

Jousting with Dragons

Kolence and Software Standards

Margaret Greenberg

In summary of the article in the June issue¹, Kolence completed college with a Masters in Math and a “Specialty in Digital Computers” despite an admonition from his advisor, Dr. Nash, that there would be a need for only six or seven programmers. At that time most computers were huge and could only be afforded by government agencies. And many in academia did not see the pent-up demand in the commercial sector to transfer their work from accounting machines to a full-fledged computer.

After college he intended to join Navy Intelligence because he was advised by an officer that was the “only place where the Navy was actually using computers”, but waited too long and the spots were filled. Fortunately, he received assignments in computer operations instead of “running silent and running deep” doing submarine espionage in the Cold War. Instead, he got the opportunity to work on an unreliable Univac I. He learned to maintain the circuitry and as well as the value of following good, preventative maintenance. These experiences gave him confidence that he could organize a system for keeping a computer in good running order. In his next assignment, “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.”

He worked with Dr. Grace Hopper, famous for developing COBOL. He gained more insight into workload balancing with hard coded tape drives. It was really tough in those early days to find and fix all the problems. He also set up a systems programming

¹ Margaret Greenberg, *The First Michelson Recipient*,
http://www.cmq.org/measureit/issues/mit70/m_70_6.html (June 2010).

group.

In the private sector continued:

In early 1961 (after the BMEWS contract concluded), he joined North American Aviation Los Angeles Division,



a government contractor for the XB-70. This plane was a real technological beauty. In the mid-1950s, General Curtis LeMay was eyeing a replacement for the B-52 and the B-58. In the Cold War we needed a bomber capable of delivering a payload to our adversary's home cities, a world away. Therefore, it would need to be fast, cover long distances and fly high.

This project closing proved to be a game changer for Kolence.

The whole design was based on wind tunnel results and an initial computer model which took about 18 months to complete. Then, there was the movable wing-tips that provided the ability to make use of compression lift and gave the plane the ability to conserve fuel at high speeds.

[*Front view of the XB-70 with all three wingtip angles*](#)



In flight, the XB-70 could lower the outer wing sections either 25 degrees for flying from 300 knots to Mach 1.4, or a severe 65 degrees for speeds from Mach 1.4 to Mach 3+. Measuring just a bit over 20 feet at the trailing edge, these wingtips represent the largest movable aerodynamic device ever used.

The skin was made of stainless steel only .002" thick--that was the tolerance in the holes drilled in the frames of WWII fighters. Looking at the design specs, one engineer called it "foil". There were many other design features that were later implemented in later aircraft; however, the XB-70's (*Valkryie*) days were numbered by costs and heavy orientation toward missiles instead of manned bombers. At one point, manned aircraft were declared obsolete; however, the bomber camp in the military hoped that this design and its accomplishments would cause the missile camp to reconsider. By mid-1959, the costs of building the bombers (vs missiles) caused the military to scrap the bomber part by year end and direct the *Valkryie* toward reconnaissance. That's where the program stood when Kolence started in 1961.

On May 1, 1960, Francis Gary Powers took a U2 spy flight over Russia, got shot down, was captured and admitted he was spying. President Eisenhower was caught in a cover up by Russian President, Nikita Khrushchev. Thanks to Russian development of SAMs that were improving more rapidly than designs and capabilities of manned craft, there was absolutely no hope of redemption for the *Valkyrie*. There were two prototypes built with scheduled testing in 1964, however, the entire program was cancelled in 1969. It did fly faster than Mach 3, achieved fuel savings and much, much more. To read further about this fascinating project, click [here](#) or visit Wright Patterson AFB museum on your next visit to California.

For the second time, Ken experienced layoffs of very experienced and capable engineers, when he was forced to remove about one third of his staff. Though he, himself was not at risk, he dreaded the pattern of buildup/tear down that follow project conclusions and project scrapping or revamping. He became concerned about the fragility of government contractor jobs and decided to work for a manufacturing company where layoffs were much less of a threat. He put out feelers to IBM and Control Data (CDC), which were two of the best hardware manufacturers of the time; however, he was really not in the mood to go to the IBM HQ in Poughkeepsie, NY and cold winters. In January 1964, he drove up to interview with CDC in the Stanford Industrial Park. There was enough time for a drive around town and he fell in love with Palo Alto and the San Francisco Bay area. When offered the job, he accepted and moved his family in early 1964. He never interviewed with IBM—he'd found a home. It was not a financial decision.

He was hired by Dr. Claire Miller, Chief Engineer and general manager. His assignment was to work on "improving the capability . . . 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." He

was reporting officially to Dr. Miller's number two, Dr. Dick Zemlin, with authority to go directly to Dr. Miller without his OK. Not a bad situation at all.

His first task: "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. I figured I had plenty of time to get it done.

Dick introduced me around the organization. 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."

Unfortunately, Kolence had no idea just how much his help was needed when he signed on and buyer's remorse set in after the discovery of development chaos. All happened long before IT became "commercialized" and "structured" and "disciplined".

"I began regretting that I had joined CDC, 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. 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 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 architected series of computers, all of which had the same basic instruction set. The 3100, 3200, 3400, 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 classes of customers. Different operating systems were provided for the different types of computer systems. 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 be 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”.

This later became a controversial move for the industry: software and hardware not as a single product, but a series of somewhat independent products. He was going back to Factory magazine ideas of work structure.

“... 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. 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 and 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 which were 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.”

In addition to standards for development, he introduced the concept of Change Control and Change Management. Software development and management was often a free for all, open range event. That style was true for both commercial and scientific software development. PERT and Gantt charts were highly respected even in those days. Gantt

provided the who responsible for what and when. Additionally, the idea of parallel processes with task time allocations attached could be used to identify critical paths. Gantt charts were instrumental in the WWII effort, therefore, had high respect from business people who'd served.

The following narrative is interesting to read, if only to see how he methodically went about building the process of organizing software development and distribution at CDC. Much of what he set down as process is standard in development in all types of organizations today.

“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 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 experience. 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." [a document control and distribution system had already been established] "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*.

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. . . 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 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 what I wanted to do, and asked [my staff] 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 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, [they] went ahead and built the software. After a month or so, Don [lead programmer] 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. . . . The paper was 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, 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. When the report on the conference was published it was one of the most referenced papers.

In hindsight, it's amazing that no one else latched onto the value of having measurement software to prove that their systems worked or even that no developer displayed any concern about proofs of product performance. But over 40 years ago, the time wasn't ripe for such tools. Management had no interest in spending extra time in proving software quality—they and their staffs were overloaded with development and maintenance requests. Analysts and programmers were professionals, who "knew how to write efficient code." But times were about to change.

Next installment, we'll learn about Ken and Dave's adventures in starting their company, K&K, in what later became known as Silicone Valley.