An Interview with

KENNETH W. KOLENCE

OH 348

Conducted by Jeffrey Yost

On

3 October, 2001

Palo Alto, California

Kenneth W. Kolence Interview

3 October, 2001

Oral History

Abstract

Software industry pioneer Kenneth W. Kolence begins by discussing his time as maintenance and operations head on the UNIVAC for the Navy; his work setting up the operations organization and scheduling procedures for the engineering programming efforts at RCA; his tenure at North American Aviation developing process design and instrumentation time; and his joining Control Data Corporation to work on integrated management and design processes, SW product concepts, and performance measurement tool prototyping. Much of the interview concentrates on Kolence's co-founding of K & K Associates, which was soon renamed Boole & Babbage, the first software company in Silicon Valley. Other topics include Boole & Babbage's competition with IBM, and the founding 1968 NATO Software Engineering Conference.

Yost: This is Jeff Yost. I am here today with Ken Kolence at his home in Palo Alto, California. This is an interview conducted under the auspices of the Software History Project of the National Science Foundation. The interview date is October 3, 2001.

University of Illinois

Yost: Ken, could you begin by describing how you first became interested in computing and your experience with computing at the University of Illinois?

Kolence: Well, it was the autumn of 1952 and the beginning of my senior year at the University of Illinois. My major was physics, and thus obviously math as well. As it happened, most of the physics majors in my class were not doing too well. It had to do with the fact that very few of us had gotten a good grounding in theoretical mechanics in our junior year - and if you don't have a good grounding in that you are in trouble in physics. The visiting professor that taught theoretical mechanics that year skipped all of the text chapters for the first semester, saying that they were trivial. Unfortunately I didn't find them trivial, and I wasn't alone. [laughter] But anyway, I think that was why my advisor in physics strongly suggested that I take a new course in computing, which turned out to be about how to do the logical design of computers. It sounded interesting, and I had an open slot in my schedule, so I signed up.

I started the course a week or so after it had started, and the professor (Dr. Jim Robertson) had just began explaining the logic used to design an arithmetic unit. I remember for some reason saying, "You mean to say that computers only do arithmetic?!". Everyone laughed, and that was my real introduction to computing. Fortunately, everything after quickly fell into place and I learned how computers worked both logically and electronically.

The following semester, Dr. Jack Nash, the assistant head of the University of Illinois' Digital Computing Laboratory, taught a new course on programming. It included explaining how instructions worked and what would now be considered as the most basic ways in which they can be put together to in sequences controlled by compare and jump instructions; e.g. subroutines and loops. Also, the Illiac I (a name that came later) had a 48 word or 96 instruction nascent OS (called DOI for Decimal Order Input) that permitted programmers to use a teletype paper tape input device to prepare data and program instructions for the computer. However, most of the time in the course was concerned with managing the growth of computational round-off errors due to the limitation of 40 bit words. Proper factoring of mathematical equations to minimize error generation was stressed, and means of determining the probable error of a result were discussed.

We also got to write and run programs on the Illiac I, and it was this above all that convinced me I wanted to make a career in computing. So I decided to stay at the University of Illinois to get my Master's degree in computing. Since there was no formal means to do this, I got my Masters in Math, with a "Specialty in Digital Computers". Along the way, I learned more Boolean algebra than I ever thought I would need to know. I was wrong: it is essential to any adequate design methodology.

I should also mention another important reason for staying at the U. of I., namely the great team of professors and grad students that were at the Digital Computer Lab: Among them were Drs. Jack Nash, Ken Mueller, Jim Robertson, and John Taub (the head of the lab), as well as Stan Gill and John Wilkes from Cambridge, England.

As an aside, there are two things that Dr Nash said to me that I strongly recall. The first, which led to my deep interest in software design methodologies, was his insistence in the programming course that we were not to use flowcharts. He said that they were inadequate as a program design tool. He was of course right, but he gave us no better alternative. Secondly, I also recall Dr Nash coming to me in the Lab one day while I was in Grad school, and asking me what I wanted to do when I graduated. Of course, I answered that I wanted to be a programmer. His answer was that he wanted to warn me that he felt that only about 7 Illiac or Von Neumann type computers would be needed in the U.S., so the job market might not be too good. Fortunately, I didn’t believe him.

I graduated with my M. S. in Math and a “specialty in computing” in June of 1955, and signed up to join the Navy and go on to active duty at Officer Candidates School in September.

United States Navy

Yost: So your first job out of school was as maintenance and operations head on the UNIVAC for the Navy?

Kolence: That’s right. Actually, a Naval officer getting his PhD in math recruited me for Naval Intelligence, because (he said) that was the only real place where the Navy was using computers. But by the time I got around to joining the Navy in July of ’55, those billets were all filled up and so I became a prospective engineering officer. I was sent to Officer Candidates School and got my commission as an Ensign in late January of 1956. Because of my background in computers, and thanks to some help from the head of the math department at the U of I, I was assigned to the David Taylor Model Basin (now called the Naval Ship Design Center) in a place. called Cabin John, Maryland, on the Potomac River just outside of D. C.

There was a Navy-owned and maintained Univac I at the Model Basin, and I started off as programmer. However, this Univac had become quite unreliable and, especially the tape drives (called Uniservos in those days) were often down. They desperately needed somebody to bring the machine up to a decent level of reliability. Because of my background in physics and computer design, I was assigned to the job of Operations and Maintenance supervisor, with the specific responsibility to bring the Univac up to a satisfactory level of operational reliability and keep it there. At the same time, an enlisted man (whose name I cannot recall) with an EE degree was stationed at the Model Basin. Between the two of us we laid out a plan to bring the system up to a high level of reliability, and laid out a corresponding program of preventative maintenance to keep it at that level. He took care of the Uniservos and their I/O interfaces, and I took care of the mainframe circuitry. Within about three months, the machine was at a quite satisfactory level of reliability. At that point we focused on the preventative maintenance plan needed to keep it there.

With the UNIVAC I, if you were going to operate the computer you also had to know how to fix it as well. So the Operations responsibility was more than simply operating the machine; it was fixing it as well. As I recall, the speed of the Univac I was only about 200 instructions per second. It also had two separate arithmetic and control units operating in parallel. If the result of any operation was different between these two paths, the computer would stop and the operator could reset and restart the computer from the Operator Console. We had several mid-level civil service people who were operator-technicians who could handle such problems. We often had jobs that ran around the clock, and so one or more of these people would be alone on the swing or graveyard shifts. If problems occurred which were beyond their capabilities on these shifts, we had more senior people who were on call on any given night. I was one such person, and I was called out in the middle of the night frequently enough that I had a great incentive to keep reliability high.

This job was the last thing I had envisioned myself doing in the Navy. But as a matter of fact, it was really an important thing for me to learn to do it because first of all, it gave me confidence that I could organize a system for keeping a computer in good shape. It surprised me that it was necessary for me to set it up. After all, when you come out of University you think everyone acts professionally and does things right. But we had a bunch of civil servants who really didn't do anything except patch up the computer. Instead of doing preventative maintenance and making sure it didn't fail, they'd fix it only when it failed. So that was a good thing for me to learn, that you can't do that and succeed with computers. You've got to get them and keep them in good shape all the time. This lesson was well learned, and in my next billet I used what I had learned to build an applications production control system which provided considerable assurance that production runs were correctly run, as well as the information needed to re-run jobs that failed due to bad tapes or related failures.

After 18 months at the Model Basin, I was assigned to the Philadelphia Naval Shipyard in June of 1957 for the second half of my tour of active duty. They were going to install a UNIVAC-II, which was the same as the UNIVAC-I, except it had core memory instead of mercury tubes. They were working on developing a fairly complete set of management and accounting systems for the Shipyard, using a UNIVAC I available in the Franklin Museum. They were writing the applications in the ***B Zero*** language and compiler, which was under development by Dr. Grace Hopper at the Univac headquarters in Philadelphia. "B Zero", was one of the two predecessors commercial application languages from which the COBOL language and compiler structures were synthesized. IBM's Flowmatic language/compiler was the other predecessor used in the development of the COmmon Business Oriented Language and its associated compilers.

My assignment was to develop and implement the application workload production planning, scheduling management, and failure recovery procedures, and to train the existing civil servant personnel in their use. There was no operating system available on the UNIVAC I or II, and as a result the tape drives each sub-program of the applications to be run had to be “hard-coded” tape drive numbers to be used to read and write the necessary files. We had 8 drives available, so we would have to mount and dismount the tapes used and/or produced for each job. Setup work could then easily generate a large amount of setup time, reducing the daily capacity of the computer to complete the required applications. Likewise, we had to organize the use of the reels of tape to assure that we could recover from any tape failures, primarily by rerunning the jobs needed to produce a new tape.

The first thing we did was to lay out charts that showed the files that were needed and how we could place them on tape drives so that they could be taken down after each job was completed and while the next job was running. We did this for all jobs and then modified the application’s job code to specify the desired tape drives to be used for each job’s files to minimize or eliminate the down time within the application due to tape mounting and dismounting. When that was finished, we identified which files were needed to reproduce the various master and update files. Once done we were then able to allocate specific sets of reels of tape to each job file in a way that automatically gave us the ID of reels needed to reproduce a tape that had physically or logically failed in some way. The physical tapes used to provide backup generation capability were then appropriately labeled, and placed in tape storage cabinets whose ID’s were then included on the “production control charts” for each application and system of applications.

In practice, these production control charts and those produced later for other applications were successfully used for a number of years.

The issue of how efficiently the UNIVAC II was being used was addressed by taking three electric clocks with digital readouts and connecting them to the UNIVAC II (when it finally was installed) in a way that gave us a readout of (1) the total time the UNIVAC II central processor was executing AND/OR reading/writing data during each shift, and (2) the time the UNIVAC II was idle and tape drives were being mounted. The third clock gave us the time for each shift. Given this data, one could then identify the percentage of time the UNIVAC II applications were executing, and the time spent in a wait mode. This data was used to project the amount of data that could be processed within a shift for the various applications

We did a pretty good job of organizing and implementing the operation of the UNIVAC II. At least we received some very nice complements from the various Naval personnel who came through and discussed how we managed the system.

Finally, in late 1958, I completed my active duty commitment. I placed on inactive duty right before Christmas in 1958, and went to work locally for RCA Defense Electronic Products in Moorestown, New Jersey.

Yost: What did you do at RCA and what was RCA building?

RCA Defense Electronic Systems - BMEWS

Kolence: RCA DEP was designing and building the first component of the Ballistic Missile Early Warning System, better known as BMEWS, at Thule, Greenland. The system in Thule included a transistorized computer built by IBM specifically for BMEWS, called the 709TX, and was logically equivalent to an IBM 709 computer (which was a tube machine). We had a 709 as part of our system. The complete system in Moorestown was originally meant to provide a tracking radar capability for the Thule system, but this capability was replaced by an electronic tracking capability rather than the huge mechanical tracking radar built for use in Moorestown. The Moorestown tracking system was then used primarily for use in tracking rockets launched from Cape Canaveral.

When I started work at RCA, they had just received an IBM 709. My job at RCA was similar to what I had done at the Shipyard, and it was my responsibility to set up the Operations Organization and scheduling procedures for the engineering programming efforts. Considerable software had to be written and tested for the Thule BMEWS system, and then integrated at that site. In addition, when there were launches from the Cape, the tracking radar was used to track the rocket, and subsequently considerable data reduction processing was performed. My job relative to this work was simply to provide the access to the computer needed by the people tracking the launch vehicles. Typically, time for the data reduction runs was provided in the evening shifts, as was time for production runs associated with engineering work. The daytime shifts were primarily used for the engineering software development and testing activities.

The IBM 709 had an operating system (called IB/SYS as I recall) that provided excellent support for testing activities such as getting memory dumps with instructions and data properly expressed in decimal and alpha form, as appropriate. That is to say, dumps did not require people to translate them to from hexidecimal forms. Also, programmers could insert checkpoints in their code and extract various types of information from executing code. Both Fortran and assembly language code was supported, but assembly code more so than the Fortran compiler.

The IB/SYS operating system also had many other features to manage production runs and that simplified the scheduling and distribution of results so I was able to fairly quickly set a smoothly running Operations group. However, IB/SYS itself was not easy to set up and manage and it was being updated fairly often, so I had to set up what would be called a Systems Programming group today. Also, IB/SYS had an output log tape or provided a log using punch cards (I can't remember which), but no data reduction program. So I wrote an assembly language log analysis program that provided us with the ability to track usage and allocate charges to the appropriate engineering projects as well as the data reduction work associated with missile tracking.

Once this was done I found myself with some spare time, especially when the 709 was running reliably. Having just written the log analysis program, I had remembered Dr John Nash's strong statements on the inadequacy of flowcharting as a design notation. I agreed with him at this point, and decided to see if I could find a more satisfactory notation to represent the design of software. For several months, on and off, I tried to represent some small pieces of software using Boolean algebra logical equations in the same way that one used them to represent hardware design logic. After a while two things became clear to me: (1) defining the appropriate Boolean variables at various levels of logic quickly became much too confusing for normal use, and so (2) some other notational form that had much better visual decomposition properties and yet defined Boolean states appropriately was needed. I had no idea what to do next, so I put the whole idea away for another day. (I returned to it several different times over the years though)

Sometime during all these activities I read an article in “Factory Magazine” that showed how standard sampling theory could provide good information on how much time was spent doing various job activities such as operating a machine, getting tools, bringing materials to the machine, etc. I immediately saw that one could use the same method to determine the various places where a program was spending its time, and thus where its efficiency could be improved. Similarly, one could determine which subsystems were bottlenecking the overall system. But one would need some means of randomly interrupting the system or program and collecting the information on what was active at the instant of interruption, or what was the last instruction executed. There were no “timer interrupts” on the 709, so I filed this observation away also until I was able to find a way to randomly sample an executing program or configuration.

Finally, I should also mention that during this time I was able to attend several meetings of the IBM scientific users group called SHARE.   They were my first contact with other professionals using IBM computers, and I became quite active in the group.   In particular, I became good friends with Jack Strong from North American Aviation, Los Angeles. Jack was one of the founders of SHARE and also GUIDE - the equivalent organization for commercially oriented installations. One of the SHARE meetings was in Los Angeles, and since I could see the BMEWS contract was coming to an end, I asked him what the possibilities might be for a job with North American. He said the chances were very good, and to let him know when I would be available.

Several months later when BMEWS was completed and I felt free to leave, I contacted Jack and he offered me a job at the aircraft division of North American Aviation. The formal name of this division was the Los Angeles Division, and was located at the Los Angeles Airport. I accepted the job, packed up my family, and left for Los Angeles on New Years Day, 1961.

Yost: Was the job much the same type of work?

North American Aviation – Development Process Design & Instrumentation

Kolence: Well, the answer to that is both yes and no, because of the different types of responsibilities that I had during my tenure at the LA Division of North American. For example, I was initially assigned to develop a plan for introducing the new COBOL language and compiler into the Commercial Applications development and operations organizations. I worked on this in my first few months on the job. This was certainly different from any other work I had done.

However, just as I was starting to understand what the overall effort involved was, the manager of the programming development group left for a better job and I was offered that position. Naturally, I accepted it. At this point my responsibility became to assure that the new applications were reliably produced and delivered on schedule, and similarly that existing production software was maintained properly when needed. In essence, these were the same kinds of responsibility that I had had in the Navy and at RCA, but now what was needed was a well-defined and consistent application development process. So although the subject matter was different, it was still the same kind of work that I had been doing since leaving the U of I.

On top of all this, I had become the chairman of the SHARE "SHAREmanship" committee when Jack Strong had resigned that post. This committee had two major responsibilities: (1) to greet new SHARE members and help them get involved in SHARE activities, and (2) to help the membership to address new areas of interest. Effectively, this latter responsibility meant that I had to establish SHAREmanship subcommittees for a number of topics that were not otherwise being addressed. This kind of involvement with a professional society was quite new to me.

Before going somewhat into the details of what I did to meet build an organization that consistently met its schedule and development responsibilities, I need to describe what computers were used by Commercial Applications, and the major sub-organizations of the Department. We shared the use of two IBM 7090's and later 7094's with the Engineering Computing Department, but primarily we used IBM 708's and 7080's for our major systems, and IBM 1401's and 1410's (later called 7010's) for all new applications. The 1401's were initially used to convert punch card based systems to computers, and then to interface with the 7080 based software systems. The 1401's were small systems, and we had 10 or 12 of them. Typically, each system consisted of a CPU, card reader/punch, tape drive(s), and a fast "chain" printer. The 1410/7010 hardware came in after I had been promoted to General Supervisor. We used an assembly language named Autocoder for the 1401 and an extension of it for the 1410/7010 systems. There was no reliable IBM COBOL compiler in 1961, but FORTRAN was available on the 7090/94's.

Organizationally we had three major groups: Systems Development (and Definition), Applications Development (and Maintenance), and Operations (including Production Operations). The Systems people were responsible for the defining the logic, data structures, and structure of the proposed system and its interfaces. Applications was responsible for translating these into code and programs and/or subprograms, and testing the resultant system and eliminating bugs. Applications also had the responsibility to update existing systems and debugging them if they failed. Applications people ran the initial production runs, and Production Operations worked with them until such time as the new system was determined to be stable. At that point, the Applications people turned the application over to the Production operations people.

This sounds like a very reasonable organization today, and in fact overall it was a good one in 1961. The problem then was that there was very little in the way of formal procedures and standard methods of defining systems.   Each person – be it a Systems analyst or a Programmer – had their own variations in the ways they would describe their parts of a system. This led to difficulties in assuring the system was adequately defined and tested and/or interfaced with pre-existing systems. Also, the concept of formal requirements was very vague in the early 1960's, and the ideas of data structures, catalogs, and databases in general were not yet formalized in any useful way.

At any rate, I talked with my lead people (Charlie Dawes and Jen Brian) about how we could raise our productivity, standardize our activities, and be consistent in setting and meeting our scheduled system initial production dates. Also, we wanted to interface better with the production operations group, and standardize the documentation that was needed by that group to take over new application production, including such things as restart and file recovery problems. We identified the biggest problems with schedule development and adherence to include the inconsistency of the system definitions provided us by the Systems Development people, the lack of a consistent program development process, a lack of effective programmer training (experienced programmers were very hard to find in those days), and the difficulties of keeping new development schedules on time when key people were always being pulled off to take care of problems or needed changes in existing systems. It became clear that we had to get our act together and develop an organizational structure and the associated procedures to address all of these issues. I should note here that the LA Division of NAA had a contract to develop the B70 supersonic bomber, and thus the need and money for expanding and formalizing the responsibilities and processes of the DP organizations was available.

The first thing we decided to do was to split the Applications Development organization into two major sections that were to address (1) the maintenance needs of the old (708/80 and 7090) systems and the newer 1401 and eventually 1410/7010 systems and (2) the new systems development and implementation activities. We also identified the need for a small staff function that would develop the operational documentation needed by the Production Operations group to take over new production systems. This would guarantee consistent and complete documentation was provided and/or updated for new applications and program maintenance activities. Another person (Harris Weaver) was subsequently hired to be the main interface between Applications Development and Operations to develop and provide appropriate procedures and training for Operations personnel. He officially was part of Applications Development, but also provided many staff services to the Operations organization.

The equivalent formalization of the interfacing functions between Applications and Systems Development was deferred until we had actually implemented the above capabilities. In fact however, we were never to formalize this interface since today's ideas on requirements analysis, definition, and software architectures were not yet understood.

Charlie Dawes took the job of Maintenance Supervisor, and Jen Brian became the New Applications Supervisor. I can't remember the last name of the staff lead who managed the documentation group but I think his first name was Jim and had been a production operations person. Some months later I added a staff operations interface person named Harris Weaver, who also had an excellent background in computer operations.

The Maintenance group was formed primarily to minimize the disruption of new application development activity, and thus remove an important "root cause" of the development schedule slippage. In fact, a quite significant reduction in new development schedule slippage was observed within a few months of the full implementation of a separate Maintenance group.

Yost: So the division of labor added greatly to the efficiency?

Kolence: Absolutely, and this improved the response of the Maintenance group immediately. There were 3 or 4 people in Maintenance besides Charlie who had responsibilities for different operational 708/7080 applications systems. Since no new 708/7080 systems were being developed at that time, this was enough people to take care of all maintenance on these systems. Other personnel who had been doing 708/7080 systems maintenance were thus freed up to work in Jen's group on new 1401 and 1410 systems, which were intended to replace the old systems at some point in time. This was because the 1410's we were getting had IBM's first random access disks, with which we intended to avoid the long tape file updates that existed with the old 708/7080 systems.

However, even more people were needed for new 1401and 1410 systems, and we often had to hire less experienced programmers and give them additional on-the-job training. I recognized that maintenance responsibilities could also provide an excellent environment for new 1401/1410 programmers to learn how more experienced programmers coded various parts of applications functionality. So new hires with minimum training were initially assigned to Maintenance to work on 1401 systems.

Now 1401's were sort of like a UNIVAC 1 in terms of speed – in other words, slow - and had a small console to manually control program execution. So recalling my work at RCA, I recognized that I could finally collect data on where programs spent time in code execution by randomly manually stopping execution of a 1401 program and recording the location it stopped at. We could then build a histogram of these code locations, and reduce the overall time of execution by re-coding those portions of the program taking the most time. Thus, the knowledge that new 1401 programmers received by doing maintenance work would also give them insight into how to write more efficient code. I tested this idea, and it proved completely correct.

As a result, a standard procedure and analysis forms for 1401 maintenance people were defined as a part of any maintenance being done. It required them to randomly stop program execution 100 times and record the addresses of instruction so observed. Then, using a histogram of the data tied to the memory locations used, they had to correlate these addresses to a listing of the program and determine if and where performance improvement could be easily obtained by re-coding certain portions of the program. If so, and with Charlie's OK, then the code would be rewritten for performance purposes.

About a three or four month stint in Maintenance resulted in programmers who, if they desired, could be used in the New Applications group. However some preferred to remain as maintenance programmers.

Once the Maintenance group was operating smoothly, Jen Brian's New Applications programming group was able to avoid most serious application completion schedule delays. We also were able to increase productivity and standardize our system production documentation by defining the standard information to be provided to the Production Operations group for each new application and its interfaces to existing applications. "Jim" set up these standards in conjunction with Harris Weaver and Jen Brian. Obviously, I also contributed some ideas based on my operational experiences at the Philadelphia Shipyard and RCA.

To an outside observer like my boss it probably appeared that we now had a well defined and stable process for developing new applications and maintaining operational systems. He certainly complemented us highly on how smoothly everything was working under the new organization. However this was not completely the case because the Systems Development group did not have a consistent approach to defining and/or describing new systems. Unfortunately, the person who headed the Systems group was not at all interested in systemizing his own organization, and his people were permitted to prepare system definition documents in whatever form they felt was most appropriate.

Fortunately, Jen had come from a similar environment and was usually able to work with the material provided. In those days, the core architecture of almost every commercial program was essentially some variant of a sequential tape driven processing model, so Jen was able to translate the specs and processing rules in whatever form they were provided. He then gave this information to the specific programmer or programmer team, and was able to guide them in their development activities. Similarly, he was able to develop and use a basic plan template to estimate the corresponding programming and testing time requirements. As a result, his actual system completion and production qualification dates were normally very close to his scheduled dates.

I should mention here that David Katch, who was to become the co-founder of Boole & Babbage, joined the Systems group about this time and he, Jen, and I worked together on the problem of defining some standard forms of systems definition for the Systems group. Although we could not come up with any completely satisfactory forms of definition, we did get the Systems people to at least discuss with us (Applications Development) in advance what information we needed and what forms of descriptions would be reasonable for each new system. Our major requirement was simply stated as: "Tell us what you want, but not how to do it." As a result of these discussions, we were no longer completely surprised by the forms in which new system definitions were provided to us.

At the beginning of my second year at North American, the Applications Development group was operating in a reliable and productive manner. There were always new things that came up, but they were generally easy to handle.

For example, the IBM 1410 arrived, and we found that its original operating system did not adequately support the management of our planned production workload. We had no choice but to significantly extend its capabilities, which we did. I hired two people from RCA Moorestown (BMEWS) who were part of their System Programming group, and they easily added the necessary abilities to the IBM system. They thus became our Systems Programming capability.

My workload was reasonably light, and so with the permission of my boss I accepted the job of being the 1962 SHARE Program Chairman. SHARE held meetings every six months, so my commitment was to organize and put on two SHARE meetings.

Since I had earlier become the SHAREmanship chairman, I was able to add a number of new topics to the normal SHARE agenda. For example, SHARE did not have a committee to work with IBM to extend the FORTRAN language and its compiler capabilities. Several people felt this was necessary, so I set them up as a SHAREmanship subcommittee and gave them time on the Program. Likewise, there were individuals who were interested in things like Systems Documentation, Simulation, PERT/CPM products, and various other management concerns. They all got slots on the Program, and the SHARE Spring meeting was a great success. Attendance was up, and many other SHAREmanship committees were formed for the Fall meeting. I was nominated for and elected that Fall to the SHARE vice-president's position, and had responsibility for guiding a new SHARE division concerned with management issues.

Even with all of my work and SHARE program responsibilities, and the excellent performance of the New Applications development people, I still found myself dissatisfied with the flowcharting methodology we used. The basic problems I had with the methodology were that:

(1) Detailed or low level flowcharts could rarely be mapped directly to higher level flowcharts, and vice versa.

(2) Operations on individual datum values and on values of more complex data structures were not easily explicitly described as a whole, and

(3) Any given flowchart was not a unique representation of the process to be performed. That is, any given flowchart for a specified process can only be one of a set of flowcharts, each of which will produce the same transformations. However, the individual flowcharts in the set,

when translated to code, would seldom use the same amount of memory, resources, or result in the same overall completion time.

In other words, flowchart notation cannot be used as either a top-down design description or a bottom-up design description.. It also cannot be used to manage the operational properties of the resultant code. Taken together, flowcharting is obviously an inadequate notation for Software Engineering[1]. My old graduate advisor, Dr Jack Nash at the University of Illinois, was right! Flowcharts are not a good a good design notation for software.

I must admit that back in 1962 or ’63, I was not able to so neatly wrap up the problems of flowcharting notation, but at least I fully understood that one could not use them in a top-down design process. And I further understood that this fact easily led to bugs and unexpected performance results in actual code. Finally, I must also admit that I still felt that there had to be a design notation completely compatible with the methods of computer logic design.

Fortunately, in that same time frame, a notation called “Decision Tables” came into use, and it basically had many of the properties needed to provide a top-down design capability. However, not all of them, so Decision Tables faded from use after several years. (They recently have returned in a different context or design paradigm, but they are now called “use cases”.) Probably the main reason for the loss of interest in the pure Decision Table form of the time (the

[1] Sometime in 1962, I was sent to Edwards Air Force Base for some reason. But while I was there, I watched an aircraft take off for the very first time. Two thoughts popped up unbidden in my mind: “My God, that pilot is trusting his life to the aeronautical engineers who designed that plane” and then “I’d never trust my life to software

early mid-1960's) was the fact that data structures were not well understood or used at that time, as data bases just coming into vogue.

An hierarchal view of the structure of data files and data bases to be used in a software design, along with certain changes to the normal DT or Use Case (UC) forms to identify the boundaries of each type and level of a data structure, is necessary to provide a notational capability to develop a full top-down and bottom-up software equivalent of a typical hardware engineering drawing tree. Such a capability, I believe, is one of the essential prerequisites to a viable software engineering discipline. (Historically, development of this type of decomposition/re-composition characteristics for drawings of machines was one of the more important enabling capabilities that led to the first engineering discipline, namely mechanical engineering.) Structured programming notation shows when a *process* step moves down to or comes up from a detailed level, but does not provide an overall top-down or bottom-up picture of the structure of the logic at each level of data structure. In general, any given software description given in sequential process form represents only one of many possible factorings of top-down logic into a sequential process form.

I spent quite a bit of time at North American trying to develop a usable methodology of system and program definition based on my versions of decision table and data structure descriptions. I gave several talks on my ideas at local ACM meetings, but the ideas were so different from the existing "sequential process" paradigm of programming that most people found them

today – we need a software engineering discipline." From that point on, my goal was to find ways to contribute to developing such a discipline.

meaningless. Finally, I realized that I needed to somehow develop a full example of my ideas by producing a complete set of system and programming definitions, the actual code produced, and the means by which to generate a comprehensive testing capability. I had no idea how I could find some way of doing that.

It was late in 1963 when I came to the unhappy conclusion above, and I was feeling pretty negative for other reasons as well. The XB-70 bomber was the main contract at the Los Angeles Division of North American Aviation, and in the late summer or early autumn the government had decided to cancel the production of that plane. The result in Applications Development was a severe layoff, and I had been forced to terminate a third or more of my staff. Since completing the RCA BMEWS project had also led to layoffs, I began to see that this would probably be a pattern if I stayed working for companies that depended on military equipment contracts. I detested the idea that I might have to go through cycles of build-up and tear-down of my organization, and wanted to find a more stable business field to work in.

After giving some thought to my options I decided that I wanted to get a job that might permit me to continue working on my software design process, and also I figured working for a manufacturer would be a good thing for my career. I also decided that the best companies to work for were IBM and another company called Control Data Corporation, or CDC.   I knew I could easily get a job with IBM, because I was the SHARE Vice President and generally IBM gave people like me a good job if they asked for one. I called a friend high up in IBM and he said there was a good chance for work, especially with him in Poughkeepsie, New York. But I had grown to very much to like California, although I was having some bad eye problems from the smog in the Los Angeles basin. So, I also called CDC whose headquarters for software development was in the Stanford Industrial Park next to Palo Alto (near San Francisco). I spoke with my SHARE friend Donn Parker. Donn recently had gone to work for CDC, and he too said there was a very good chance I could get a job there. I felt a lot better after these calls, but I still had to take care of some work at North American, and I didn't know what kind of jobs I might be offered by either IBM or CDC. At any rate, I decided to wait a few months until after Christmas (1964) before actively looking for a job.

Yost: How did you make your decision about whom to work for?

Kolence: Well, the more I thought about it, the less I liked about going back to cold winters, and Poughkeepsie was definitely such a place. So I figured I should talk with CDC first and see what kind of work they would offer me. I drove up to Palo Alto in late January 1964 and, since I had gotten there early, just drove around the town. I have no idea why, but after a while I began to feel there was something very special about Palo Alto. And suddenly I said to myself: "I'm home!" I knew it was where I wanted to live, and sure enough this is where I'm still living. Obviously it was an emotional decision, not a financial one, but it was still the right one for me and my family. And that was well before Palo Alto and the Santa Clara Valley got the name Silicon Valley.

Anyway, I went to my job interview and I was offered a job. Dr. Clair Miller (whose formal title was Chief Engineer) was the general manager of the facility, and he wanted me to work on improving the capability of his organization to deliver CDC systems software and its documentation on schedule and bug free. Somehow I also was given the impression that he thought producing a sample software product to demonstrate such a methodology would be a good way to convince his people to follow it for standard CDC systems software. Happy as a clam, I accepted the job. Later, I found out that Donn Parker had told Clair about my work in SHARE, knew of my NAA work, and convinced Clair that I could solve his software development management control problems. I went back to Los Angeles, turned in my resignation, put the house up for sale, and on March 1, 1964 went up to Palo Alto to find a place to rent. And here I am in the year 2001, thirty-seven years later.

Yost: It's a great place. But before we talk about your work at Control Data, was there any interest in SHARE in your measurement activities or the rest of your work?

Measurement? I probably never mentioned it at SHARE, because SHARE was really only concerned with using the IBM scientific machines, i.e., 7090's and 7094's. SHARE people considered 1401 systems as printers rather than "real computer systems". I felt I needed to implement the idea on a computer with a timer interrupt to really prove the value of the system. In fact one of the other NAA divisions, Autonetics, did have a 7094 with a timer interrupt. But IBM timer interrupts were special-order features, and were very expensive ($1,000 per month) in the money of the day. Autonetics did use the approach to solve a performance problem with one of their major production systems, but thereafter they never bothered with the idea again. Nor did any other NAA Divisions. As a matter of fact the same thing happened at Control Data, but I'll get into that later.

As far as my design management work was concerned, I remember giving some talks at SHARE on decision tables and data structures. But as I said earlier, people just didn't really understand why I felt it was an important approach. As a matter of fact, I gave up on discussing our development methodology at SHARE, except privately if someone asked me about it. Remember, SHARE was interested in engineering and scientific computing activities, and I was involved with commercial systems.

Yost: When did you start at the Control Data Corporation, and what areas did you focus on in your early years with the firm?

Control Data Time – Integrated Management & Design Processes, SW Product Concepts, and Performance Measurement Tool Prototyping.

Kolence: I started work at CDC in early March 1964, and was assigned to work for Dr Dick Zemlin. Dick was Clair Miller's right hand man, and I was led to understand that I reported to Dick mainly for Clair to avoid having too many people report directly to him. In particular, he told me to feel free to talk with Clair without getting his OK.

At any rate, Dick mentioned to me that there were some organizational changes coming up in the systems development activities, and asked me to review the existing development procedures and let him know what I might suggest to improve them. He also mentioned that they were thinking of developing a data base query system, since they didn't have one, and I was welcome to prepare a proposal for it. He mentioned that he had made the same offer to Dr Ted Olle, and he would pick which one to develop. That sounded fine to me, because I thought I could design such a system based on my decision table and data structures work. However before working on it, I decided I should meet with the various CDC people individually and find out what the existing development process was and how well it worked. The date for delivering the query system proposal was 3 months away, and I figured I had plenty of time to get it done.

Dick introduced me around the organization, and I met several people who had studied at the U of I. However, except for one person, they had all been there after I had left. At any rate, the more people I talked to and found out how their development process was structured, the more depressed I became about my decision to work at CDC. CDC's approach to producing software for their computer was undocumented, unmanageable, and causing their customers fits.

I began regretting that I had joined North American, and especially so when IBM announced their new System 360 line in late May or early June. I read what software IBM was going to provide for the different 360 models and my heart sunk – I remember thinking that CDC could not have even conceived of such a system, and no way could they possibly build anything like it with their current processes. I came very close indeed to calling up my ex-boss at North American and asking if I could have my old job back.

But I really liked the Bay Area, so I decided I could help CDC a lot. I went and told Clair and Dick what I thought of the existing ways in which they built software, and that a much more structured and reliable process had to be used if CDC was to compete with IBM. A month or two later, they asked me to develop and operate (in a staff sense) such a system. I agreed, and they made me Manager of the Analysis Department[2]. They also gave me a few people to help and some additional open positions to fill as required.

Well, I accepted the challenge and never did personally get a chance to build a sample software application using my decision table and data structure design methodology. However, following my usual pattern of work, I eventually did get a chance to write up a reasonable explanation with examples of my methodology.

Before going into what I did to provide CDC with a viable and reliable design and development methodology, I need to first give a quick summary of the computer lines that CDC had at this time, and then describe the scope of software that had to be produced and released for these lines.

CDC was strongly competing with IBM with their 6000 series computers at sites that had large scientific computation workloads. If I recall correctly, there were ultimately 3 systems in that series, the 6400, 6600, and the 6800. I believe though that only the 6600 was available at that time, and we had a fully loaded 6600 system in our Operations group. The 6600 architecture

[2] In addition to acting as the staff function that operated the software development management system we

was really oriented toward separate computation in a main CPU, and several supporting I/O data and OS management stream. It had a central computer, the 6604 with 64K of 64-bit word memory, and 10 small “peripheral computers” with a total of an additional 64 K bits of a smaller word size – probably 24 bits. One of these peripheral processors contained the operating system that controlled the main 6604 processor. In addition, it had several tape drives, a large disk system, a printer, and an operator console with a card reader and punch. In effect, it was a small multiprocessor network and the 6600 processor itself could be kept in problem execution for close to 100% of the time. This was why it was so popular with high mainframe computation workload sites.

In addition, CDC had a smaller and more conventional architectured series of computers, all of which had the same basic instruction set. The 3100, 3200, 34 00, and 3600 systems shared a more conventional architecture, using tape drives, IBM 1311 small disk drives, and a console with card and printer attachments. These machines could operate in a stand-alone mode, or be interconnected to one or more other CDC systems. Some of these lower-end systems were sold as IBM 1401 systems printer systems, but all of them had a timer interrupt which we eventually used for sampling measurement purposes.

CDC provided the following types of software for the computers: Operating Systems, Assembly Language Systems, including debugging tools and other utilities, FORTRAN compilers, and Application Systems such as PERT and CPM (Construction Project Management, a PERT-like application with capabilities similar to PERT), and other special applications useful to different

developed, the Analysis Department also acted much as a Software Process Improvement group does today.

classes of customers. Different operating systems were provided for the different types of computer systems, and updated and extended OS versions were released periodically. Mostly, updates and extensions to the language compilers and assemblers were also released with the Operating Systems. Other Application code updates might or might not be released concurrently with the OS systems for each computer line. In any event, bug fixes for these software systems were provided in various ways, some formal and others not so formal.

In addition to these development activities, there were groups that were responsible for preparing manuals for these software systems, for assuring the systems were operationally reliable and upward compatible, and so forth.

As I began to think through the needs of an overall software management system for CDC, it became clear to me that there were a number of conflicting requirements that needed to be resolved in some manner.

My first problem was to decide to how to consistently name each major piece of software and its various parts; e.g., code, manuals, revision numbers, etc. Simple enough I thought, we'll just assign a name, "product number" and "part number", along with a version number, to each functional software item. We will then refer to the overall complete package of software and all of its parts in various forms as "the software product".   That meant that the development process had to designed to provide for each of the product parts, and do so in a timely manner.

Eventually, but far from right away, that was how the term software product became accepted within CDC.

Yost: Now this was very early on that you had an idea of software as a product,. I understand you ran into great resistance within the company and in the larger community. Is that correct?

Kolence: Well, yes within Control Data, but remember this was 1964 and a lot of people, including most of the CDC people, considered the code to be the only thing they were involved with. So they saw no reason to go through the complicated management process I was proposing to coordinate the production and "bringing together" of all of the parts our customers needed. In other words, the developers did not want to be subjected to the discipline that the concept of a product imposes. Fortunately for me, Clair Miller, the Chief Engineer, was fed up with getting in trouble because software was never ready to be released when it had been promised. At any rate, all of Clair's Directors got together and complained to mightily. My understanding of what they said was something like this: "Ken may be right about the need for better management controls, but what he wants to do is too complicated. Either get rid of Ken or we're all going to quit." But Clair called their bluff, and told them, "OK, turn in your resignations." They were pretty chastised, and after that they became more cooperative and even started writing some procedures to put in the Programming Handbook.

The next issue I had to consider as part of the product concept was basically how to obtain approval (and from whom) to initiate efforts to develop a software product or an enhanced version of an existing product. Also, what specification, test, code, and related documents and material were to be produced? Finally, who would be responsible for producing what, what would the costs or man months of effort be, what reviews and reports on progress would be provided, by whom and when or at what activity completion points. In other words, what did we need to know about the product functionality and the process of producing the product material, and what documents and/or events would trigger reviews and approvals to continue the efforts?

Beyond the scope of a single software product, there was the concept of a “release”, which would usually constitute a suite of software products that worked together and all of which contained some new functionality. The delivery date of a release required that all of the constituent products completion dates fall very closely together. So by what means could we determine whether or not a problem affecting the product/release package had developed, and if so, how could we overcome the problem? Finally, what material would be provided with a release to aid the recipient in installing and bringing up the parts of each release package?

It was also clear to me that the software development process to be defined had to be applicable to all software products that were developed, regardless of functionality, size, or any other parameter. The reason for this process requirement was that the points in the process that provided management information as well as the type of information actually provided had to be the same for every product if management were to be able to manage the overall process in a consistent manner. Once established, this process could then be evolved and improved as experience in its use was gathered.

There were a number of other problems that had to be resolved, the most important being how to provide management with timely information that permitted them to identify problems at all levels of the general development process in a way that also helped identify the causes of the problem and effective ways to resolve the problem. There was no single way to provide all of the desired management capabilities, but the CDC PERT software could provide a consistent method of (1) providing management with a realistic overview of individual products an release product sets under development, and (2) identifying the existence of schedule problems early enough to be able resolve or otherwise minimize them. So I decided to use PERT as the primary means of collecting, organizing, and presenting management with program status information.

Dick Zemlin, my direct boss, had sketched out a very high level diagram showing the stages of development of CDC software "products" (though the term product was not used by him) and I used this as the starting point of the required PERT diagram. Many things needed to be defined and agreed upon as to what these intermediate stages consisted of, but the first thing to do was to obtain an agreement with the development organizations as to what aspects of a software product required top management approval, and which aspects could be decided by the Director and/or the managers of the organizations doing the coding. We needed this definition to specify the contents of various documents to be approved by management or its representatives, and to indicate what types of information should be omitted. In other words, we needed to define who was responsible for what in software product development.

My basic solution to this fundamental requirement was to state that the "externally observable characteristics" of each software product required management approval and, once approved, could not be changed without management approval. The "internal design" features of a software product were the responsibility of the management of the developing organization. “External” meant “those features of a software product which are observable or under the active control of a user”; while the “internal design” was made up of the machine code, table structures, algorithms, etc. which would jointly implement the external characteristics. In essence, I was simply saying more precisely that management controlled what the product did, but that management left the implementing decisions up to the developing organization. (In other words, “Tell us what to do, but not how to do it”.) Later, we refined this concept by establishing standards to be followed or used, but as described above the idea turned out to be workable and acceptable to both the developers and to senior levels of CDC management.

Given the acceptance of this division of approval authority, I proceeded to define the collection and contents of the various documents to be produced in the course of the development of a given software product. Software product development was initiated using a “Software Initiation Request’ or SIR. It defined the elements of the software product to be provided, e.g., software, concise summaries of the product’s external design objectives and functionality to be provided, the exact design documents to be developed, responsibilities, costs, schedules, responsibilities and control dates. These latter were dates defined at each design stage approval, and used in the PERT management control systems to identify problems in meeting the products needed in the subsequent development activities. For example, product User Manuals were derived from the design document called the External Reference Specifications (ERS), as were the Test Cases used to assure the coded product corresponded to the ERS documentation.

The SIR document also defined the composition of a Design Review Board (DRB) for each software product. That Board was responsible for assuring the External Reference Specifications for the product met the product's Design Objectives and represented a coherent description of the proposed product capabilities.

Approval of the SIR and subsequent functional design documents was done in a monthly meeting of the Management Approval Board (MAB). The MAB was chaired by the Chief Engineer, Clair Miller and made up of the various Directors and those managers submitting material for approval. The manager of the Analysis department (me) acted as the secretary of the meeting, prepared and pre-distributed the agenda and any documents to be considered for approval, and the Analysis department's comments and recommendations on the agenda items along with the responses to those comments by the submitters if needed. Each item on the agenda was discussed and, typically, approved. Occasionally however the item under discussion was returned for additional or revised work, and even more rarely, not approved. This is not to imply that this Board was just a rubber stamp meeting. Rather, it was because careful attention was paid by all participants to assure all problems or objections were resolved before submitting material for approval.

There was a second Board composed of the same people, which also met on a monthly basis. This was the Management Review Board or MRB. There were two major functions of this Board. The first was to review the PERT reports for each Product under development, at both the overall and component levels, e.g., was the planned release date of the product still projected to be met, and were all items needed for the release (such as user manuals) on schedule. If not, what actions could and would be taken to bring the PERT projections back to the desired schedule dates. Once again, all participants worked hard to make this meeting a positive and productive meeting. Within a surprisingly short time, the PERT information led to a much higher degree of coordination than Clair Miller's organization had ever had.

The second purpose of this Board was to review the status of commitments that had been made by the Applications Development organization to various Users and other CDC organizations. A "commitment list" was prepared, and over a several month period nearly all of the original items on it had been taken care of in appropriate ways.

The complete set of procedures discussed above were gathered together in a "Programming Handbook" and released for review within the organization in early November 1964. They were formally approved and phased into practice starting at the beginning of 1965[3]. Over time, the handbook material was extended, streamlined, and upgraded. It became a very unifying document, with all groups within the Software Applications organizations contributing and unifying material of their own.

By the end of 1965, and thereafter until the CDC Software Applications organization was moved to Minnesota, we were meeting over 95% of our scheduled software deliveries.

Yost: That is quite an amazing statistic, especially for that time frame. But what happened to your interest in measurement and your Decision Table design methodology. Did you get a chance to work on them?

Kolence: Yes. As far as my measurement ideas, right after I became manager of the Analysis group and, as I mentioned, I had two people assigned to work for me. Don Lytle was one of them, and I can't recall the name of the other person. I knew that the 3000 family of CDC computers had a timer interrupt and I was eager to resolve several questions about my idea, especially those related to the requirement that the data collection had to be done in a random manner. For one thing, I wanted to satisfy myself that the sampling process itself would give the correct data under a variety of different program states. For example, I wanted to make sure that the possibility of synchronization of some program execution loops with the sampling code could be prevented. Another thing was to figure out how stable the measurement results would be over a wide range of sample quantities.

At any rate, I explained to Don what I wanted to do, and asked him to build the software so that we could generate a random number, convert it to a time interval, and use it to interrupt an executing program and extract the location at which the computer stopped when it was interrupted. Also, I wanted to be able to vary the number of such samples gathered to see what a

[3] A control and distribution system, called Document Control, had already been established as part of Analysis to reproduce and distribute all the documents to be produced and distributed in accordance with the Handbook procedures. Document Control also managed the process of making PERT runs and producing reports as requested.

reasonable number for sampling a program might be. Finally, I wanted to see how much, if any, extra run time was added to a program being sampled by our program.

At any rate, Don and the other person went ahead and built the software. After a month or so, Don came to me and was ecstatic about how well the sampling program identified just where a target program was spending its time. We then laid out a plan to make a careful study of the questions I had about the sampling processes. Eventually we found that the results were stable over a wide range of total observations, and that the sampling rate did not need to be randomized because the chance of being synchronized with the program was quite remote.

We let people know the measurement program was available and that we would help them to use it. We got absolutely no requests to use the program. However, we did get a few nasty memos from people essentially saying that they were professionals and should be trusted to develop programs that were highly efficient. However, CDC had sold a number of their 3100 computers to replace IBM 1401's as printing system, promising that they would exceed or at least equal the printing capability of the 1401's. Unfortunately, the systems delivered only ran at about the one-third the speed of a 1401, and all of the customers were very unhappy about it.

Once we found out about this problem, I spoke with Clair, pointing out we could provide the information needed to perhaps satisfy the promises that had been made. He agreed and set up a team to use the tool. They were able to eventually match the speed of the 1401's, but since the print program had to handle a large number of different file formats they had to optimize the print program over a large class of workloads. Still, they were able to cut run times by two-thirds. But, no one else used the measurement tool insofar as I know.

A few years later, a good friend of mine named Dave Morley was transferred to the Analysis group. Since he was an accomplished 6600 programmer, and the 6600 system had a very unique architecture, I was curious to know how to provide a measurement system for that type of architecture. Dave built such a system, and proved out the same type of questions I had had for the 3000-based software. Still, no one wanted to use it. I didn't try anymore within CDC to push the idea of measurement.

Insofar as doing any work on my "Decision Tables and Data Structures" program development notation or methodology, in early 1967 I found myself with some time on my hands. I had been asked by Clair's boss back in Minneapolis to move there and to apply my same methods of setting up an effective product design process to establish a unified hardware and software design process for CDC. It was tempting but I had refused, saying that I didn't want to move from Palo Alto. After that I discovered that plans were afoot to move everyone back to Minneapolis. I could tell my days were numbered at CDC, and Dave Katch (who had come to work with me in late 1965) wanted to start up our own company. I agreed to do it if we could get a contract before quitting. So I wrote up a fairly complete paper on the methodology, including a complete example of its use, because I wanted to summarize all of my understanding of the notation/design methodology in case I was unable to return to it for a while after Dave and I started our own company[4].

Yost: What was the name of this work?

Kolence: Let's see. Here it is. It's called; "Decision Tables and Data Structures, A New Design Methodology" and is dated July 1, 1967. It's long too – 153 pages.

I also wrote up a much shorter paper for the local ACM organization on the need to integrate a fully compatible management system with any technical design methodology. Entitled "On The Interactions Between Software Design Techniques and Software Management Problems", and was only 7 pages long. It was published in the local ACM newsletter. But, once again, I got no interest in my paper. However, in 1968 when I was invited to attend the NATO conference on Software Engineering, each person was required to prepare a paper on the subject. Since I considered my paper to directly bear on the subject of Software Engineering, I submitted the same paper. It was accepted, and when the report on the conference was published it was one of the most referenced papers.

Boole & Babbage

[4] While I think of it, I also should mention that Jen Brian also joined CDC here in Palo Alto shortly after Dave joined me. Jen became manager of the Palo Alto Operating Systems group.

Yost: So you decided, with Dave Katch, to start a company called K&K Associates, which soon was renamed Boole & Babbage. How and when did this happen, and why did you change the name to Boole & Babbage?

Kolence: Well, as I mentioned earlier, I had agreed to start up our own company with Dave if he could get a contract before we started. Dave had some contacts in Fairchild Semiconductors, and they had wanted a software system to use with a chip tester. For exactly what purpose I no longer remember, but I think it had something to do with culling out bad chips. Dave told them we could do it, and they gave us a contract. This happened sometime in June or early July 1967, so we found some inexpensive office space, quit CDC, and started to work. We already knew that we wanted to name the company Boole & Babbage Inc. But we had no money to incorporate so we settled on a partnership until we could afford to incorporate. Since our names were Kolence & Katch, or Katch & Kolence, we took the name K&K Associates. That way I could say the first K was for Kolence, and Dave could say Katch was the first name.

Dave and I had agreed when we were together at North American Aviation that we wanted someday to form a company, and at that time had agreed on the name Boole & Babbage. I had reserved that name at the state agency that gave out corporate names, and had kept it in force over the years.

Now Dave had worked for me at CDC primarily as a person who represented the Analysis department on Design Review Boards and helped to produce procedures and related material for the “Programmers Handbook”. So he was interested in the design process methodology, but not the measurement products. I wanted to build sampling monitors for the IBM system 360 computers, but several things prevented me from doing that for a while. First of all, we didn’t have any money except what we able to earn. So we really couldn’t afford to hire people who could work on these products. Secondly, IBM had promised to provide Operating Systems that could simultaneously managed several different programs running concurrently. Unfortunately these systems had not yet become reliable, and besides which none of us knew the guts of these systems. All in all then, we really couldn’t even consider starting to build the measurement software products. So we focused on consulting work, hoping to get together enough money eventually to start building and marketing the measurement products.

Yost: How did you get venture capital to eventually finance your product development?

Kolence: Well, once again Dave Katch came up with a source of more money: His insurance man was associated with several people who were interested in investing in start-up companies such as ours. In particular he introduced us to Franklin Pitcher (Pitch) Johnson, a fellow about my age or a little younger who had taken a FORTRAN course and done a little programming. He was also a Stanford MBA with a bit of venture capital to invest. We explained the notion and uses of a software product to measure software performance and identify where the greatest performance improvements could be made.

He discussed the idea and potential with his group of investors, and they came up with an offer. Basically, they would give the company (not Dave and I) $50,000 for 90% of the stock, and would also guarantee us a loan of $100,000 when we were ready to begin product development. We also had to put in $5,000 apiece for our 5% of the stock. Since we were the first software product company in what was just beginning to be called Silicon Valley, we felt lucky to find any capital to fund development. Dave and I discussed the deal, and decided to accept the offer.

By today's recent software venture capital deals, one might think we got taken pretty badly. However, in those days money was worth a lot more. One could live pretty well and even buy a house on a salary of $1,000 a month. For example, my salary when I left CDC was about $1,250 per month. So we could hire some very good people with that money and guaranteed loan that we were given. (I must admit though that I wish stock options had been available at the time.) In fact, we were able to complete the products with this amount of money, plus what David and one or two others of us were able to earn on some consulting jobs.

We incorporated as Boole & Babbage, Inc. on October 1, 1967. However, We had hired a very capable young man, Gary Holtwick, a month or so before to help us with some consulting work. And we also hired Dave Morley from CDC about the time of the incorporation. If you recall, Dave built the CDC6600 version of the program sampling monitor. He also had extended its capability to monitor the overall utilization of various hardware components such as the CPU, disk drives and "channels', along with the equivalent for tape drives, etc. This type of information was not needed for most systems prior to the advent of IBM System 360 computers, but was very important for these new systems. I'll explain more on the importance this of this later.

Sometime after the first of the year (1968), our contract with Fairchild was cancelled. It – the contract – had been sponsored by Bob Noyce, but when he left Fairchild his replacement decided to follow a different approach to resolving the problems we had been working on. This was actually a good deal for us, because it freed me to put the product development effort into high gear.

Yost: Very interesting. I had wanted to ask you if at the start of K&K Associates and Boole & Babbage in ’67 you had an association with Fairchild. Let me also ask if there was a sense that Silicon Valley, or what came to be called Silicon Valley, was going to take off as a high tech region? Or not?

Kolence: You know, I think the answer is, yes, sort of. Not for software, but definitely for silicon. There is no question about that. Integrated circuits were starting to come out. In fact, the building where Noyce independently invented integrated circuits with Shockley is not very far from here. Actually it’s within walking distance. I go past it quite often.

Yost: And Boole & Babbage was the first software firm in Silicon Valley?

Kolence: Yes, the first software product firm. There may have been some contract programming companies, but they weren’t product oriented.

Yost: Was there much association between you and other emerging software companies, such as ADR or awareness of those companies at that time?

Kolence: No. It wasn’t until Larry Welke started up his ICP directories that I really even knew there were product-oriented software companies in most other parts of the country. I can’t think of any other names at that point in time where they were building software products *per se* here in what became Silicon Valley. I guess I did know about ADR and a few defacto software product companies based in Southern California. But at Control Data I would not have been interested in them. When the Boole & Babbage products came into being, Larry was already publishing his directories and was about ready to give out his “Million Dollar Sales Awards”.

Yost: Did you develop the IBM measurement programs using the development process you had developed and installed at CDC?

Kolence: I’m afraid not, for several reasons. First of all, we had experience in building these products, at least for the CDC product line. Also, the experiments on randomization of the sampling process didn’t need to be repeated, so we knew the algorithms we could use there. Finally, the products were functionally divided into a sampling program and a report generation program. The sampling program was the same for both products except for where data was collected from in the OS code. The Report Generation program was likewise basically the same except for the data manipulation subroutine. All of these reasons were important because we really couldn't afford the time and money it would of taken to use the formal methods that were necessary for the large scale systems of CDC.

I guess I need to give the product's names to discuss why we could develop them so quickly. The program measurement product was called the Problem Program Evaluator, or PPE. The hardware configuration utilization product was called the Configuration Utilization Evaluator, or CUE.

For PPE we needed to know how to interrupt the computer using the interrupt timer, which was fortunately a standard feature on System 360 machines. Then we needed to know where the interrupt stored the memory location of the address of the instruction the machine would return to when the interrupt ended, and finally we needed to find a way to determine if the interrupt had actually been completed or not. With this information we could determine the location that would become randomly active, and be assured it would only be reported once per sampling action. The code modules that extracted this type of data were called the "data extractors" and the report generators were called the "data analyzers".

Essentially the same process was used for CUE, but now the locations being observed gave the busy or not busy status of the set of hardware of interest. From this information we could then process the data to show when component hardware sub-systems were busy or not, and when subsystems overlapped one another in execution. Generally speaking this was sufficient to determine where to change code or hardware to effect the desired amount of performance increase.

So the architecture of the products was almost identical for PPE and CUE and was easily coded. The hard part was finding the required data within each version of each release of the IBM OS. That is to say, we were really only building fairly simple utility products, not full-scale applications. We had built up a small development staff, mostly of ex-CDC people, and they worked together well. Dave Morley was the person responsible for the analyzer programs, and for specifying the data needed for the data analyzers. Gary Holtwick and Andy Chapman were responsible for the data extractors and the testing efforts. Andy was of particular value to us because he had been involved with these OS versions since their original "beta" releases about a year earlier.

Yost: Who were the early customers of PPE and CUE?

Kolence: Well, the earliest users were three test sites in San Francisco. They provided us with the descriptions of the several different versions of each type of OS so we could then determine where within the OS we could find the proper sample data. Once we had done that and coded in correct locations, we then needed to make a number of runs using a set of known, typical applications and sets of data. Then we would let the installation people try it on their own. Assuming everything seemed stable, we would keep that version. However, there were so many different OS versions of each type being released in that time period that we were hard pressed to keep up. There were three different 360 OS systems promised for delivery, two of which were being tested and released at that time: PCP (Primary Control Partition) which could only accept one program at a time, MFT (Multiple Fixed Tasks) which could only operate over a given set of programs (I think). The last OS promised, but not yet in test at that time was called MVT (Multiple Variable Tasks).

As I recall, we had three test sites that we got to use in return for the software and some reasonable help in using them. Chevron was the main one, the City of San Francisco was a second site, and I can't remember the last one.

But the first commercial customer, one that we got money from, was the Guelf University Data Processing center in Canada. I can't recall the exact order of the other early sales, but two others in San Francisco were the Bank of America and Del Monte. Two others I recall were AT&T in New Jersey and John Deere in Moline, Illinois. We made the Guelf sale in January of 1968, and increased sales every month until we were selling about eight per month by about September of 1968.   Further, our sales were all from word of mouth references and some SHARE and Guide presentations later in the year.

Yost: How did you price the products? I don't believe there were any guidelines for pricing software products at that time. How did you come to price PPE and CUE?

Kolence: No, there were no guidelines for products, and certainly none for the type of products we were offering. Plus, there were many people who honestly felt that they shouldn't have to pay for software. And, worst of all, there were no budgets for paying for non-IBM software.

What we really wanted to do was sell PPE and CUE as a package, so that the systems people would use CUE and the Applications people would use PPE. Now in those days you could buy a disk – a 1311 model as I recall – for a little over $12,000. So we priced the two as a package at $12,500 so they could trade the purchase of a small disk for both PPE and CUE. Since we had to keep upgrading them both to work on the latest releases of each IBM OS, we also gave them a free year of "software maintenance" – that is PPE and CUE upgrades - and thereafter it was originally $500 per year. However, this fee was too low, and eventually we charged 10% of the purchase price.

Back in those days the amount of money we are talking about seems ridiculously small now, but as I have mentioned that was before all kinds of inflation that hit during the late '60s and '70s.

Yost: How did you get sales leads and how did you close them. Did you have salesmen or what?

Well, we did hire a salesman, but selling software was such a new thing that he wasn't really able to bring in much business. As I said earlier, most of our business came about by word of mouth and also articles in Computer World describing the results of using the products. Also, I went to SHARE and made presentations, and since I still had a number of good friends there I was able to set up sales presentations with them. But the best early leads came from a booth we had at the Fall Joint Computer Conference, held in San Francisco in early November 1968. Many, many people stopped by our both, including IBM salesmen.

Our products provided a completely new kind of capability to improve systems performance, both software and hardware. Since people had never seen what could be done using our techniques and products, they were naturally somewhat skeptical that their installations could be improved so easily. So, when people did get interested we made them an offer that few refused. It was a great closer.

We would say, "Give us a signed contract for both products and we will come out on our money. Give us three programs to analyze and give us 24 hours to fix them, with help from your people who know what the code has to do. Then, we'll re-run them and if we don't save you thirty percent run time, we'll forget the sale." And at the same time we would run CUE and show them where their bottlenecks were, so they could also figure out where their system needed tuning. I don't recall ever having lost a sale doing that.

Yost: So you were able to beat thirty percent all of the time?

Kolence: Oh yes, or if not, so close that we still closed the deal. Many times we far exceeded that percentage.

Yost : What was the largest percentage improvement you can remember?

Kolence: Well, the one that comes to mind was at Lockheed Sunnyvale, just down the road from here. They had a data analysis program that they used on telemetry data, and it was running much slower than they had expected. They had already looked at the code, and hadn't found any obvious reason for it. So, we ran PPE against it and immediately found the problem: there was a data correction routine that being entered for each piece of telemetry data, regardless if there was a problem in the data or not. As I recall, correcting that and a few other pieces of the code resulted in an over eighty percent improvement in run time.

As it happened, I found out then that my professor from the University of Illinois, Dr Jack Nash, was head of Engineering Computing for Lockheed. I didn't know he was in the Bay Area, and was delighted to see him. After speaking with Pitch Johnson, who was Chairman of our Board, I asked Dr Nash to become a Director of Boole & Babbage. He was most pleased to be on our Board., and was very helpful in many ways.

Yost: How did IBM react to their customers using PPE? Am I correct in remembering that you disabled PPE to a certain extent so it would not work on IBM products?

Kolence: That's correct, and that's why it was named "Problem Program Evaluator". I figured if PPE was used to look at, say, a COBOL or Fortran compiler, or even IB Sort, that users might put a lot of pressure on IBM to improve the run times of such systems. It would have been very difficult for IBM to provide software that was efficient over all the workloads various customers might have. Remember the problem that CDC had just trying to get the same kind of efficiencies for just a small number of customer print workloads.

Actually, PPE could even have been built to provide histograms of even the OS in execution. I figured that the last thing I wanted was to give IBM a reason for blocking PPE from working. So we put code into the PPE data extractor to prevent it from collecting histogram data from IBM products. Well, I was probably smart to have done that with PPE, but as it turned out the real threat to IBM was from CUE and I didn't realize that would be the case.

Yost: Yes, I see your reasoning now for PPE. However, before I ask you more about why CUE caused IBM problems, I want to ask more about Boole & Babbage as a company. Did you abandon your work on software development methodologies? And what other products, if any, did Boole & Babbage develop?

Kolence: Lets simply say that there was a hiatus in my interests in software development methodologies. I actually have never abandoned my interests or studies in the subject. In fact I have spent a great deal of time over the last half dozen or so years working on understanding the general principles of design activities. However, I think you mean did Boole & Babbage abandon the work Dave and I had done on the subject at CDC. The answer is no. I had to focus on the product side of the Boole, and gave him the responsibility of providing consulting work, especially on how to establish and use good software design practices.   Over time we developed a fairly profitable consulting practice in this and related topic.

Sometime around 1970, the Supreme Court ruled that the Patent Office had to give out patents on software products, so we prepared a file application and submitted it. The two inventors given in the patent were Gary Holtwick and myself. I felt Gary deserved to be my co-inventor since he had done the programming on the extractor. (Thinking back, I should have also listed Don Lytle and Dave Morley for the work they did at CDC.)   The US patent Number 3,644,936 was awarded on February 22. 1972. We were unable to enforce it however for several reasons, one of them being that the US government claimed that their contracts with us gave them rights to use the methods in our patent. It was a big mistake to accept that argument, but our lawyer talked us into it.

While I was still at the Boole, there was only one new type of product that we developed, because of how, in my opinion, IBM tried to put us out of business. This was a product to try and reduce the amount of seek time on disks, and thus I-O wait time, by reorganizing the placement of these data sets. This product, whose name was something like Disk Data Set Optimizer, had a data extractor that collected information which gave us the ability to calculate how much I/O search time could be reduced by moving workloads to different locations on a given disk, or to other disks. The original thought was to use Linear Programming techniques to determine an optimal data set placement from the data collected. However, this proved to be extremely difficult to do. Instead we used heuristic technique that consistently produced very good results. Most if not all of the disk defragmenter software products today ultimately descend from this product.

The product was an excellent seller, and helped us to weather the difficult times that were coming up for the Boole. One of its most important results was that it got rid of the typical Data Center practice of assigning all data sets used by a department to its own 2311 disk pack. That step alone vastly improved performance in shops that had used this technique.

Yost: Going back to the problems that IBM caused you, could you first explain what IBM did and why it was done.

Kolence: First of all, I have no proof that IBM as a company did what I think they did, or why they did it. So this is my personal view of both things. Regardless, the events I will describe did happen.

Let's start with the reason I believe led to everything. Back in the '60s and early 70's, companies normally asked their IBM reps to help them with the equipment plans for the companies. I understand that IBM established levels of what was called "account control", the highest level being when the IBM rep was the de facto person who decided what equipment the company would obtain. Once the time sharing System 360 came into operation with either the MFT or MVT operating systems, the problem of determining what was the right equipment to get became impossible to solve by the previous methods. This was because when you are executing a mix of software on a time sharing basis, the bottlenecks that occur are a characteristic of that particular workload mix. A different mix of programs and you may well get different bottlenecks. The problem is that without some measurement tool by which to see what is happening with different mixes, one has no idea of how to reduce bottlenecks and thus improve throughput times.

Apparently, the IBM reps to most companies suddenly were overwhelmed by complaints by their customers because the equipment they had purchased was not giving them the throughput the had thought they would get. This certainly happened at most of the Northern California and San Francisco area sites. When the CUE product was brought in and used consistently, the company technical people found they could figure out themselves how to resolve their throughput problems. The companies began to establish their own equipment planning groups, since they could get the data whenever they needed it. In turn, this meant that the IBM account rep no longer had full account control, and his superiors were not happy. Eventually this was happening in a large enough section of their customer base that they decided to do something. That something, according to the sworn testimony of a person (whom I shall refer to as “Mr. J.”) directly involved in the effort, was to build a copy of the two Boole & Babbage products and give them away free to their customers. I believe they expected this would cause the Boole to go under fairly rapidly. They were almost right, but they forgot how much maintenance effort it took to keep those products reliably working.

The account that was selected to produce these copies was that of the Stanford Linear Accelerator or SLAC in the hills above Palo Alto. SLAC was (and still is) a world class physics center concerned with experimentally investigating and verifying various predictions of particle physics theories. I believe that SLAC administrators were unaware of what was happening, but that this was not true of several people in the computing facility. To begin with, we were invited by them to make a presentation on PPE and CUE, how they worked, and how one could use them to solve various problems. This we did, expecting them to let us know if we could give them our usual offer to go there and run our products, collect data, and show them that our claims were true. We did get a few calls, and when we called to see if they were ready to buy we got vague promises but no action. So we finally gave up on trying to make a sale.

Somewhat later we received, from some unknown but honest person, a copy of an internal SLAC computing center memo that outlined the progress two IBM sales support technical people had made toward building a copies of PPE and CUE. In it, they said that it could be released and distributed to COSMIC within a fairly short time. COSMIC was and may still be a NASA sponsored system for the distribution of free software to government agencies and whomever else might be interested.

To say the least, I was shocked, angry, and most of all scared at what might happen to our sales, which were doing quite well. My first inclination was to talk directly with a person who I assumed would be able and willing to prevent their copy from being distributed free. After discussing this with Pitch and others, it was agreed that was the best thing to do. The person was at that time a highly respected officer of the National ACM organization, or had recently been, so I naturally assumed he was a reasonable and ethical person. I called this person up, told him I understood SLAC was getting ready to distribute their copies of our products, and wanted to speak personally with him. He agreed to meet with me the next day. I went up and he said, "Let's take a walk."

We went outside and walked to a place where no one was around, and he admitted that they had built the copies of our products and that he was going to release them to COSMIC. In other words, he was the fellow in SLAC that had responsibility for the project. I remember saying to him: "Please don't release this to Cosmic. If you want to make it for yourself that's your business, but you're trying to kill off this whole thing.   We are the ones that invented it. We are the ones that invested our money and our effort to make sure it would work and to provide it to other people and to keep it maintained".   And he looked at me and he said, "You know, what you should have done is made it in hardware, not software". That made me very angry, because he had to know that there was no way either product could be easily put on a board. I must have shown my anger, because he looked right at me and said. "I am going to distribute it" and walked away.

I have never been so angry with anyone before or since.   Here is this guy, a big professional, you know, and he's doing his best to help IBM keep in control of all the installations, configurations, sales stuff.   From that point on I had a very negative attitude toward an awful lot of people in the ACM. That attitude is gone fortunately. But his decision certainly changed my career.

Well, anyway, SLAC went ahead and released their software to Cosmic. But it was a strange thing that the day that it became available from Cosmic, in every account that we were in sales negotiations with the IBM rep came up and said something to the boss like this: ‘Look we’ve got this software in Cosmic now, it’s free, it does the same thing as the Boole & Babbage software, and you don’t have to buy it.” All of our sales activities stopped. We were dead in our tracks. The only thing we were selling was our new disk data set organizer. . Fortunately, we had our consulting work too This went on for several months. But we were barely making it and I, in particular, was going around just working my tail off.

To top things off, a little while later I went to a GUIDE conference and tried to drum up some sales. And here’s this guy Mr. J. talking about the measurement products they had built at SLAC. He never once mentioned Boole & Babbage but he really strongly intimated that they had invented it first, and that “someone else” had stolen the ideas and commercialized and commercialized them. Apparently he didn’t know who I was because although I knew his name I had never met him before, and here he was busy trying to take credit for inventing all these products. I lost it. I lost it completely. I really lost it. And I got up and I told him what I thought he was, what kind of a person he was, et cetera. And I did this in front of a crowd of people.

Well, I was not too popular with the GUIDE people after that, because they believed the SLAC/BM’s story. So that outburst got back to Boole & Babbage, to Pitch certainly. And I lost my temper on some other occasions. I had been working so hard to generate sales that I couldn’t remember if I had said something, read it, or dreamt it, or what. I began acting irrationally. I was very close to a nervous breakdown. I blew my stack at Gary Holtwick, the guy who wrote the original extractor code. Gary went to Pitch Johnson and complained about me - I don’t blame him – I had lost my cool completely.. There was no question about it.

So I guess Pitch decided he had to get rid of me, but first he had to find someone to take my place and get up to speed on what we were doing. I could tell that I was being set up to be fired, since I was being forced to fire people like Dave, Jen Brian (head of marketing for the Boole) and other top people I had brought in. I turned in my resignation every month for about 6 months and Pitch would refuse to accept it. Finally, after he had found his new president and had him up to speed, Pitch fired me. I don’t recall exactly when this happened, but I’m pretty sure it was in the summer of 1972.

Quite frankly, I was relieved the entire thing was over for me. I went home and slept for a month or so. But I never would have fallen apart the way I did had SLAC and IBM not done their dirty work. Looking back at things now, I also think the field of computer measurement would be far advanced from where it is now.

Yost: Did Pitch have a number of other enterprises?

Kolence: Yes, but not software at that point in time.

Yost: When COSMIC released the SLAC software, was there a real sense in your mind that Boole & Babbage would have a difficult time recovering?

Kolence: Oh yes indeed. If I had not worked as hard as I could to get some sales, and other people working very hard to do the same thing, we absolutely would have gone under. It was something like 6 months before our sales began to come back. The Cosmic products were not being maintained, and IBM was always coming out with new OS releases. The lack of reliable maintenance forced them to by Boole products. I have no idea why maintenance wasn't provided, but perhaps the SLAC/IBM people found too much of their time was being taken up by making changes to their software to get them to work with the new OS releases. Maybe also higher-level people in IBM found out what had done and stopped it. After all, IBM could easily have gotten themselves caught up in anti-trust law violations. Whatever it was, it was too late for me.

Yost: How many employees were there?

Kolence: There were twenty-five or thirty when the COSMIC release occurred, split about evenly between consulting and support teams for the products. There were considerably fewer afterwards.

Yost: Were most of the people, besides you, Dave, and Jen Brian also former Control Data people?

Kolence: Yes, to some extent. David, Katch, Jen Brian, and myself were really all from North American Aviation, through Control Data. But Gary Holtwick, Dave Morley, and several others were from CDC.

The Founding 1968 NATO Software Engineering Conference

Yost: Lets go back now to October 1968 and your invitation to attend a NATO sponsored conference intended to found the field of Software Engineering. You also have already said some things about some papers you brought to the conference. But how did you get an invited to attend it?

Kolence: Once again I owed a debt to my friend Donn Parker. As I recall he was offered an invitation but could not attend. Donn strongly recommended me instead, even though I was with Boole & Babbage by then, and explained why he thought I would be a good attendee. He never mentioned it to me though, so when I got an invitation I was really surprised and thrilled to be invited. The conference was to be held in Garmisch-Partenkirchen, near Munich from October 7th to the 11th. Dr Alan Perlis from Carnegie University was the organizer, and NATO was the sponsor. Perlis picked a good time to hold the meeting: It was Oktober Fest time in Munich right after the conference.

Yost: You've indicated earlier that you coined the term "software engineering", and later came to learn that it had come into limited use.

Actually, I don't recall ever using that term in any of my writings until after the conference, although I probably used it in some discussions at North American Aviation and CDC. But, I had never heard or read of anyone else using the term before that conference, so I was both surprised and delighted when Perlis came up with the name and the conference.

Yost: How successful do you think the conference was in establishing a software engineering discipline? Or was it really just an attempt to professionalize or give greater status to programmers?

Kolence: The most obvious answer to your first question is that the NATO conference was not successful at all in establishing a true engineering discipline for designing and constructing software. This answer must be true because we clearly do not now have an engineering capability comparable to those based on the physical sciences. Such a capability is what is implied by the term software engineering.   But software is not a physical system, it is a description of what one describes how a given physical system (a computer of some sort) should act. Thus software is more akin to engineering drawings, either maintained on paper or within a computer. That is, one may consider software to be part of the family of what might be called notational or descriptive symbolic systems. I think that is the case, so probably it is incorrect to assume software engineering should be considered as a sibling in the family of the physical science based engineering disciplines.

Does that mean we should thus repudiate the title of software engineer? No, there are people who call themselves software engineers, and who build software systems that other people trust their lives to that software. So a great deal of progress has really been made since 1962 (when I first began considering myself as a software engineer) and in that respect toward what may be considered as a valid engineering capability. Further, I believe many of the people who call themselves software engineers are honestly trying to live up to that description of themselves.

I definitely think using the term for a NATO sponsored conference actually did, for the first time, provide programmers with a more professional description of what they were trying to do. In my opinion, that was a very good thing. In my own case, when I returned to Palo Alto, I began advertising for Software Engineers rather than programmers. There was no such category for the Help Wanted classified ads until I began using the term in the Boole’s advertisements, but within only a few months almost everyone was using the term. So I clearly was not the only person who intuitively felt it to be good title to use.

Summing it up then, I think the answer to both of your questions above is yes, but that while software engineering may be based on good principles of design activities, it is not a physical science based engineering body of knowledge. So we have to build an engineering capability based on some other body of knowledge, which in my opinion will draw much – but not all - from that other great body of knowledge, mathematics.

Yost: You Bring up an interesting question then. I know the conference report was apparently an attempt to use the submitted papers to synthesize a sense of what Software Engineering might be or how it might come into being. But just a handful of people prepared the report, and this was done well after the conference. So my question is: How was the conference structured and what was presented at the conference?

Kolence: To begin with, the conference did not appear to me to be structured with any real intention to develop or understand how one would go about developing a software engineering discipline. I don't recall any sessions about how such a discipline might be brought about, or what basic kinds of science might provide the underlying understanding of "software" equivalent to that which anchors the established existing engineering disciplines. The invitees were each supposed to provide a position paper on some topic we felt would contribute to the development of a discipline of software engineering. I may be wrong on this, but as I recall that was all of the direction we were given.

I decided to submit my short, seven page paper I had written right before leaving Control Data, the one entitled "On The Interactions Between Software Design Techniques and Software Management Problems". I felt it laid out some basic reasons why design and management processes had to be synthesized together early on in any true Software Engineering methodology.

I also took with me a copy of my full-scale 153 page paper “Decision Tables and Data Structure, a New Design Methodology.” The purpose here was to show a complete example of how a program description could be structured in a way that correctly provided both decompositional (top-down) and composional (bottom-up) views of process logic corresponding to the levels of data structure used. At the last minute before leaving for the conference, I also brought several successful runs of both PPE and CUE outputs by which to show how the performance characteristics of software and hardware could be determined an improved if necessary.

Quite honestly, I remember feeling I had three significant pieces of a software engineering methodology in my suitcase. I left for the conference with high hopes that it would truly be the start of a real discipline of software engineering.

When we arrived at the conference each of us were asked for a copy of their submitted paper and given time for a ten or fifteen minute talk on its contents. In my case, I also asked for some extra time to show the PPE and CUE reports, and was given a few minutes in conjunction with another talk by some academic on methods of instrumenting software. I believe Dr Perlis also opened each day with remarks on the activities of the day, and finished the days with comments on some of the presentations. But there were 50 of us at the conference, so no one (except Dr Perlis) had time to talk about how our presentation material might fit with that of others and into a software engineering methodology. I must admit, after a few days I had become rather disillusioned about the prospect of this conference providing any helpful ideas about how to begin forming a real engineering capability for software.

Part of my reason for this disillusionment was related to the backgrounds of the attendees. About a fourth or so of us were from the business world, and the rest were basically from academia. It was clear that Dr Perlis had invited the business people simply to appear that he was interested in what we might have to say. I had the distinct impression that he had no interest in talking with any of us who disagreed with his main principle for software engineering. This was apparently: that he only way to build software was to only have small (6 or 8) people teams, and not burden them with management reporting responsibilities. He liked to say things like "Don't throw Chinese Armies at development problems", an allusion to both IBM's large and well managed (for its time) System 360 software effort, and the Chinese army's incursion into North Korea in the 1950's. As a matter of fact, he said it so often that it was clear he didn't believe in the main point of my little paper, which was and is that software product development requires (but is not itself sufficient) the design of rational and correctly integrated management and technical processes.

So, in answer to your question Jeff, I think there were some interesting presentations at the conference, but really there were no constructive discussions about how to bring about a discipline called Software Engineering.

Still, I was in Germany, I could practice my German, I had met several people with whom I would became fast friends over the years, and I was going to attend Oktoberfest at the end of the week. So I never complained, and still felt it was a great honor to be invited to attend the conference. On the other hand, I was not invited or even told about the conference in Rome the next year. So I guess I didn't make any impression on Dr Perlis.

Two of the people who would become my good friends were a gentleman named Alex d'Agapeyeff and Ian Hugo. Both of them are Englishmen. Ian was a freelance consultant and writer, and had the job of recording the conference, and afterwards acted as the editor of the conference report.

Alex was the founder and Managing Director of an English company named CAP, which if I recall correctly stood for Computer Accountants and Programmers. Among other things, CAP did a lot of commercial applications development and consulted with other companies on systems analysis. I told him about our products and showed him the outputs I had brought, and became very interested. I suggested he might be interested in representing us in the UK, since he knew many if not most of the people using 360's in London and the other cities. He liked the idea but worried about having the right kind of people to support the products. So I offered to send Andy Chapman over to London for several months to help get him started after we had gotten PPE and CUE fully productized. In February of 1999 or so Andy went to London, and CAP started selling our products. Later, with our permission, Alex struck a deal with another company, based in Holland or France (I can't recall) and also named CAP, to be his agent on the continent. So Boole & Babbage was one of the very early US companies to set up agents in Europe to sell American software products.

Yost: How did Alex take to selling software products? I gather his people were all like consultants or programmers.

Kolence: It did take him a bit of time to warm up to it. But when he realized his people could use it to tune their systems and then in addition sell it to the customer for use on their other systems. He started liking the idea a lot. He talked about building his own products and having a ”software tool kit”

The Founding of the Boole & Babbage User Group (BBUG), and Its Conversion to the Computer Measurement Group (CMG).

Yost: This may be a good place now to ask about the Boole & Babbage User Group.. When and how did it get started, and what was Boole & Babbage’s role in it?

Kolence: Yes. The Boole & Babbage User Group was founded by a user, Dave Schumacher of Lockheed Sunnyvale, in the Fall of 1969. It became the Computer Measurement Group (CMG) in 1975, and is now the world wide professional organization for computer and software performance and capacity management issues.

Dave Schumacher came to me in the summer of 1969 and said he wanted to start up a Boole & Babbage user group in the Fall. He wanted to get my OK to do that, get some monetary support, and find out how I wanted the Boole to be involved. Well, to begin with I was delighted with Dave’s willingness to start up such a user group for us.

However, if you will recall I was very much involved with the IBM scientific User Group called SHARE. It was quite successful in presenting the user's needs to IBM, and pressing them to provide for such needs. At that time, SHARE was independent of IBM, though IBM did pay for coffee breaks and such. It was a great organization, and IBM benefited greatly from it being a "loyal opposition". CDC also had a User Group, the name of which I cannot recall. However, unlike SHARE, CDC ran the group completely. That group was not at all a vital organization, and I attributed that to a lack of independence from CDC, especially since I had observed the same problems with other user groups. So when Dave asked how I wanted the Boole to be involved, my answer was something like: " Fine, we'll give you money to cover your expenses, but other than that it is up to you entirely. I don't want to have anything to do with it and I don't think Boole & Babbage should have anything to do with it except to just work with the group." Since Dave knew SHARE, he understood what I was saying, and he proceeded to organize and let the users know what he was doing, and the first meeting was held in the Fall near the San Francisco airport.

The first meeting was really just concerned with Boole products and how they could best be used and at I guess about fifty people showed up, including a number of possible sales leads. At subsequent meetings however Dave had other companies with measurement products asking to show their products and give presentations. I thought it was a great idea to let them participate, figuring it would help the entire industry of computer measurement to grow. It did, too.

By 1973, we had a large number of vendors making presentations, and paying for booth space. At that time, a number of us felt that the group name should be changed to reflect this more general collection of interests. Dave and several others including myself (I wasn't at the Booel anymore then) discussed this at the 1973 meeting and, as I recall, were given the OK by the attendees to make plans for such a change to be presented and voted on. This was done, and the incorporation of the Boole & Babbage User Group as the Computer Measurement Group was approved at the 1974 meeting in Montreal, to take effect at the 1975 meeting in San Francisco. It was at that meeting that I gave a series lectures on software physics. But I'll go into all of that when we discuss the Institute of Software Engineering.

I would like to mention here that Dave Schumacher had, unbeknownst to me, gotten agreement to initiate an annual award to be called the "A. A. Michelson Award for the Advancement of Computer Systems". It was named after the famous physicist who was best known for his efforts to measure the speed of light. Don't ask me why he chose that person to name the award for, but he did. At any rate, I received the first award in Montreal, and it is still being awarded at the annual U.S. CMG conferences.

The main portion of my Award reads: "This First award is given in recognition of you outstanding contributions to computer metrics through:

- Your contributions by your early work as a scientist, engineer, marketer, and teacher. Your scientific research led to the design of computer measurement tools, your engineering led to the development of those tools, your marketing skill convinced the computer industry to accept the tools, and you taught computer professionals to use these tools in the measurement and evaluation of their computer systems.

- Your creation of the industry of computer performance measurement by bringing to market the Computer Usage Evaluator (CUE) and the Problem Program Evaluator (PPE) software monitors.
- Your pioneer work in the rigorous methodology called Software Physics for use in the computer measurement and capacity planning fields."

CMG continues to be the major professional society for those interested in managing performance and identifying bottlenecks of various sorts. A rich collection of papers on all aspects of measuring and using metric data to plan and managing computing and web based systems are presented at the CMG conferences. There are currently regional CMG's in Australia, Austria & Eastern Europe, Canada, Italy, South Africa, and the UK. Within the US., there are 20 different CMG chapters.

CMG and its foreign affiliates have been a major force in providing local installations and networks with the capabilities plan their own equipment needs and to provide reliable and satisfactory online transaction times and volumes. I'm proud to have been associated with such an important and viable organization over the years.

Yost: Thanks, it seems that keeping the Boole & Babbage User Group free of Boole control was an excellent decision. Another thing I wanted to ask you about is what was the role of the National Bureau of Standards in software measurement?

Kolence: I can't answer with regard to academic support, but there was and still is essentially no support for the industry side of measurement.. I did have a small contract with NBS, when I was at the Institute. My job was to write up a paper on how software physics could be used for charge-back purposes. But they were not at all interested in the "software engineering" possibilities of software physics.

Yost: Has there ever been federal support towards the measurement field?

Kolence: Only in that they bought a lot of the Boole's software. The basic answer to your question is "no" if you mean support of other than government and academic institutions. The army has produced some measurement systems I think, and the Navy originally sponsored the Practical Software Measurements Handbook. Of course, the DOD sponsors the Software Engineering Institute at Carnegie Mellon and they have some interest in metrics. But none of these organizations build software instrumentation, so the metrics they use are always hard to collect.

Several academics were and may still be active in metrics. In my opinion, the three most knowledgeable being (in no particular order) Dominico Ferrari at the University of California, Berkley; Victor Basili at the University of Maryland; and Jim Brown at the University if Texas, Austin. They may or may not have been supported by government grants, but they all attended CMG meetings at various times and wrote articles and/or books on subjects of interest to the CMG community. But in the main I doubt that many computer science academics are interested in computer metrics.

Yost: Lets finish off this interview with two related topics, how did you develop your Software Physics theory, and how did you form the Institute for Software Engineering and what did you do to make money?

Kolence: OK.

Yost: In your work in measurement software you came to see the need for more advanced theorizing about how it is done and more formal rules, and you came up with the software physics concept. Could you talk about how your ideas came together for that and how those ideas evolved in the early years?

Kolence: Well, the idea goes back to my days at North American Aviation, when the idea about the need for a software engineering discipline came into my head when I watched the first takeoff of an aircraft, and I realized the software built in those days could not be engineered to a point where one's life could be depended on it. In the following weeks, I mulled over the idea that a software engineering discipline had to be based on a physics of some sort, but I couldn't come up with an idea of what such a physics would look like. So I remember finally saying to myself something to the effect that I didn't know what kind of software physics we needed, but I did know that it would need to be able to measure things. What kind of things? In general I didn't know, but at least where I knew of one measure of interest; to wit, where a program spent its time. And I remembered the article in Factory magazine about random sampling methods. It was that recollection that caused me to set up the hand sampling of 1401 performance histograms during maintenance activities.

At any rate, I stopped wondering about what a software physics would look like at that point, until was running Boole & Babbage. Then I realized that I had better get some idea about what a software physics could look like, in order to keep coming up with new and useful measurement products. Probably, this was about the time we defined the “Data Set Optimizer”, or whatever it was called. Once I realized this, a thought came into my head to the effect that “how would I recognize a software physics theory when I saw one?” I felt that such a theory needed to be predictive, and since the natural physics, including chemistry, were the only truly predictive theories, then software physics would need to be modeled on physics and/or chemistry. And, it would also need to be expressible in some form of mathematical notation. So I started by trying to draw analogies between the basic variables of Newtonian physics and some of the properties of executing software.

Some of the software analogues to the variables of mechanics were fairly obvious. For example, an instruction, when energized, acts as a force because it changes the state of some symbol in some register. As a result, work is done on that register. One could equivalently say that an instruction does one unit of work when it changes a digital symbol from one state to another. (There are some simple questions of units and quantities here that need to be defined.) Also, one can say that velocity and acceleration can be defined in terms of the rate of sybol state changes per unit of time, and these can even be treated as vector quantities relative to different machine registers. But, the killer to this approach is that I could not find any quantity that was a satisfactory analogue to a quantity of mass.

It was about this time that the “troubles” began at the Boole, and I had to give up any further thinking about what a possible software physics would look like. Prior to those times however, I did present some papers on the subject even though I was still stumped. In late 1969, I gave a paper at the Fall Joint Computer Conference in which I sort of summed up where I stood as far my work on developing a software physics. I also gave essentially the same talk and paper at the GUIDE conference where I “lost it” at “Mr J’s” Guide session.

After I was let go from the Boole (“freed” was the term I used at the time) in the Fall of 1971, I was completely used up, and it was several months before I began being my old self. I was needed to earn some money at that point, and fortunately a good friend and supporter of my work helped me to get a consulting job at the Bank of America headquarters in San Francisco. The problem they wanted to solve was how to plan their equipment requirements. It was right down my alley, and I was very happy to get it.

My job was to develop some process which could be used with reasonable accuracy to predict workload growth and how what hardware configurations would be needed.

The B of A was a very progressive computer user, and were one of the first banks to use a computer. As I recall it was called ERMA, and I think it was built by GE or someone other than the normal computer manufacturers. Prior to my coming aboard, they had run a classical experiment to select a new set of computers from the newly available high-end machines. They took six of their standard applications and the corresponding data for their end-of-quarter processing for these jobs. They wanted to compare the times to complete these jobs on three different IBM computers, which were I think were IBM models System 360 65, 85, and 95. They used IBM's hardware monitor to count the instructions executed, as well as the elapsed times and overlapping CPU/I-O execution times. (I am pretty sure they used CUE for this latter data.) The experiment was run using the latest OS version of MVT (Multiple Variable Tasks), first separately for each application on each machine, then for two different sets of three of the applications run concurrently, and then all six applications concurrently, on each machine model. In essence they were trying to determine what processing power was deliverable to these three application configurations, by each machine model.

The data from this experiment showed that the number of instructions executed was essentially equal to the sum of the instructions executed by each application, regardless of whether these applications were run separately, in sets of three, or altogether. This was true for each IBM machine model, and true for both CPU and I/O instruction executions.

Although it might well be thought to be obvious, to me it was a confirmation that the Software Physics property "work" was an "extensive property", i.e., the total work of a system is the sum of the work done by the parts of the system. Thus one could with some degree of confidence characterize CPU workload mixes in terms of work to be done, and thus the total work required by various potential workloads, for at least the CPU. Since the CPU instruction execution speed was constant on the average, one could predict the execution time used by the workload.

However, I/O times varied with the placement of data on a disk and the individual I/O frequencies and concurrency of use by different data sets. So I needed to develop some equations that would give me the time it would take to locate and read or write the block of data required or provided, and how much I/O interference time would be generated on other I/O's. I did this, and saw how one could model various workloads, provided certain data could be obtained. In other words, the problem I was to solve was solvable.

I also realized at that same time the way to develop a software physics theory was to recast the work I had already done in terms of the energy and work done, since that permitted me to get rid of the need for a physical mass property.

It was at that instance that I had a real epiphany. I actually feel my brain crackling and apparently rewiring itself, because in just a few seconds I completely understood how to develop a viable and useful theory of software physics <u>and</u> in the context of the natural physics. However, it did take me a few years to provide appropriate definitions, notation, examples, and test and/or confirm some of my ideas experimentally.

After completing my consulting assignment, the Bank of America people recommended me to the Banking Institute in Park Ridge Illinois, to help them develop guidelines for collecting charge-back data for their members. As part of this assignment, I was able to use the idea of software physics work to provide a standard structure for determining overhead cost and likewise to determine direct costs and their overhead load to determine total costs for running each application. As part of this effort I was able to properly provide the various definitions needed to properly determine how to measure these various work quantities. At the conclusion of this effort, I turned to the writing of the final draft of my manuscript for the book entitled *An Introduction to Software Physics.*

Now, I want to emphasize one important point. I very carefully considered whether or not I should call this work a software physics. I finally decided to use the term because I honestly felt that the elements of the theory did in fact connect the ideas and properties of the theory to those of the natural physics. Sometime latter, I also read Claude Shannon's book on Information Theory and realized that Software Physics was in many ways simply an extension of his work, but restated in terms more appropriate to computer concepts. I most emphatically did not name my "software physics" because I needed such a body of knowledge to validate the term "software engineering." In fact, I don't believe I even used that term in the book, although the company I founded to educate the people in the applications of software physics was called the Institute for Software Engineering.

My book included the first description of the general process I called *Computer Capacity Management* and this concept was warmly accepted throughout the industry. After giving the technical concepts and basis of software physics and outlining the Computing Capacity Management model, the remainder of the book was given over to detailed information on what software physics methods were designed to be used with each function in the capacity management model.

Officially speaking the Institute for Software Engineering, or ISE as it was generally known, began business in 1976. However, as I have already noted, I introduced software physics at the first Computer Management Group (CMG) Conference, held in San Francisco in late 1975.. ISE was structured to make money by providing memberships and associated benefits. We provided reports and books on various capacity management processes, and also course on all aspects of the Capacity Management Model. We also held an annual conference in the late spring after the first year, and we also did a fair amount of consulting.

[Please Note: Mr. Kolence added footnotes for additional information during the editing processing.]