideal addition to any executive's desk. They decided to "one-up" the concept and built a wooden sandbox out of particle board, only much larger than the catalog version.

The box was approximately six feet square with a 12-inch seat on all four sides. The inside was about two feet deep and filled with Styrofoam chips. They built it in John's basement and snuck it into the office one night placing it in the hallway in front of Frank's office.

We had no idea how it would be received, but it turned out to be a major enhancement to not only the décor, but was in frequent use for small gatherings of weekly or ad hoc meetings. It became a central symbol of the innovation and motivation of the entire technical staff.

It also proved that programmers could use their hands as well as their heads!

Joan Carlton & Ruth Belvin with Sandbox in background

Significant CS Enhancements

Thinking about the computer hardware availability on today's desktop, it's hard to imagine the constraints faced by the CS developers. Where we commonly allocate RAM to a PC in quantities measured in gigabytes, the initial IDC half-duplex S360/67 had only 512K bytes (that works out to 128 4K pages for both the CS resident nucleus and user virtual memory). It was exciting times when we got a 50% increase to 768K after about two years. Even after the support for full-duplex was implemented, the two CPUs had to share only 1280K bytes.

Not until the arrival of the S370/158, in the late 1970's, did RAM expand to 2048K bytes - huge by the standards of the day. However, even with these relatively meager RAM allotments, with the continued performance enhancements, we were able to handle several hundred concurrent users on the 158, with 0% wait-state, and as low as 20% supervisor-state[6].

After locating and fixing the majority of CS abends and implementing the SRU accounting feature to CS, Harit's group produced several major enhancements. To summarize, these included:

- Post-paging and pre-paging of a user's page working-set. This significantly reduced the number of page fault interrupts when re-dispatching a VM to use its next time-slot.

- 2780 & 3780 support for Remote Job Entry tasks under CS/ES and CS/OS-PCP.

- Virtual Access Method (VAM) – A form of virtual BDAM that utilized the CS paging mechanism, and was considerably faster in data transfer, as well as using fewer resources for data transfers.

- A new virtual machine scheduling algorithm that utilized a three-pool, two-queue concept to minimize a user's console wait time when ready to read more console input, while maximizing compute-bound applications use of CPU resources.[7]

- Creating customized code in the COMTEN communications front-end control unit, and CS support of that interface. The interface was designed to minimize interrupts between CS and the COMTEN. The reduction of interrupts in CS reduced required Supervisor State time to support a client VM console, making more time available for Problem State execution.

[6] All virtual machines run in real Problem-state so CS could respond to the virtual usage of supervisor instructions with the appropriate simulation. So, with no Wait-state, and low Supervisor-state, this meant the remaining CPU time was devoted to running virtual machine code, which was VERY billable.

[7] A constant client complaint in early versions of CS/ES centered on having to wait for their terminal to "unlock", to allow the next line of console input when doing trivial activity such as EDIT input mode. We "fixed" that! The goal was to have that unlock-wait not exceed 0.1 second.

- Support for “loosely-coupled” S360/67 processors via a channel-to-channel (CTC) adapter and shared control units for DASD, tape and terminal I/O. By sharing the master user directory, an incoming user could be directed to logon to either processor, thus allowing load-balancing between otherwise two independent CS systems. The physical distance between processors was limited only by the allowed cable lengths for the CTC adapter. Likewise, when “tightly-coupled” processors were eventually supported, they too, could be coupled to produce even better load balancing.

- Support for “tightly-coupled” S360/67 processor in full-duplex mode. This allowed a logged-on user’s VM to be dispatched to its time-slice on either of the two coupled CPUs, based on which CPU was available at that moment. The particular CPU could change from dispatch to dispatch instance, since all memory, DASD, tape and unit record devices were shared by both CPUs.

Post- and Pre-paging

Before this enhancement, when a user VM was dispatched for its round-robin time-slice, it began with a single page of virtual memory, namely the page identified by its virtual PSW. As a program began to execute, it would eventually reference another virtual memory page, resulting in a “page-fault” whereby CS would determine the location of the referenced page on the user’s paging device, and proceed to bring it into real memory for the user’s continued execution. This process resulted in several sequential actions by CS in Supervisor State, all of which accounted for time that was non-productive to the user, and clogging CS from performing other critical tasks.

The solution was both elegant and simple. While the above scenario did play out for a newly established VM, when that user’s VM had exceeded its time-slice, its current “working set” was determined, and CS built a CCW chain that would copy those pages from real memory into a specially allocated space on the 2301 Drum in sequential bands using a single SIO. Then all the user’s real memory pages were released to the free page pool and other users were dispatched.

When the suspended user’s VM was eligible for dispatch again, its working-set size was noted, and sufficient real memory pages were allocated to hold that working-set. Then, a CCW string was built to read their working-set pages from the drum directly back into the newly allocated real memory pages, all with a single SIO. Only when that set of pages was back in real memory and the associated page table entries created was the VM again dispatched. This overall action greatly reduced both the number of interrupts to be serviced by CP, and also reduced the amount of Supervisor State time required in support of the user. It also sped the user’s terminal apparent response time to its current operations.

2780 & 3780 Support

Several clients either ported their own programs and/or data into the IDC system, or required regular data updates for their use. Historically, they had entered their data on punched cards, and frequently needed output reports on a timely basis that were larger than practical on a terminal. To meet their requirements, they now used IBM 2780 (and later IBM 3780) remote batch service devices. These devices were similar to the CPU's own card reader/punch and printer, but instead of being connected through local unit-record control units, they were designed to connect to a bi-synchronous communications line that could span long distances (miles) to connect to a remote CPU. The IBM CP67 software did not support this type of connection, so Jay Walton first developed its support in CS for the 2780. Ellen Wax took over the support, and ultimately expanded it to include the 3780. This allowed a remote user to feed card decks directly into the user's virtual card reader, and direct the user's virtual card punch and printer output spool files to the attached remote device.

Virtual Access Method (VAM)

Harit had conceived of a virtual method of DASD data transfer before leaving CSC, but never had the opportunity to code and debug it. IDC gave him that opportunity.

The concept was simple: Define a minidisk to a user that is formatted to look like a paging device. Then allow a user program to communicate to CS with a virtual RPQ instruction to manipulate the CS paging and swap tables.

To "read" a 4K block from the minidisk, the user program would refer to a virtual memory page address, and a block (page) number on the minidisk. CS would set up the paging tables to bring in the indicated block when the user program next referenced a location within the virtual memory page. Thereafter, the content of that page would be whatever was read from the minidisk plus any changes made by the running application in the user VM, and would also automatically be part of the user's working set under the normal CS paging algorithms.

To "write" a 4K block, the user program would indicate a virtual memory page, and a block number on the minidisk. CS would then schedule that page to be "swapped' out to the specified minidisk area, and retain the page within the user's working set under the normal CS paging algorithms and locations.

This method of I/O, while very efficient and fast, did not fit well with the normal ES file system, since that file system was built around 800 byte data blocks defined on the IBM 2311 DASD. But, with the advent of the NS system (See New System, below), we were able to design a new file system built around 4096 byte blocks and therefore made extensive use of VAM for data storage.

Virtual Machine Scheduling & Dispatch

The traditional method of scheduling and dispatch had been to "round-robin" all currently defined virtual machines (users), allowing each, in turn, to occupy the CPU resource for a specified time period, perhaps several seconds. This design, when applied to several dozen users, meant that any one user's VM could be "locked-out" of its terminal for several seconds, or even minutes, while other users got their time-slice.

For users doing data-entry tasks via their terminal, this created a very unpleasant environment. To combat this, the allocated time-slices could be shortened, but that would add significantly to CS overhead in user swapping tasks.

The solution was elegant. Each user was flagged as being in either a running or waiting state. Each user's running history was maintained, and the Scheduler identified the user as belonging in one of five queues. Two of the queues (known as Q1 & P1) were reserved for VMs that used minimal real resources; namely, CPU time and working-set size. For VMs using more resources, they were moved to the Q2, P2 & P3 queues:

1. Q1: If a VM had a running history with very short bursts of CPU usage between terminal reads, and kept its working-set below a predetermined maximum threshold and was runnable, it would be placed in this queue. VMs in this queue had their working-set resident in real memory, and were dispatched to a CPU in a round-robin manner.

2. P1: When a Q1 VM completed a predetermined number of dispatches and returned to a console-wait state, it continued to qualify for Q1 status, and would be moved to the P1 queue, and its working set post-paged to the drum. Another P1 VM would be moved into the Q1 queue. Each user's VM in P1 was implicitly flagged as qualified to return to the Q1 queue when console-wait ended. If a Q1 user's VM exceeded its CPU time allocation or working-set size limit, it would be moved to the P2 queue and subsequently scheduled under those rules (see below).

3. Q2: If a VM in P2 is runnable, and its working-set will fit into existing real RAM, it is moved into Q2, and its pages loaded into RAM. When the Q1 has no runnable users, then VMs in Q2 will be dispatched in a round-robin fashion. Time slices in Q2 are slightly longer than Q1 time slices. Failure to get to a console-wait state before completing the Q2 time slice moves the VM to the P2 pool for subsequent P2/Q2 rotation. Q2 VMs, by definition, would be allowed to run for a slightly longer time-slice, and/or a larger working-set, but it would now only have access to the CPU when no Q1 user was running, or after waiting for a predetermined time period, whichever came first. If a Q2 VM exceeded the Q2 time allocation, or entered I/O-wait (but not console-wait) it

would be moved into the P2 queue in a FIFO position. If the VM entered console-wait while running in Q2, it was post-paged and moved to the P1 queue.

4 P2: If a Q2 user's VM exceeded a predetermined number of time-slices without entering into a console-wait, or entered I/O-wait (pending disk or tape operation) it would be post-paged and moved from the Q2 queue to the P2 queue in a FIFO manner and the first P2 runnable VM would be moved back into the Q2 queue, assuming pre-paging space was available in RAM.

5. P3: If a user's VM continued to appear as "compute-bound", that is, running through its Q2 time slots without entering into an I/O or console wait-state, then it would be post-paged and moved to the P3 queue. From here, it may periodically be moved back to the Q2/P2 state, but with longer intermittent delays.

When a user's VM entered a read console-wait state before expiring its time-slot, it would be post-paged and moved into the P1 queue in order to get them back into a Q1 running state as soon as the virtual machine became runnable again. From there, based upon subsequent CPU usage, it could either remain in the Q1/P1 queues, or rotate back down through the Q2/P2/P3 queues.

To insure that a P1 user's VM could always get back to Q1 when qualified, a minimum number of real memory pages were always reserved for pre-paging these users, as needed. This insured that users doing simple terminal input got the shortest possible response time to any input transaction.

The overall effect was to give terminal data input users high access to the CPU, since their VM used very little CPU (and memory) during its time-slice, and at the same time gave longer time-slices to users doing more compute-bound tasks, gradually decreasing their access to the CPU, until they went into a virtual wait-state.

COMTEN Support

The normal terminal front-end to the S360/67 was an IBM 2701 Communications controller. This controller evolved over time into a 2702 and a 3701, each with increased capacity for concurrently attached terminals, and higher communications speeds. The drawback on these IBM controllers was both the lease price and the communication protocols between them and the CPU.

COMTEN produced a competing communications controller, which IDC brought in-house and began to use. Not only was it price-competitive, but it was itself programmable. We proceeded to create new programming within the COMTEN to reduce the number of interrupts generated

back to the CPU during any terminal I/O process. We also provided the terminal users with an "unlocked" terminal as much as possible, even by queuing the input locally and passing it to the CPU and associated virtual machine as requested.

Loosely Coupled CPUs

Even with the advanced scheduling/dispatching algorithms in place, as the number of concurrent users increased, we needed to expand and make use of a second S360/67. Then the problem became one of load-balancing between the two CPUs.

One solution was to direct incoming terminals to either one or the other CPU on a constant basis. This could be easily done by having a COMTEN controller for each CPU, with users constantly directed to the associated CPU. However, this could still lead to one CPU being heavily used while the other sat idle, or nearly so.

The other solution was to have both CPUs attached to all the device control units in a shared configuration, where either CPU could access any incoming communication line, and any DASD or magnetic tape device. Furthermore, a Channel-to-Channel adapter (CTC) was established between the two CPUs. When a new communications port (i.e., user) became active, one of the CPUs would intercept the incoming logon request, then communicate with the other CPU, via the CTC, to compare current system loads. If the other CPU appeared to be less heavily loaded, the first CPU would pass off the logon dialog to the other CPU for action. Otherwise, the first CPU would continue to allow the logon to occur locally.

It did not matter which CPU ultimately accepted the logon, as both had full access to all relevant devices, and the user could proceed as normal. Device accesses, on devices that were shared, were preceded with appropriate "lock" and "unlock" actions via the CTC so as not to be contradicted by the other CPU.

Tightly Coupled CPUs

The IBM S360/67 had an optional configuration for dual CPUs to share memory and all data channels. This configuration was imagined to allow two virtual machines to operate simultaneously, each with its own CPU and allocated memory and devices. While great in concept, the hardware was available long before IBM programmers had any software to implement this feature. We decided to not wait for an IBM solution.

The entire CS team shared work on this feature. Initially, Joan Carlton and Mike Wyman worked on the resource locking problem that is paramount in this hardware configuration. The main problem lay in allowing only one CPU at a time to utilize CP (CS) code that was not reentrant. Toward that end, there was a special instruction, "Test and Set", designed to create a software

“lock” for an associated block of CP code. The question, then, became one of dividing CP up into somewhat independent areas, each with its own lock. Testing soon showed that while it did allow both CPUs to operate in Supervisor State simultaneously, there became classic cases of each CPU waiting on a locked resource of the opposite CPU at the same time. Not a good condition!

The final solution came with Frank & Mike creating a master lock for all of CP on a particular CPU. The opposite CPU could operate in Problem State, but if a pending interrupt queued on that CPU, it had to wait for the other CPU to unlock CP before processing that interrupt. This turned out to be the most efficient method of full duplex operation and was thus coded by Joan Carlton as the final solution.

Testing all of this code was another interesting problem. Before IDC obtained the full-duplex system in-house, the only other S360/67 full-duplex system in the area was at the IBM Cambridge Scientific Center. Frank negotiated test time on this computer on Sunday mornings from midnight to 6 AM. So, for many weeks prior to installing our own system, Frank Belvin, Harit Nanavati, Ellen Wax, Jerry Cristoforo and Joan Carlton did all the support testing on the CSC computer on early Sunday mornings.

CS was used in production by IDC until 2001 when IDC migrated from the Waltham Data Center to a new Data Center in Boxborough, MA. At that time CS was replaced by IBM’s VM/370 time sharing operating system.

New System (NS)

When John Rainer and Jim March finished the previously mentioned ES enhancements near the end of 1970, Jim left the ES group to help open a San Francisco computer center. After about three months, Frank Belvin recalled Jim to the Waltham office to head a new documentation group. It was to be a six month assignment, but turned into two years.

Finally, Frank approached Jim about a new programming task. In our never-ending goal of getting more simultaneous users on the CS environment, we needed to think about a new virtual operating system that would require fewer resources to run.

Specifically, the New System (NS) should occupy fewer pages of virtual memory, operate with a new DASD file system based on the VAM I/O feature, use a CS RPQ instruction for virtual console I/O to avoid all the normal types of virtual machine interrupts associated with console I/O, and be as re-entrant as possible. Jim, in concert with Frank, designed and coded the NS system.

Like ES before it, the only non-reentrant page was page-0. Besides tables of constants, page-0 contained all the First Level Interrupt Handlers (FLIH) that, if necessary, passed control to their Second Level Interrupt Handlers (SLIH) in higher pages.

The main nucleus occupied the residual of page-0 and extended into page-1 (X'1000'); all code was entirely re-entrant. Page-2 (X'2000') became a SVC functional page (more on that below). Page-3 (X'3000') became a VAM I/O buffer, and page-4 (X'4000') was the Transient Program load page. Main programs could load, starting at page-5 (X'5000'). In contrast, CMS loaded Transient Programs into address X'11000', and main programs started at X'12000'. Normal CMS virtual machine memory size was X'40000', or 64 pages, each 4K in size. Under NS, usual virtual memory size was about 8-10 pages; that being what was needed to load and execute most main programs. Thus the first requirement of smaller virtual memories was easily satisfied.

The VAM facility in CS made reading of executable memory images easy and fast. The "loading" of Transient Programs became simply the mapping of a page on a VAM formatted DASD to the Transient Program load page. Then, the first reference to the actual page caused a normal page fault to bring the code page into the virtual memory space. This type of real I/O has always been specified as a high priority type of action, so VAM merely allowed NS to take advantage of that priority path.

The SVC functional page was the answer to keeping the resident nucleus as small as possible. All SVC SLIH routines were kept in a "chained" set of pages, where the start of each page had a variable length table whose entries contain a list of supported SVC calls within the current page, and their entry point address with the page. There was also the page number of the page containing the next higher set of supported SVC codes, and the next lower set of SVC codes. So the SVC FLIH would check the current SVC code table, and if within the range of the supported SVC codes in that page, link to the associate code location. If the desired SVC code was not

present in the current code table, it would invoke a VAM operation to map in the next lower or higher page in the chain and continue the search until found, or if not found, enter the appropriate error state. The SLIH routine would, by design, always return via the link to the FLIH for continued execution at the interrupt address.

In general, the SVC calls were not related to the SVC calls of either ES or OS PCP, but were used to supply basic resources to main programs. These included such services as:

- Free storage allocation and release
- Disk file manipulation
- Magnetic tape operations
- Virtual Unit-record device operations
- Dynamic program loading and/or linking
- Calling to defined Transient-area programs
- User console I/O

Several ES utility programs were also created to allow transfer of both code and data files between ES and NS. This allowed users to create and debug their own programs under ES, then when operational, transfer them into the NS environment for production use.

With these enhancements, by the end of 1974, NS had become an expedient solution to IDC client “tailored” programs, while keeping their resource requirements small.

Jim moved into the Tailored Systems division of IDC to make use of NS on a client practical level.

Cast of Characters

Jack Arnow – Jack graduated from MIT in 1950 and joined the MIT Lincoln Lab where he worked for seven years on the development of the USAF SAGE system – the largest on-line real-time system in existence then. He became the director of the Lincoln Lab main computer center, from where he created the joint-study effort with the IBM Cambridge Scientific Center to build the CP67/CMS operating system. In the spring of 1968, Jack began the formation of Computer Communications Center (CCC), a commercial time-sharing venture. On December 18, 1968, CCC merged with the Interactive Data Services division of White, Weld & Co. to form Interactive Data Corporation (IDC). Jack remained as President of IDC after the merger.

Don Allen – Don joined IDC in 1969 in support of the CS group. During his time at IDC, he was initially responsible for the concept and development of the CS post- and pre-paging facility. He went on to work on the design and implementation of the Q1/P1-Q2/P2 scheduling concept. Once in place, that scheduler was one of the most positive ways CS supported the increased user load, and changed the way IDC could support an increase of simultaneous users.

Frank Belvin – Frank received an EE degree from Georgia Tech in 1955 and a Masters degree in the same field from MIT in 1961, after which he joined Lincoln Labs where he met Jack Arnow. Frank's early work for Jack was in the development of the Lincoln Labs Multi Programming System (LLMPS). When Jack formed the joint study effort with IBM's Cambridge Scientific Center to develop CP67/CMS, Frank was the prime systems programmer from Lincoln in the development of the CP component. From LL, Frank was recruited by Jack to be the VP of Software Development at CCC.

Frank Belvin - 2009

Joan McDonald Carlton – Joan arrived at IDC in June, 1969, from the MIT Instrumentation Lab. Her initial assignment was to develop special projects for Jack Arnow. Reporting to Frank Belvin, her first project was to rewrite the customer billing system in COBOL, originally written by Jack in Fortran. Eventually, Joan officially joined Harit Nanavati's CS group where she was instrumental in the development of the full-duplex CPU support of CS. She also became involved in tweaking the customer accounting components, and rewriting (again) the customer billing system, this time in XDMS. That system remained in use for 25+ years.

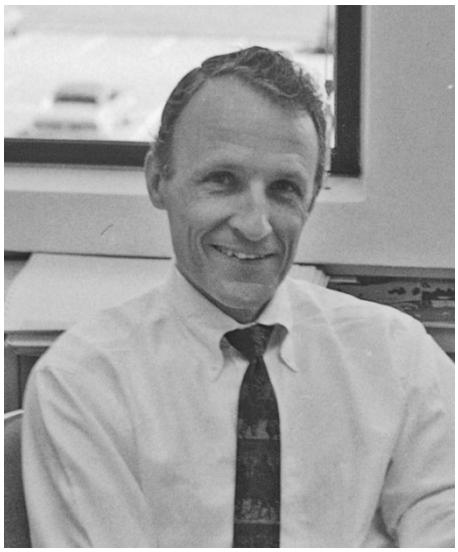
Joan's career at IDC spanned 39+ years during which she managed many technical areas and was the IT Director for many of those years.

Joan Carlton

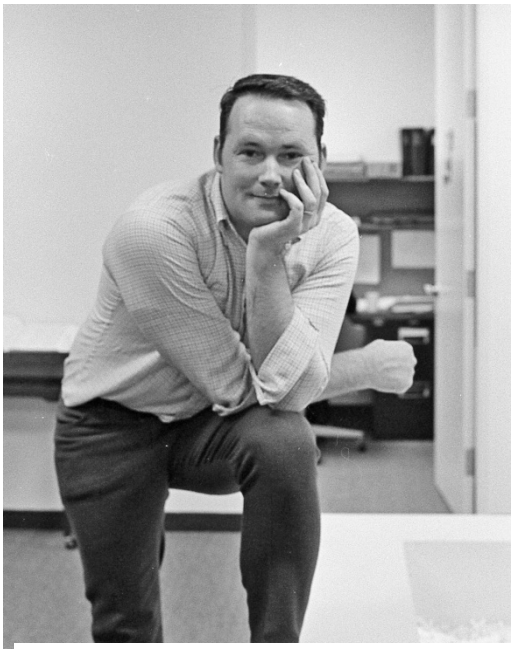
Jerry Cristoforo

Jerry Cristoforo – Jerry held a Bachelor of Arts degree from Northeastern University in Mathematics. He went on to work at the Cambridge Scientific Center from 1969 until June, 1974, when he joined IDC in the CS development group. There, working with Harit and Ellen, he worked on the continuing exercise of CS dump analysis, and eventually wrote software to assist in that effort. Jerry remained at IDC for eleven years before moving on to eventually having a 25 year career at State Street (1994 - 2019) in Hangzhou, China.

Norm Daggett – Norm was a 1947 graduate of MIT with a degree in EE. While working with Jack Arnow at Lincoln Labs, he became one of the principle people involved with long distance computer communications. He came to IDC from LL at the time of the IDC formation and was responsible for the installation of the entire IDC communications network, world-wide. He was one of the first to employ Time-Division Multiplexing for handling terminal communications from remote corners of the world into the IDC computer center. He also designed and built a complete "patch-panel" to enable the rerouting of incoming leased-line traffic in order to maximize user access to the IDC system across AT&T line failures.



Norm Daggett

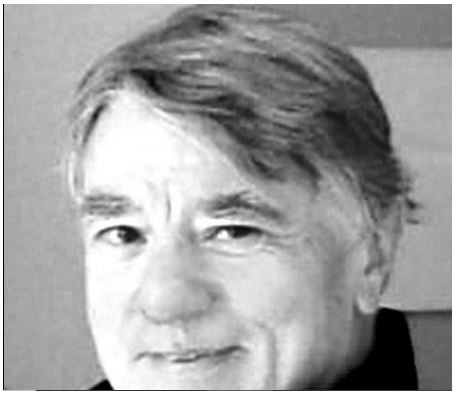


John Devlin

John Devlin – John arrived at IDC in 1971, as one of the first new hires in our revised documentation group. His photographic skills soon became known, and he became the de facto company photographer as well as documentation archivist supreme.

Bob Frankston

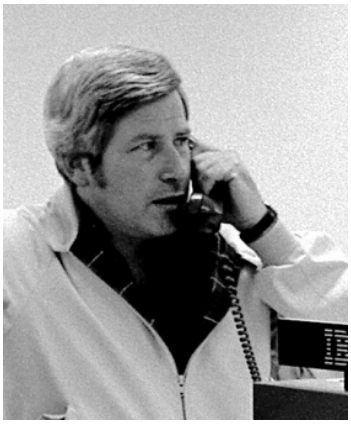
Bob Frankston – Bob came into IDC as a result of the merger between CCC and the White Weld Banking Analytics system team. Bob had been employed as a programmer to help create the FFL (First Financial Language) on their SDS-920 computer for Joe Gal. Bob was still a high school student at the time. He went on to become a student at MIT while continuing to work half-time for IDC, and is listed in the Alumni Directory as a member of the class of 1970. His major contribution at IDC was the design and much of the coding (along with Mike Wyman) of the Integrated Symbolic Debugger (ISD). He also assisted Mike Wyman in porting the Analytics financial databases into the IDC environment, as well as the FFL product.



Joe Gal

Joseph Gal – Joe was a 1958 MIT alumnus and a 1960 graduate of the Harvard Business School. While employed by White Weld, he became interested in the application of computers to financial analysis and investment decision making. That led to the development of the First Financial Language (FFL) for use with COMPUSTAT, the Standard & Poor's data base. All the FFL development was on an SDS-940 computer as a service for White Weld's customer base. When Joe (and his staff) merged with CCC to form IDC, Joe became Chairman and CEO of IDC.

Susan Kruger – Susan was a 1974 MIT MS graduate who subsequently came to IDC. She initially worked in the ES development group with John Rainier, ultimately becoming its manager. She remained with IDC for over 40 years in various technical management roles.



Frank Mahoney

Francis Mahoney – Frank, more than any other person, was responsible for the successful installation and operation of all the computing and peripheral hardware for 24-hour operations at the IDC computer center. He not only ran interference with IBM on all acquisitions and problem resolutions, but pioneered the concept of 13-hour operator shifts, with three shifts per week. This allowed a uniform one hour overlap as each shift transitioned into the next, and allowed the operators a standard 4-day "weekend". Frank had previously been the operations manager of the MIT Lincoln Laboratory computer center.

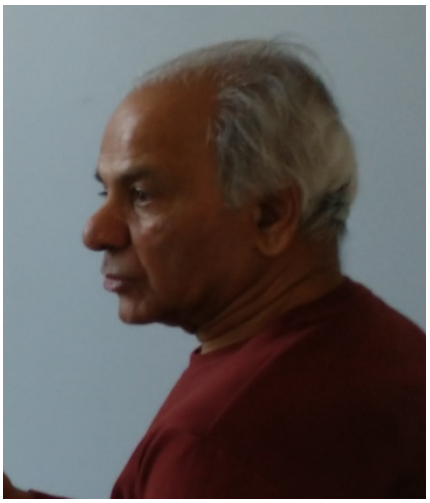
Jim March & Jane Evje - 2016

Jim March – Jim was introduced to computer programming in 1960 as a freshman at MIT. He took the "Intro to Computer Programming" class under the instruction of Professor John McCarthy (who later went to Stanford University to establish its Artificial Intelligence Lab). Jim went on to take every undergraduate programming class offered, which were all "elective" classes; there was no IT major then.

Leaving MIT after his junior year, Jim joined the USAF and was assigned to the USAF Personnel Research Lab at Lackland AFB, San Antonio, TX, as a programmer and instructor. There, Jim participated in programming statistical analysis programs using large magnetic tape files, and creating utility programs for the lab. He left the USAF to join the MIT Lincoln Laboratory as a systems programmer in 1967, initially supporting their IBM S360/65 PCP OS.

Jim left Lincoln Lab to join CCC in September, 1968, in support and development of CMS.

Harit Nanavati – Harit was an MIT alumnus (1964 MS in Mechanical Engineering). His initial programming experience was in assembler on the MIT CTSS system as part of his Master's thesis. Subsequently interviewing with IBM, he took a position at Kingston, NY, where he spent a year developing applications for the 2250 display unit. In the fall of 1966, he joined CSC to work with the team developing CP67. It was there that he initially had the idea for the Virtual Access Method (VAM) that he subsequently brought to IDC, and completed its development while managing the CS development & support team.



Harit Nanavati - 2019

John Rainier

John Rainier – John was an MIT alumnus with both a Bachelors (1964) and Masters (1965) degree, before working as a Research Assistant in the Civil Engineering department at MIT. In the summer of 1968, he took summer employment at Lincoln Labs, working for Frank Belvin and Jack Arnow on a computer time accounting system. He was recruited by Frank to join IDC in March, 1969, to work with Jim March on the development of ES, eventually becoming the ES Group Manager. Later, he was promoted to Head of Applications Programming.

Bob Seawright – Bob was a math major (B.A. and M.S.) from U. of Mississippi. While enrolled in the Ph.D. program at U. of Wisconson (Madison), he was recruited by Union Carbide as a scientific programmer. When Union Carbide began negotiations with IBM for a S360/65, Bob volunteered to assist in the evaluation phase at IBM.

During a visit at IBM, he was made aware of the development of a new Operating System at the IBM Cambridge Scientific Center known as CP67. Union Carbide opted to send Bob to Cambridge to work on installing and operating the OS PCP under CP67. While there he wrote extensive BAL programs to enhance the

Bob Seawright - 2019

ease of OS operations in the virtual machine. Paramount among those were Online/OS, JCLscan and Super-Scratch. It was there that Bob met Jack Arnow, Frank Belvin and Jim March as they were forming CCC.

Bob joined CCC in November, 1968. At IDC, Bob was instrumental in assisting the support of the various IBM language compilers and run-time environments within the CMS environment, and in assisting IDC clients who insisted on running a virtual OS PCP under CS.

Ellen Wax – Ellen was a systems programmer at MIT's Computation Center in charge of supporting the OS/360 system when she was recruited by IDC. She was hired by Frank Belvin in December, 1969 to join the CS Group. Previously, Ellen was at the Raytheon Company in Bedford, MA where she programmed a large-scale simulation model. In the CS Group, in addition to projects such as remote terminal support, Ellen focused on improving system reliability and performance. During her 37-year career at IDC, her role expanded to include all aspects of CS mainframe support including hardware acquisition and keeping the system up-to-date and viable with such features as **data replication to a remote site for disaster recovery.**

Ellen Wax - 2015

Mike Wyman - 2009

Mike Wyman – Mike started programming in 1963 at McKiernan-Terry Corp, developing computer models of towed underwater vehicles on an IBM 1620. He then spent three years at National State Bank, Elizabeth NJ, developing banking applications, first on a GE 210 (the machine originally built for Bank of America's Project ERMA) and then on a GE 415. Following this, Mike joined McGraw-Hill as a systems programmer working with IBM's OS/360 and teaching COBOL programmers JCL and memory dump interpretation.

Mike joined the Interactive Data Services Division of White-Weld in April of 1968 as a system programmer to

work on their SDS 940 system. This division was spun off from White-Weld in December of 1968 and merged with Computer Communications Center (CCC) to form Interactive Data Corporation (IDC).

Mike's career at IDC spanned over 30 years during which he was responsible for architecting a number of IDC's early financial products including the XPORT portfolio analysis system, the Cobug COBOL debugger, OS programming environment emulation (ISAM, Sort, etc), the Integrated Symbolic Debugger, the XDMS 4GL and the Batch Exec system.

When the IBM PC came on the scene, Mike helped IDC embrace this technology by producing a family of products which seamlessly integrated the user-interface capabilities of the PC with the back-end databases of IDC's financial data. These products included PC Screen, DataSheet, DataFeed, DataWindow and RemotePlus.

In addition to producing end-user products, Mike was also instrumental in the development of the core operating system and network capabilities of IDC. Contributions included new CP scheduler, full-duplex 360/67 support and IDC's X.25 network.

Acknowledgements

This document could not have been completed without the assistance of Jane Evje, my wife and chief editor, John Devlin, prime photographer, Mike Wyman, John Rainier, Bob Seawright, Harit Nanavati, Ellen Wax, Joan Carlton and Jerry Cristoforo, all of whom contributed both to the technical success of IDC and many of the details of this memoir.

Afterthoughts

As I approach my own 80th birthday, I've become aware that of the five individuals that began CCC, I am almost the last man standing (along with Bob Seawright). In that role, I felt compelled to set to paper some of the technical facts, as I can recall them, regarding the birth of that company.

I can only hope that my fellow contributors, and other interested readers will gain some clarity on the contributions of those early days.

-- Jim March