A Personal Recollection of Software's Early Days (1960–1979): Part 1

Ernest E. ("Lee") Keet
Vanguard Atlantic Ltd.

The author, a participant in packaged software's early days, worked with IBM up through the System/360 launch and then with Turnkey Systems, an early provider of packaged software. This article traces the author's background from graduate school through the Turnkey Systems sale to National CSS in 1979 and subsequently to Dun & Bradstreet.

This article is the first of two extracts from a much longer document that may one day be published as a book. My working title is *Confessions of a Serial Entrepreneur: Surviving the Information Technology Revolution.* I wrote this initially for myself, my family, and friends who were interested in anecdotes about my early days in the software industry. Several friends have encouraged me to make parts of it public, especially Luanne Johnson[1] and Burt Grad, the founders of the Software History Center.

My history with computing goes back to the late 1950s. My trek toward both entrepreneurship and technology began with a youth misspent playing with explosives and electronics, while simultaneously developing a string of neighborhood minibusinesses that let me indulge my taste for gadgets. I had the all-American paper route, sold handmade items door-to-door, ran a penny arcade for neighborhood kids, collected and sold scrap paper and discarded bottles, and did whatever else seemed lucrative and achievable at the same time. I went to Cornell at the ripe age of 16 to study engineering, graduating with a combined BS and MS in 1962.

The gurus of Cornell's engineering computing venue in the early 1960s were Bill Maxwell and Dick Conway. Bill was a teaching assistant on his way to a professorship, and Dick was already a full professor. Dick, well-known in computing circles, later led the program at Cornell to implement a version of MIT's Project MAC. Bill became my graduate advisor. The fifth-year program included a thesis, and based on the programming course I had taken the prior spring, I decided that something with a computing requirement might be fun.

A friend and fraternity brother, Pete Giles, was pursuing a thesis on a theoretical approach to solving a linear programming problem via approximation. Over beers, Pete sold me on taking over the implementation of his team's project because he was going to graduate six months ahead of me. Maxwell did the rest of the selling, and soon three of us fifth-year engineers were, as a team, working furiously at building a computer program to do matrix inversions and other compute-intensive stuff that would result in an optimum shop-floor layout. Pete graduated in mid-1962, and I and my two sidekicks completed the project by working nearly every night and weekend at the computing center on a Burroughs 220, the successor to the 205 Datatron (both vacuum tube machines). In fact, the 220 was possibly the last commercial vacuum-tube machine ever built.

High-level languages were just getting their start in those days. Fortran was one of the earliest, developed starting in 1954 and first released in 1957. Algol followed in 1958 with a major update in 1960. Unlike Fortran, Algol introduced recursion, indirect addressing, and character manipulation, among other features. Cornell had adopted it as the language for use in its computer programming courses, as well as Fortran, because it was a precise and useful way for capturing algorithms.

Our biggest problem was that we were inexperienced programmers, and our Algol routine had to invert a 40 × 40 matrix—10 seconds of work today, but an all-night affair then. This meant that we had little time to store interim results (especially on the unreliable and slow drum), so we tried to cram a six-hour runtime into a six-hour window and frequently despaired as the paper jammed on the 403 or the program failed 10 minutes from completion with no time remaining to get the traces and dumps to study. Only a few years later, I would have known to segment the program and take checkpoints, points from which you can restart in case of failure. Had I known, I could have used two nights productively instead of 20 that ended in total frustration. But, somehow we got it done, and our paper went on to win the Silent Crane and Hoist Award for 1962 along with a check for $25, which my new wife Nancy and I needed desperately.

IBM, which I joined right after grad school, was a great training ground that let me pick up the entrepreneurial thread begun in my childhood. I had three distinct careers at IBM in five short years: systems engineering, application software development, and sales, all three of which would prove invaluable for my next stage of life. I loved IBM; they gave me a postgraduate education in more ways than one. They paid for an added MS (in operations research from New York University) but more importantly gave me the on-the-job-training I needed to understand how things worked in practice. I think I gave them almost as much as they gave me. When I declared my interest in leaving, they did almost anything they could to keep me there, something I remember fondly, as it let me think the bargain had been fair.

By 1967, I was ready to strike out on my own. I could run reasonable-sized technical projects well, had run a large sales territory, and had acquired the sense of invincibility that every IBMer had in those days. Tod Pontius, another IBMer from the Bridgeport office, and I decided to start out on our own when it was clear that IBM couldn't meet the demand for software support and services generated by the IBM System/360 launched in 1964. As part of the early System/360 need-to-know team and with a strong background in manufacturing applications, I was certain that we could sell ourselves profitably to businesses, especially manufacturing companies, wanting to step into the world of electronic data processing.

Turnkey Systems: Origins

Turnkey Systems Inc. (TSI) was founded in April 1967 as a contract programming firm specializing in getting the still-new IBM System/360s working productively for our customers. Our original goal was custom development. In our view, Applied Data Research (founded in 1959, with a first product in 1965), Informatics (founded in 1962), and the handful of others already in the products market were true pioneers, but we were not yet convinced that there was a large market for purchased software. In 1967, most software was still distributed for free by the hardware manufacturers or shared between the authors through SHARE, GUIDE, and other user groups.

We learned that two guys selling on their own was a lot different than selling for IBM. The concept was attractive, but day after day passed without a sale.

Tod and I started the business with $15,000—$7,500 each. We had no customers or prospects, just very naive hopes. We rented an office, hired a secretary, and set out to sell our skills. Months went by, and we learned that two guys selling on their own was a lot different than selling for IBM. The concept was attractive, but day after day passed while we ate our seed capital without a sale.

At IBM there was always someone to take care of the incidentals. Secretaries took your dictated notes and returned near-perfect letters. A guy from Facilities made sure the office was well stocked with supplies. You did your job, and everyone else did his or hers. Perfect. I learned that this idyllic world did not extend to most businesses, certainly not a business with two guys and one Girl Friday. I arrived at work one morning and was told that there was no toilet paper in the ladies' room. After I almost said, "call Facilities," I drove to a store and bought the paper myself.

At long last, we had our first customer: Jenkins Valve. They had a primitive manufacturing control system, using card images on tape. They had manual order processing, no automated production planning, and no link between their manufacturing and financial systems. Their chief financial officer (CFO) had started an ambitious project to bring them into the 20th century and hired a consulting firm to design modern systems, and we asked to bid on the implementation work. Tod and I worked day and night putting together a bid and specifications.

On the day before the bids were due, Tod, our secretary, and I agreed to work until everything was finished, which was a massive amount of typing and document assembly. Around 3 a.m. on the day of our interview, our secretary quit. (I can't understand why—we had let her sleep at least twice that week.) At 8 a.m., I sent Tod ahead with the executive summary and a foil presentation while I did the proposal's final typing and assembly. Neither Tod nor I had slept or changed our clothes in two days. Just before noon I shaved in the office bathroom and rushed to Bridgeport with the final proposal. The meeting was still in session, and we got our message and bid in with some semblance of professionalism, albeit delivered, seemingly, by two Bowery bums. We won the business.

Some of what we did for Jenkins was revolutionary for the time. Orders would come in via teletype, with no fixed formatting. A program I designed and wrote had to parse the incoming character stream, interpret it, and put out fixed-format records for the programs that Tod was designing. This turned out to be a generic and challenging task, because we could only tell what was coming next and how to format it by interpreting the character stream as it arrived. Much later this problem would create a whole subindustry, especially with the advent of electronic data interchange standards.[2] At the time, I had to invent mechanisms to do on-the-fly context analysis and reformatting. Our access to the Jenkins computer was nights only, so we worked nights programming and testing and days meeting with the customer or trying to develop other business. The project lasted almost a year, and by early 1968 we had demonstrated our skills and had sold a few more small jobs. Business was looking up.

As we added people, our business practices became more important. We wanted a family-like culture, with everyone deeply involved in the business success: families as well as employees. We created a stock option and purchase program that included every employee—an innovation at that time—along with an artificial market that let employees acquire stock, trade it among themselves, or in certain conditions sell it back to the company. Later, when away-from-home travel was the rule for nearly all employees, we created a Family Travel Benefit (FTB) that put $10 in a pot for every night spent on business travel. These funds could only be spent to take along a partner or child during some future business trip, and were duly noted on expense reports.[3] Employees would squirrel away these FTB credits and look forward to tacking on a weekend in Orlando or San Francisco. Consequently, travel became in part a benefit, not a burden, especially for the house-bound spouse, and we spent very little extra for a huge payback.

One of our early customers was Burndy, a manufacturer of electrical connectors. Jerry Kaufman, their vice president of data processing, hired us to develop an order entry system for their new System/360 Model 50, using 2260 display terminals. The 2260 was the first "online" interactive terminal for the System/360, and Jerry thought it could be used to replace keypunches, eliminating errors and delay. Jerry was a very smart guy, impetuous, and a martinet, but a warm human being.[4]

In the late 1960s (from the introduction of the System/360 in April 1967 till around 1970), the software to support online transaction processing—that is, the systems software between the application program and the hardware—did not exist. Time-sharing systems and special operating environments had been developed by IBM and others, but there was no IBM solution for running one interactive program to serve multiple terminal users. Time-sharing is a different solution to the problem, and an inelegant one at that. If hundreds of users are running the same program in separate time "slices," the overhead is enormous compared to letting one resident program serve the multiple users while keeping (only) their data and flow controls separate. Just interacting with the terminals within the commercial operating systems of the day was a chore.

IBM did distribute a primitive control routine that let application programs (written in assembly language, not high-level languages used by most businesses) communicate with the 2260 display terminal, sense an incoming message, and transfer outbound messages. Further, to make a terminal display, a message required detailed instructions that told the terminal how to format the text (color graphics were still off in the future) and precisely how to send it.

My team of developers at Burndy, led by Kathy Emerson, were application people, not systems programmers, so there was no real choice except to develop our own "middleware" to let high-level programs interface to the IBM 2260 terminals. To do this, we built a shell in Cobol that other Cobol applications could invoke when they needed terminal services—for example, to send a message to a terminal in a format acceptable to that device (versus some other type of device) without knowing the device protocols or operating system interface. The interface mechanism was straightforward: First, the programmer filled out a table with the name of each field in the message and its display characteristics (bold, flashing, protected, underscored, and so on).[5] Then the programmer "PERFORMed" a paragraph of code that we supplied that made the actual external call and returned to the next instruction on completion. Everything else was done externally, in our shell or by the operating system resources that our shell invoked.

Jerry was delighted, the application was a roaring success with its users, and we had implemented one of the first online application systems in the country. Our reputation for skill at building complex systems grew, but in fact the shell and standardized interfaces we had built at Burndy let us hire garden-variety Cobol programmers and set them to work writing complex online applications. What's more, Jerry saw that he could use our shell to develop other online applications, and we worked out a license so he could do just that.

Turnkey's first 'package'

It was becoming obvious to me that we should try setting up a software products business to see if we could learn how to resell a program. By this time we were convinced that there was a market for software products if properly packaged and supported. Larry Welke's International Computer Programs catalog in 1969 started to list packaged software along with user-contributed software.[6] Although packaged software was less than 10 percent of the listed offerings, it was growing rapidly. (In his ICP catalog summary, Larry says that by 1973 it was up to 49 percent). We thought we saw a trend, so we agreed to package what we had built at Burndy.

In 1969 we finished our product, which we named Graphics. Like the project work at Burndy, it was an OS-only, 2260-only support package. It let programmers write Cobol programs that could issue CALL statements to handle all the communications with the display terminal. It let an ordinary Cobol programmer do programming that would have been impossible without deeper system skills. Without Graphics, it all had to be done in assembly language, and programmers had to know how to format character strings to send to the device.

Bob Bouton, who had joined Turnkey as vice president of sales and marketing (but who in truth had no one working for him), and I set out to sell this product to anyone who answered a tiny ad we placed in *Computerworld*.

From a business standpoint, Graphics was a disaster. We sold about four in 1969, and in the process discovered that we had addressed only some of the technical and marketplace issues. A software package had to support multiple operating systems (for IBM systems at the time, this meant OS and DOS). It had to have scalability, which meant multitasking capability. It had to support multiple programming languages, terminals, and environments. It had to handle the exclusive control of records, avoiding the possibility that I could read it, you could change it, then I could overwrite your change. We had built a prototype, not a product.

Multitasking versus time-sharing

The System/360 hardware and operating systems had introduced the idea of multitasking to IBM customers, allowing one program to run while another was waiting for a slow-playing event to finish.[7] Multitasking differs from time-sharing, in which numerous different "virtual machines" are swapped in and out of the active memory on a time-sliced basis, giving each user his or her "own" machine for a few milliseconds. With multitasking, several programs are operating at once, each one in its own protected space, but the interrupt system allows control to pass to a program that is ready to process and then return to the original program after a slow-moving external event finishes, such as reading a disk record.

However, within a single application program running in its own protected space, there was no way to make this switching happen. To write a single program that would allow multiple terminals to do order processing—and achieve multitasking—required that each terminal have its own copy of the program running separately. This was clearly unacceptable for more than a few simultaneous users. Even if it had been acceptable, there would have been no way to prevent the simultaneous-update problem, that is, A reads a specific record, then B reads that same record, then A updates the record, and finally B updates it, wiping out what A had done. Clearly, some level of control was required within the partition that was running the application.

Pseudoconversationality

To solve this problem, we invented something that subsequently became known as pseudoconversationality, which is not really multitasking. Pseudoconversationality made our shell or "telecommunications control program" look to the operating system like a single-thread batch program, but a single application that could internally host multiple terminal threads. For example, within our shell, which occupied only one address space, many terminal operators could use the same program, each entering or retrieving their own data and maintaining their own flow control. When one of these users, or threads, wanted a time-consuming system resource, like data from an external device, the control program

In January 1970, we launched Task/Master, our first true software package and, to my knowledge, the first commercial telecommunications monitor.

would issue the request and remember which thread had asked for the service.

At that point, the operating system took over, allowing other tasks to operate while the service was being performed. On return of control to our partition, the thread that made the request would be allowed to execute to completion and then (and only then) would transfer control to any other thread that was ready to run, without the return of control to the operating system. It was an ingenious but simple piece of technology. It let one program manage many terminals and kept the application from holding up the whole system. Unfortunately, it did not prevent one terminal from holding up another terminal, because when any one of these pseudothreads demanded an external service, the operating system took over and passed control to an unrelated application in another address space. This turned out to be a problem requiring a clever solution: finding a mechanism to let many users thread through a single application without control being relinquished to the operating system until there was nothing more to process.

Four horsemen

When we first set up Turnkey's software products division in 1969, there were four of us who did everything—design, programming, documentation, testing, packaging, sales, and installations.[8] I had hired well to staff this new division, bringing on three young guys with great technical instincts if not honed skills. By far the finest programmer I ever met was Steve Ward, an early recruit. I then hired Dennis Sisco, just out of the army, giving him his first real job after college. Joe Farrelly became the fourth horseman, the solid, "hey guys—not so fast," anchor for the team.

Steve came on board with no Cobol knowledge. On his first day on the job, I asked him to construct a source-code generator that would let us distribute unique (to customer specification) source code from a master "deck" of Cobol code. Worse, I asked him to write this program in Cobol, that is, Cobol generating Cobol. This was on a Thursday, and I gave Steve all of our Cobol manuals along with a self-study program. I asked when he thought he could be ready to start, and he said, "I'll try to have something to you by Monday." I said "What?!" He said, "I think I can at least get you a working prototype by Monday." Sure enough, on Monday Steve gave us a finished product that with almost no revisions became our method of custom distribution for four products over the next decade.

Steve illustrated an oft-repeated truth, that programming is even more of an art than it is a science. Programmers should be educated, and trained, but even educated and trained programmers (like me) can be average at best. Donald Knuth of Stanford University, the author of the seminal work *The Art of Computer Programming*, estimates that only one out of 50 college undergraduates can become truly good programmers. Steve was one out of thousands.

Dennis was almost as unusually creative. Somehow he understood system architectures without having ever studied system architectures. A lot of the elegance of the mechanisms in Task/Master, our major product, came from his fertile brain.

Joe was a different sort entirely, but without him we would not have made it. He knew when we'd bought our own hype, had "overdosed" on lack of sleep, and when a task wasn't worth the effort. Joe worked as hard—maybe harder—than any of us, and he kept us sane.

Our team's mantra became "promise, then build." We saw that what we were selling to Company A led to enhancements that Company A needed but that could also induce a sale at Company B. We were always running to meet commitments. With the optimism of youth, we were always selling the system with a few more features than it actually had and then rushing to upgrade it in time to make the delivery. We sometimes got hoisted by our own "vaporware" and had to fight to keep up, but the product steadily got better. I don't think we ever disappointed a customer, and we never announced anything that we didn't deliver.

Launch of Task/Master

In January 1970, we launched Task/Master, our first true software package and, to my knowledge, the first commercial telecommunications monitor. We started to sell the product nationally, but before the end of 1972 we had signed up several distributors in Europe. By 1974, we were getting a third of our sales in software products from outside the US.

American Tobacco and multitasking

We sold one of the early versions of Task/Master to American Tobacco (later to become American Brands). Although initially we had announced Task/Master as a multitasking system, we were actually still using pseudoconversationality. As noted, with pseudoconversationality there was no overlap of I/O requests from various threads sharing one online application. True multitasking would have let the various terminal and database actions overlap. Pseudoconversationality was fine for American Tobacco's purposes because true multitasking isn't necessary unless a site is running many simultaneous transactions per second, and they weren't even close.

Nonetheless, one day their data processing manager called and said, "We did a test and it's not truly multitasking." I said, "Oh, didn't you get the service bulletin about that?" And he said, "No, I didn't." So I created and sent off a backdated technical memo with a bulletin number suggesting that it was one of many in a series, far from the first (which it actually was). It said multitasking, which had been disabled in the current release because of a serious design flaw, would be repaired in the next release of the product. To deliver the multitasking, a crew of us immediately went to Oxford, Connecticut, where we rented computer time from Uniroyal, and spent long weeks developing the first multitasking system for terminal management. As soon as it was done, we raced it over to American Tobacco, and later sold it to others.

The fact that we programmed multitasking in Cobol still amazes and amuses people in the know. Cobol is not an especially efficient language—it generates many more machine-level instructions than would be the case by a programmer writing at a lower level. To use it for an operating-system-like control program was either foolhardy or brilliant: foolhardy if the performance was inadequate; brilliant if the performance was acceptable, because it created a truly portable product that could be customized to the customer's needs at the source level. In retrospect, after much fine tuning of its performance, it was a brilliant success.[9]

Terminal independence

In addition to true multitasking, we added all of the tools needed to make it easy for Cobol programmers to write online applications entirely in Cobol. Among its other features was something we called terminal independence. Terminal independence was a notational means of specifying what we wanted the display to look like, a predecessor to the later markup languages like HTML and XML. The programmer provided a table of attributes and our device-specific terminal independent modules (or TIMs) inserted the correct control characters so that the image on one device looked the same as on another device. Based on each terminal's address, Task/Master knew which TIM to invoke to translate each message. Our growing library of TIMs was one of the strong selling points of Task/Master.

IBM unbundling

IBM's June 1969 unbundling announcement did not create the software products industry—companies like Informatics and Applied Data Research had been in business for years[10]—but it did accelerate its growth. Unbundling forced customers to pay for software, and in that situation a customer's natural reaction is to shop around. My first reaction to the unbundling announcement was that it would create even bigger opportunities for custom contract programming and, later, for systems integration services, because IBM had also announced that all engineering services would be charged on an hourly basis. Of course, I was right but missed the bigger implications, which came to me only with hindsight—that is, we finally had a truly competitive market for software. The customer had to start justifying expenditures for software, IBM's as well as ours.

All of our customers were IBM users. IBM at first ignored little software companies like us. But after unbundling, they began to see us as a threat to revenues and to "account control," especially in the database and communications software areas. As a result, they poured money into these products, trying to ensure a total monopoly. In retrospect, they clearly succeeded. Along the way, all that the independents wanted was, first, more information on interfaces so we could keep our products current and competitive and, second, less use of Fear-Uncertainty-and-Doubt (FUD) by their sales folks. IBM's tools people had a real inside edge in knowing what new tools would be available in its operating and near-operating systems—all we asked was that they create a Chinese Wall and pass those goodies to the outside world at the same time they passed them over the wall.

If this sounds familiar in the Microsoft context, it is. FUD was more complex. IBM's sales force could allude to new products without any documentation or specs, new products that were of course better than the competition, and their dominance would cause many to simply do nothing. As was often said at the time (here's the fear factor), you would never lose your job if you bought from IBM. Fortunately, many IBM customers saw value in both our products and in not being totally dependent on IBM. Although this was a minority of companies, it was enough to build a business.

CFO approvals and the changes unbundling brought

Even when we first started selling software in the early 1970s, to get a sale we would typically need the approval of the company's CFO—and sometimes that of the president—simply because buying software was so unusual. Common questions were, "What is it we're buying? A software package? What is that? Why can't we just do it ourselves?"

Before unbundling, the software and hardware decisions were generally made together. Once IBM started pricing software, the CFO didn't want to see every transaction, because individually they were relatively small, but system purchases still required high-level approvals. Independent software decisions, however, changed that pattern, as $15,000 or $20,000 purchase decisions for software would not typically rise to the corporate level. As a result, one of the price constraints that we became very sensitive to was the level in the average corporation at which we had to go to the board of directors, CFO, or president for purchase authority.

During the first five years we sold products, we believed this number was $25,000 or less. John Maguire at Software Ag of North America changed all that when he announced that Adabas—a database management system (DBMS)—was going to be priced in North America at more than $100,000. We all gasped, and some laughed, but John proved us all wrong and can be credited with helping to make software product sales a profitable business. He was willing to let CEOs and corporate boards debate a purchase that he claimed was "strategic" and which they slowly came to see as being part of the lifeblood of their businesses. Indirectly, this led to the elevation of the lowly data processing manager's position to the current role of chief information officer and to the re-delegation of major technology purchase decisions to this new, better-educated, and far more powerful executive. Everybody won.

Customer Information Control System

We sold Task/Master in competition with the other independents and with, most importantly, IBM. IBM had adopted the Customer Information Control System (CICS), first developed by IBM and Commonwealth Edison in Chicago as a Type III program (meaning that the program was contributed by a customer or IBM support person), as its anointed communications monitor. IBM slowly improved it over the years, making it more efficient and adding a usable Cobol interface, but at the time of unbundling it was a terrible product. Despite IBM's continuing and huge investment, CICS remained an inferior product for its first decade of existence, and it never became an easy-to-use tool. Both systems and application programmers struggled with it, and it used lots of hardware resources. As it turned out, this was all irrelevant, because—having a total monopoly—IBM now has 499 of the Fortune 500 as CICS users and has sold more than 30,000 copies.

Task/Master versus CICS

During its lifetime, Task/Master was a much better decision for the customer. It cost less, worked on all IBM platforms, was distributed in source code,[11] offered terminal independence, let programmers work quickly in high-level languages, and required far fewer machine resources. However, IBM was Big Blue, and many corporate decisions were made in IBM's favor by default. We advertised in the trade journals and in the *ICP Quarterly*, with a simple ad that simply said "Before you buy CICS check out Task/Master." In response to queries, we sent out a checklist of yes/no comparisons of the features and advantages of each package. We had, of course, developed it ourselves (with a high degree of bias) and then had the immense good fortune to have a lazy journalist adopt it for an article without checking for accuracy or completeness or asking IBM for a rebuttal. With this "independent" analysis of the two products, we did battle with Goliath. Our willingness to do anything to get the sale, from free trials to money-back guarantees, combined with endless hours of travel and late nights, made us the name to beat in "independent" software to run a company's network.

In a fair fight, the only long-term differentiator is quality. With enough quality of design, production, or service, someone can charge premium prices and still win against lower-priced competition—but only if the other guy isn't monopolizing power or plays fair. A player with a monopoly who doesn't play fair always wins, which is why I own Microsoft stock.

IBM's DL/1 ploy

In 1978, IBM finally caught and defeated us, but not by playing fairly. IBM announced that DL/1, its database system for DOS, would in the future support online access by using the master scheduler from CICS. This meant that if you wanted to use DL/1, you had to buy CICS. Doom.

I'm no lawyer, but that seems like an incontestable violation of the tie-in sale prohibitions of the Sherman Act. It destroyed our market along with that of all our competitors. Even if the customer didn't want to buy DL/1, he saw the handwriting on the wall. IBM was making CICS its strategic product for online application development. IBM had about 40 percent or 50 percent of the telecommunications monitor market, and the rest of the market was distributed among several companies. We were number two. We had sold about $25 million worth of Task/Master at that point. IBM's action did not affect our competitors so dramatically, because they sold their telecommunications monitors mostly to customers for their own DBMSs.

DBMS vendors

Turnkey Systems never had a DBMS, which was one of the earliest general-purpose software applications. Informatics brought out the Mark IV system in the mid-1960s. It wasn't strictly a DBMS but rather an integrated data storage, retrieval, and reporting system, but it proved that standard systems could be sold on a repetitive basis. Tom Nies founded Cincom Systems in 1968 and in 1969 introduced a DBMS he named "Total." Software Ag, a German company, had launched Adabas first in Europe (also in 1969) and then in the US through a subsidiary run by John Maguire. John Cullinane took technology originally developed for General Electric and in 1973 generalized it into Cullinet Systems' Integrated Database Management System (IDMS) product. Soon after, Marty Goetz[12] introduced Applied Data Research's Datacomm product.

During the early and mid-1970s, Datacomm, Total, Adabas, and IDMS competed fiercely with IBM's larger information management system (IMS) and smaller DL/1 offerings, with great success. Each company added a telecommunications monitor to complement their DBMS, but Task/Master remained the only product unaffiliated with a DBMS, something we made a huge point of in our marketing. We were the only player in the market that let companies choose the DBMS and telecommunications monitor separately, which meant that we got a disproportionate share of the IBM shops that thought CICS was a piece of junk but didn't want to commit their data to a non-IBM supplier's product. Hence, our disproportionate share of the market with IBM DBMSs sealed our doom when IBM linked its two products.

As a result of our Hobson's choice—sue IBM or watch our profitable Task/Master business become a money-losing cost sump—we chose to exit the marketplace.

Moving on

As a result of our Hobson's choice—sue IBM or gradually watch our profitable Task/Master business become a money-losing cost sump—we chose a third path. We had a closed-door meeting with our board and decided that we were going to exit the Task/Master marketplace as gracefully as we could without hurting our customers. This was not altruistic: We had many "leases" and a very nice maintenance business from those who had paid for a one-time license. We made an announcement that we would service customers forever if they stayed on maintenance or under lease, and that we would keep the system current with all operating system upgrades. But in 1978 it essentially became a defunct product.[13] Although I still view IBM as an ethical company that simply made a bad mistake, Task/Master died a premature and unnecessary death.

My advice to anyone in a similar position: Anticipate your demise. Identify potential competitors who will see your success and respond with better marketing, newer products, more coverage, illegal actions, or by creating enough FUD to stop you cold. Be sure to look in adjoining markets, not just the one you propose to capture.

Finances

Finding capital to finance the growth of the industry was one of the common problems that we all shared. No one would invest in software, and no bank would lend to a software company, other than perhaps to factor its receivables. Bankers could not fathom a business with no bricks, mortar, or inventory, and the venture community—then in a nascent state as far as technology was concerned—could not see through the risk of the intellectual assets all walking out of the building every day.

Cash was our constant worry. We strung out (but never stiffed) our suppliers, hounded customers for receivables from the moment our invoice was warm, sometimes went without pay, and frequently made payroll by granting personal loans to the company. For the first three years of Turnkey Systems' existence, neither Tod nor I took a regular salary, living off what little savings we had. Even when the software products side of our business was doing well, my pay never rose much above the levels I had earned many years before at IBM.

Financing 'leases' for software

To attack the cash-flow problem, I created renewable "leases" for software, an innovative solution passed on to many other companies. Our prospects wanted a monthly use fee that would let them get below the purchasing authority limits. Although some customers really wanted a month-to-month low obligation lease, most were willing to sign up for a longer period after accepting the system. The key was winning the customers' confidence that the system was functional and stable. Once they were convinced of that, after an initial trial period, they could see the inevitability of the system becoming integral to their operations, so a long-term lease was not unacceptable. So we adopted pricing policies where Task/Master customers could either perpetually license the product or lease it, with the incentives biased toward leasing. In reality, these were monthly licenses to use the software for a fixed period of time, and at the end of the lease we had built in the customer's choices (renew, extend, or buy a perpetual license[14]), all of which resulted in high-margin revenues. We did this first in 1970, and I am almost positive that we were the first in the industry to do so. However, because we needed cash, writing the license was only half of the solution.

I arranged a series of bank lines that would lend against these multiyear leases. We persuaded the customer that we were financing their license and that therefore our service obligations should be separated from the lease. We achieved this by giving them a separate service agreement. They agreed to pay us the monthly fee for three years, with the small type making this a requirement regardless of our ability to service them. We covered the obvious sales resistance with money-back guarantees during the first few months of use, but thereafter we could borrow against the balance of what are called "hell or high-water" leases.

Software pricing strategies

One other advantage to this technique was the flexibility it gave us in pricing our software. Because maintenance for perpetually licensed software was typically 15 percent of the then-current purchase price, and because we had monthly licenses whose fixed pricing ended after several years, we would always raise the price aggressively each year. This let us do "buy quick; the price is going up" selling at year end and also enjoy higher maintenance and renewal fees. Of course, we would then offer a "special" discount to new buyers, which let them think they had a bargain for a while but also minimized complaints at renewal time.

Another rule learned from experience was to set prices high, then raise them—then, and only if unavoidable, to use pricing tactics to get back into a reasonable competitive zone. We let customers know that our products or services were worth the extra money but that we offered one-time new account incentives, money-back guarantees, financing over time, or other means that made the purchase more affordable to new customers.

One of our first customers, Atlantic National Bank, signed up for three successive three-year leases and then finally bought a license for the product, ultimately paying us more than $70,000 for a product that they could have purchased at the outset for $15,000. And they're a bank! As I will explain, when Turnkey Systems changed hands years later, the lease portfolio was its most valuable asset.

The leases solved a big marketing problem and let us get sales at the expense of our competitors, but even with the ability to finance 80 percent of the lease we had to produce up-front cash for every sale. Our sale price was, for sake of illustration, $25,000.[15] Our lease price was $600 per month for 36 months. Of the $21,600 face value, we could borrow $17,280. But this borrowing occurred typically after a three-month trial period during which the customer could cancel. This meant that compared to an outright sale we had to self-finance $25,000—$17,280 + 3 × $600, or $9,520. This was a huge amount for us, so although we loved the residuals in a lease, we could hardly afford to write one. Sometimes the choice was to lose the sale, however, so we went hungry and built a lease portfolio that eventually was worth eight figures and became the main asset of the company.

The lesson I learned, which cannot be

stressed enough, was to be a gatherer, not a hunter. One-time sales are thrilling, but you must make more and more to keep your revenues rising. Financing should be available for solid longer-term customer commitments, and on the day that your recurring revenue stream exceeds your fixed costs you will make this rule your mantra.

Cash shortages and the pressure to sell out

Cash shortages persisted. All of our capital went into the lease portfolio or back into new products or market expansion. Much later, perhaps in 1977, when the venture market was turning toward technology companies, I asked Russell Carlson, who was running Citicorp's venture operations, to address my management team to describe the future funding opportunities we might see. In the Q&A session, Russ was asked what advice he would give a company like ours, growing at a reasonable pace but always cash constrained. "Sell," he said. Everyone, including me, was shocked by his answer, thinking of ourselves as survivors—but sell was eventually what we did.

We bootstrapped Turnkey Systems for 11 years. Most of our peers did the same. There were exceptions: Informatics raised its initial capital from its first 10 customers and went public in 1966; Cullinane raised $500,000 from Wall Street in 1968, the same year ADR went public. Most stayed private and cash strapped until the 1980s when software was something the public had heard of, leading to a rash of public offerings.

Services versus software

Our colleagues selling services found the going easier. Because professional services are not cash intensive, our colleagues could bill as they worked. Software, however, is capital intensive, requiring up-front investments in research and development, marketing, and support. There was little overlap or similarity between the service and software firms, although some of us did both (mainly to pay the bills). Software customers wanted trials, return periods, and monthly payment plans. With no financing for startups, no venture money, no loans, no contract financing—not even receivables factoring—there were only two sources of funding: from personal resources and from sales. This made every sale crucial, heightened the competition, and ensured that the principals would be in the fray each and every time. We saw each other across the table often, got to know each other—perhaps a bit like Wyatt Earp knew Billy the Kid—and got to respect what the other entrepreneurs did well, whether that be sales, marketing, distribution, or simply building good stuff.

The lesson I learned, which cannot be stressed enough, was to be a gatherer, not a hunter. One-time sales are thrilling, but you must make more and more to keep your revenues rising.

Key/Master and Docu/Master

In the early 1970s, our strategy was growth, to be accomplished by surrounding Task/Master with online applications that leveraged Task/Master's features. The first such product, called Key/Master, was introduced in 1974. One of our Task/Master customers, Chrysler, had developed a data entry program to replace keypunches and key-to-tape machines that let operators enter forms rapidly, directly into the mainframe computer. Their application kept track of operator statistics, let key portions of documents be re-keyed to assure accuracy, and included some check digits, control totals, and other means of monitoring quality. This looked to us like an application that we could sell widely, so I paid a visit to Benny Lin, the Chrysler data processing manager, in 1972. For $10,000 and a guarantee that Chrysler would have the use forever of whatever derivative product we produced, he sold the rights to me.

It took us two years, several hundred thousand dollars, and a complete rewrite to build the first Key/Master from the Chrysler idea, but when we launched it, the market was receptive. Online applications were becoming widespread, and the multistep processing, varying standards, and inaccuracies of other key entry systems yielded to the online lookups, immediate availability of the data, and other niceties that we built into Key/Master.

Many of the applications that data processing departments implemented in those early days of online systems were glorified data capture programs—enter an order and look up inventory, for example. We added user exits where unique routines could be added to

My prior experience selling to Americans screamed for an aggressive attack on the market, but my insensibility to the English class system and culture slowed me down initially.

Key/Master, and a number of our customers built complete applications using the product. Because it ran under Task/Master, migrating these customers from a simple capture-and-display application to true online processing was a logical upgrade path and source of revenue for us. Key/Master was subsequently interfaced to CICS, and when we finally abandoned Task/Master, our Key/Master revenues continued to grow. Many think that basic keypunching died years ago, but that's not the case. When TSI, then renamed Mercator Software, was absorbed into Ascential Software in 2003, Key/Master revenues were still close to their level of 10 years earlier.

In 1975, we added Docu/Master to the TSI repertoire. A friend from Sweden, Martin Leimdorfer, ran a company called Industri-Matematik. IM was in the logistics business, but as a sideline, the ever-peripatetic Martin had acquired and packaged a full-text storage-and-retrieval system he called IM-DOC. This was out of IM's mainstream, so he agreed to sell it to us for future royalties. We saw it as a logical extension to Task/Master and as a better way to compete with our peers who were moving from database management into communications. We decided to go the other way, but to focus on unstructured data.

IM-DOC—which we renamed Docu/Master—needed reprogramming and repackaging to become a software product, but it too was an instant success on launch, giving us a good set of applications with substance. Unlike a structured database system, Docu/Master produced a full inverted list of all the words in a document, minus a list of "stop words" of no use in retrieval (such as "and," "the," and so on). This let users structure queries on the fly—such as asking for "white" and "house" adjoining and in that order, with "Washington" or "President" also somewhere in the document (to avoid finding articles on how to paint your house white). For large text databases, like the *New York Times* morgue files, this became a powerful application. With Task/Master and Key/Master, Docu/Master gave us a third arrow in our quiver. For this reason, in 1978 we could afford to write off new sales of Task/Master, roughly 20 percent of our business, using the Task/Master maintenance and lease stream and the sales growth of our other products to take up the slack.

Distribution, and my days as a Brit

By the mid-1970s, following the introduction of new products, Turnkey Systems decided that the time had come to make our network of distributors in Europe into a profit producer. Up to that time, the cost of providing US-based support personnel to European distributors had consumed all the profits, so what we gained was volume only. Volume was good, because it let us afford more R&D, but we wanted to make some money, and thus far had made almost nothing in eight years of trying. Yes, we had a nice and growing business, but everything was reinvested, so—no profits.

Distributor economics

Our distributors were mainly small independent businesses that were still experimenting with software sales. It was pretty standard in those days to split the sale price with the distributor. We took 50 percent without giving much in the way of sales or technical support—that was paid for out of the distributor's half, as was translation of materials, local marketing, and so forth. This formula was destined for failure from the start, although none of the software companies knew it at the time.

Later, we learned that sales cost roughly 25 percent of a typical selling price; marketing, 15 percent; and customer support, 20 percent. In short, in a 50/50 deal, each sale properly made and serviced would lose the distributor 10 percent of the sales price. Later, the split changed to 60–66 percent for the distributor, letting both parties make some money. But even more important, it became clear that decentralization of marketing and customer support to the lowest level made the pie too small to share profitably at any level. My solution was to create an interim level of support to do "template" marketing and to provide second-level technical support and training through a TSI-owned office near our European distributors.

Setting up a European base

The plan was for me to move to London to set up a technical and business support center, to promote Dennis Sisco to run software products operations in the US, and for Tod Pontius to turn the professional services side of the business into a national presence, with profits everywhere. It didn't turn out that way, but my move to London was fun, educational, and challenging.

Rich Atwood had worked for TSI in the states as a customer support manager, and he volunteered to join me in London, managing central technical support for our distributors. Our English distributor, Hoskyns Systems, was exiting the software business so we had a territory of our own to support as well, something that could serve as a model for future remote offices.

My prior experience selling to Americans screamed for an aggressive attack on the market, but my insensibility to English culture and to the English class system slowed me down initially. One of the first things I did was use modern word processing techniques and a database I took great pains to build—both of which were rarities in 1975—to write personal letters to every data processing manager in the UK. I introduced myself and invited each one personally to a seminar intended to introduce our products to the UK market. Result? Not a single reply. In the US, we expected a 5 to 10 percent reply rate to the first of such mailings.

What to do? I conferred with a few people, who assured me that the English were so class conscious that they would never reply to a sales invitation, assuming it was beneath their station to attend something so crass. So I tried having engraved wedding-like invitations made, complete with a filmy overlay on which was listed the agenda, and a small, engraved, reply card. The card invited them to an invitation-only seminar on new American technologies to be held at the Inn on the Park, complete with a gourmet luncheon and featured speakers (me included). A £25 fee was requested, by check, "to defray the cost of materials prepared for, and to become the property of, each attendee." A reply on or before a certain date was requested as well. The same 660 people I had previously contacted now received these pretentious invitations, and this time more than 200 responded, with checks!

We had to turn away all but 40 people for the first seminar, which only increased the seminars' appeal, and eventually we entertained more than half of the buying managers in the UK. Because it was held for senior data processing managers only (we refused any and all substitutes), the exchanges over lunch were peer-to-peer, enhancing the attendees' good feelings about having come. Our sales pitch was in the clouds, describing the benefits of multitasking communication protocols, terminal independence in software construction, and the relative benefits of various software strategies. No uncouth "buy our product" pitches. The approach worked beyond my wildest hopes. In the first year I sold more than $1 million of software in the UK.

However, our distributors' financial state was still precarious. They were losing money on every sale, and we didn't have the financial strength to support them. In fact, we pressured them for payment almost on shipment, and they pled that they could not collect quickly due to trial periods, demonstrations, and long collection cycles. All true—in Europe, it wasn't unusual for the day's sales outstanding[16] to exceed 150 days, versus under 90 in the US, so our expectations were different.

My return to a company in turmoil

When I returned to the US, at the end of 1975, the business was in turmoil. Tod had taken a project in Florida at MSCU, a Tampa-based credit union, and his (then hidden) goal was to use this project to create an application software product for credit unions. His original bid to create a complete online, secure system for all the credit unions in the consortium had been $105,000. My questioning of this—prior to my having left for London—had been, in retrospect, cursory. I was assured that it was a time-and-materials contract; that the quote was an estimate, not a fixed fee or an upset price (a price at which you renegotiate); and that everyone was happy.

Shortly before my return to the US, I discovered that the client was pressuring for completion and was balking at paying more than the $105,000 (although had agreed to an add-on at another $20,000). I asked Tod and his project manager, Elaine Diefenderfer, if they couldn't pull the time-tested "change of scope" renegotiation, and the truth all came tumbling out. We were indeed working against a fixed fee, the client had already refused several times to pay a penny more, and Tod and Elaine were continuing to pay their team of seven despite having no funds and no way to divert these folks to other projects without the risk of being sued. Tod said that at this stage the only option was to finish the project and end up with a product we could sell to others, so we just had to hunker down. That meant taking the meager profits from our software business and plowing them into the as-yet-undeveloped MSCU software, something I might have considered under other circumstances.

I had built a profitable, rapidly growing software products business and did not want to see it fail. Tod refused to simply walk away from the MSCU project, so we were at an impasse. I called our lawyer, Woody Knight, one of the best attorneys I ever met, who over lunch suggested that the only course of action would be for one of us to buy out the other. I offered Tod a sum that seemed immense at the time, $500,000 as I recall, to be paid over many years, and asked him to counter-offer. He folded, and Woody drew up the papers.

And so our partnership ended. I was now the CEO, formerly Tod's titular position. My first task, which might have rendered the company's debt to Tod worthless, was to visit Jim Berryhill, the head of MSCU, to tell him we could not finish the project for the fee negotiated. I requested a meeting with his full board of governors, representing the heads of all his member credit unions. Over dinner the night before, I gave Jim, whom I had never met before, the bad news. If we were to try to finish the project for the committed amount, we'd go bankrupt before we got there, taking a profitable software products business down with it. However, there wasn't enough margin in the software products side of the business to make continuing worthwhile without a renegotiation, so if they insisted that a "contract is a contract" we would simply fold our tent. On the other hand, I was willing to continue at cost and if we did manage to finish and could sell copies of the software to others, we could make up the difference in royalties.

Renegotiation versus bankruptcy

The next day, I waited in MSCU's lobby for hours while the board convened. I could hear occasional sounds of heated conversation, but no words. At length, I was invited in and I repeated the pitch; go to time-and-materials at cost going forward, build a real software product, and maybe get back royalties—or sue us and we will file for Chapter 7 bankruptcy. One of the managers said he didn't believe me: that I wouldn't throw away the whole company over one customer. I said that between an unknown liability and the $500,000 owed to Tod, if we continued I'd rather write off the nine years and start over, but that if they would work with me I'd try to keep the remaining costs to a minimum and do my best to make the software resalable. I think the success that I had had with other products helped sway them, for in the end we tore up the fixed fee agreement, and our people continued on the project, at cost.

It took us another $125,000 and almost a year under this arrangement to finish a product we called the Automated Information Management System (and styled, with asterisks, as A*I*M*S), and in the teeth of a recession we tried to sell it to other credit unions. I learned a lot about application products in the exercise, but never sold a single copy. The market was simply not ready for us; had we been willing to hang in I think we could have made a success of it, but by that time MSCU and its member credit unions were very happy customers and there was minimal pressure on us to continue. It was now 1978, and I had other opportunities and problems, so we put A*I*M*S on the back burner, for a while we thought.

Mutiny

While I had been in London, Dennis Sisco had been running the North American software products operation, and I let that continue while I worked on the MSCU problem. Dennis, still a good friend, was (and is) an intense, no-nonsense guy. At that time (1975), the managers of sales, marketing, development, and customer service all worked for Dennis. As one, they all mutinied—Leslie Kelley, Joe Farrelly, Bob Bouton, and Bill Clifford, all of whom went on to bigger and better careers after they left Turnkey Systems.[17] I knew the problem was because of Dennis' relative youth and his unbending management style. I had no doubt that had I backed him and had we been financially sound we would have come out of it better than we went in. However, the rest of the team were pretty good at their jobs, and the idea that sales, marketing, development, and customer service would be leaderless was daunting. Especially since our bank account was near zero thanks to the MSCU debacle and paying off Tod. One of my hard-learned lessons is that a manager will never regret making a personnel change once it's clear that a person can't be managed, mentored, or educated to do what they were hired to do. Dennis, however, was a special case who I strongly felt should be saved for the company if at all possible.

I decided to remove Dennis from operations, expecting he might quit on the spot. I restated my confidence in him, told him that this would be a great learning experience, and offered him a new job, as vice president of corporate development with the express goal of finding us a merger partner. By then I had pretty much concluded that we were late in the consolidating cycle of software companies and that our best bet was to join someone with deeper pockets. Surprisingly, and to his credit, Dennis accepted.

Selling the company

Our board of directors at the time consisted of Dennis, Leslie, me, and two outsiders: Stewart Greenfield and Dave Fehr. Stew had started the Citicorp Sprout funds and was by then a well-known venture capitalist. He was trying to raise $25 million for a new fund, Oak Investment Partners, but the tough financial times had him cooling his heels. Although Oak subsequently became a series of multibillion dollar funds, the first $25 million took forever to raise, so we had the benefit of Stew's counsel for several years. Dave was a VP at National CSS, a successful time-sharing concern that had built its entire business around IBM's large computers (initially the IBM 360/67) running under an operating system that created "virtual machines" in time-sliced chunks.[18] National CSS had taken IBM's VM operating system and extended it in proprietary ways, creating VP/CSS. This let their clients share computing power across phone lines with reasonable efficiency if at great cost (the PC would destroy this business). I had invited Bob Weissman,[19] National CSS's CEO to join our board, but he pled time constraints and offered Dave. National CSS was interested in turning VP/CSS into a software product, so it seemed like a good learning opportunity for Dave.

In any event, Dennis briefed the board on his new charter and they fully supported the idea of finding a merger or acquisition partner. We didn't share any of this with the rest of the senior management team, and Dennis' estrangement from them helped keep inadvertent disclosure out of the picture. In the meantime, Dave Fehr told me that National CSS might be an interested acquirer, and that if we were interested, he would recuse himself and get someone else to do the analysis and, if appropriate, the negotiations.

National CSS had lots of attractions. First they were a local firm—we were in Norwalk and they were in neighboring Wilton, Connecticut. Second, they wanted a foot in the software products marketplace, where we were firmly established with worldwide distribution. Third, they had a large client base consisting mainly of corporate users, and the supposition was that Turnkey Systems could mine this prospect list. Fourth, they had a time-sharing service that we had tried and wanted, but could not afford.[20] Last, they were a cash-rich publicly traded company.

Dennis was a star in his new role. He made Turnkey Systems look like something precious, something not to be lost to competition, something worth its weight in gold. Bob Weissman had recruited Bernie Goldstein to National CSS as chairman, and Bernie became their negotiator. A legendary deal doer, Bernie had amassed a personal fortune by acquiring data centers under the United Data Center umbrella, and—having sold United—was between successes. Dennis and Bernie were a real duo: Dennis, professionally nearly humorless, incredibly smart, as determined as a bulldog, and Bernie, wily, an actor's actor, and experienced in deal-doing beyond compare.

Acquired by NCSS

In the end we got a very good deal, a share-for-share swap of Turnkey stock for National CSS stock at $24 per share, the current NASD market valuation for National CSS. The deal was agreed to in December of 1978 and closed in February of 1979. I became president of the Software Products Group, a new role, reporting to Bob, with the goal of rapidly building our product line by internal growth and acquisition. Dennis went to work for Bernie in corporate development. I became a member of the executive committee, something Bob agreed to but promptly forgot.

Acquired by Dun & Bradstreet

In spring 1979, Dun & Bradstreet decided to acquire a time-sharing company. George Feeney, who had come to D&B as a VP in charge of technology, had previously started GE time-sharing, and he believed that time-sharing was the future of computing for corporations. He desperately wanted to acquire the biggest company in the business, Tymshare Inc., a public company several times the size of National CSS. Tymshare's CEO, Tom O'Rourke, rebuffed D&B's advances and fell into the arms of McDonnell Douglas, so George turned his eye to the number 2 player, National CSS. All of this happened immediately after our acquisition, so close in fact that to this day there are people who believe that I, or at least Bob Weissman, knew that a deal was in the works. Bob assures me this wasn't true, and I can attest that on a flight back from somewhere with Bob a few days before our deal closed in February, I asked him point blank if there was anything at all, even something that need not be disclosed for immateriality, that was in the works that I should know of, and he said "No."

I had followed the Tymshare dance in the press so I wasn't unaware that D&B was on the prowl. When I got a call in May on a Sunday evening from Bob I was surprised but jumped quickly to a few speculative conclusions. Apologetically, Bob said, "Lee, I am so sorry. I forgot you completely. We're over in the conference room having an Executive Committee meeting, and I think you should get here if you can."

I got there, and when the elevator doors opened on the executive suite level, Bob was there to greet me. "We have heavy stuff to discuss," he said.

"Let me guess," I replied, "D&B has made us an offer."

"And do you know the price?" Bob asked.

"$48 per share," I said, taking a wild guess that a 100 percent premium would ensure that no other bidder would snatch this one away.

So Bob escorted me into the meeting and announced to the assemblage: "Lee just arrived, and told me that D&B has made an offer for NCSS at $48 per share. Either he is clairvoyant or our cover is blown."

Once everyone got over the shock of my educated but deadly accurate guess, we debated the merits of $48 cash for a $24 stock, quickly concluded that our fiduciary duty to shareholders demanded a Yes, and the deal was quickly consummated.

Turnkey Systems shareholders received instant liquidity for their shares, at double the negotiated sale price. I received many happy phone calls hailing my praises and insisting that I was a Machiavellian genius who had known well in advance of our sale to NCSS that D&B was going to buy the whole thing.

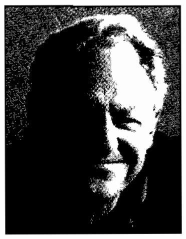
End of Part 1. Part 2 will feature Life at Dun & Bradstreet; the role of ADAPSO in the early software industry; and afterlife.

References and notes

1. Luanne Johnson is president of the Charles Babbage Foundation.
2. Mercator, the product sold today, is a modern object-oriented interpreter of real-time data streams that lets its users link legacy and modern systems with a seamless interface. This product was developed by a company that could trace its lineage directly back to the early Turnkey Systems.
3. We had to account for this as compensation, taxable to the employee, but we "grossed up" the payments so the family travel was at zero cost to the employee.
4. A year and a half later, Jerry's impetuousness got him fired, and Burndy hired Turnkey Systems to provide an interim VP of data processing: me. I did this job for almost six months, with good second-level people who saved my bacon and who were relieved to be away from Jerry's bombast. I hired my replacement—which was my main job at Burndy—who immediately cancelled all contracts with Turnkey Systems and proceeded to make radical changes to the systems architecture that Jerry had built. I'd like to think that by hiring a strong, independent guy we did Burndy a service at our own expense, but the "tear it down" approach seldom fixes problems, and despite later successes I don't think that his approach improved Burndy's systems. As a postscript, Jerry Kaufman left data processing altogether and became a psychologist.
5. Sending the data and its formatting characteristics as separate data elements is now accepted practice. HTML, XML, and other markup languages are based on this concept, but in 1969 it was reasonably innovative.
6. Also see Larry Welke's anecdote in the *IEEE Annals of the History of Computing*, vol. 24, no. 1, Jan.–Mar. 2002, pp. 85-89.
7. Before the System/360, IBM's products ran only batch jobs. Because the processor ran so much faster than the peripheral devices, most of the time the processor was in a wait state. Multitasking, however, enabled the devices' relatively slow processes to overlap. Consequently, with sufficient memory, enough programs could be run to ensure that the wait state was minimized, which meant that this expensive mainframe computer was optimized.
8. The original team consisted of Steve Ward, Dennis Sisco, Joe Farrelly, and me. In 1971, Bill Zack joined the team briefly, but in the main the design and implementation of Task/Master was the work of the original four. As my programming skills became known by the others, I was relegated to developing the user interface modules. In my "break down the door" manner I got the job done, but the programming was ugly.
9. It remains unique. Its source code is on file at the Charles Babbage Institute, and the source code is as self-documenting and readable today as it was 35 years ago.
10. If we consider one-off projects and system software developed under contract to the hardware manufacturers, the industry dates from the mid-1950s. Computer Usage Corporation (CUC), founded in 1955 by Elmer Kubie and John W. Sheldon, wrote a program for California Research Corporation to simulate oil flow. Fletcher Jones and Roy Nutt founded Computer Sciences Corporation in 1956 to develop systems software such as compilers. Simscript, a simulation language developed by California Analysis Centers Inc. (CACI) was launched in 1962, and the ADPAC compiler appeared for direct sale in 1964. Marty Goetz claims that the true software products industry started with Applied Data Research's Autoflow launch in 1965. Historians will have to sort this out, but it appears that there were no companies mass-marketing software packages directly to end users until the early 1960s, and no real

"industry" until a decade later.

11. As noted, Task/Master's source was Cobol, and we distributed this along with Steve Ward's Genesys code-modifying software utility, another unique product Turnkey Systems developed. Genesys produced custom Cobol, unique to the customer's parameters, from a master source listing. This output was then compiled, giving each customer a unique system but from a common master source listing. If they wished, customers could modify their unique Genesys output. So long as these modifications were made in separate linkage modules, they could maintain custom systems across many new releases.
12. Marty Goetz, a former Sperry-Univac programmer, was a founder (in 1959) and later president of Applied Data Research, or ADR, one of the first software products companies. ADR's first product (Autoflow, a computerized flowcharting system) was launched in 1965. Marty later sued IBM for attempting to monopolize the software industry and in 1970 won a $2 million out-of-court settlement. The settlement helped ADR grow and become listed on the New York Stock Exchange. Years later his daughter, now grown and a lawyer, sued IBM on behalf of a medical company Marty had helped found, and won again. I think Marty may be the only person to have successfully sued IBM twice and won both times, using the proceeds to build successful companies. See also M. Goetz, "Memoirs of a Software Pioneer: Part 1," *IEEE Annals of the History of Computing*, vol. 24, no. 1, Jan.–Mar. 2002, pp. 43-56, and M. Goetz, "Memoirs of a Software Pioneer: Part 2," *IEEE Annals of the History of Computing*, vol. 24, no. 4, Oct.–Dec. 2002, pp. 14-31.
13. In 1992, I was riding a train from Westport, Connecticut, to New York City. I was reading my newspaper and heard the two men sitting opposite me say "Task/Master" so I put down my paper, interrupted them, and introduced myself. It turned out that they were from American Brands, one of our earliest users. They were lamenting the fact that their management had decided to abandon Task/Master. This was 14 years after we had discontinued any new development or sold a new customer site. If there is a lesson in this, it is that software has a much longer useful life than almost anybody, including us professionals in the field, will ever believe.
14. We never sold the software, we wrote a perpetual use license. This eliminated the monthly lease or license fee but substituted an optional annual "maintenance" fee, in reality the continuing right to enhancements and bug fixes.
15. The original Graphics product was priced at $15,000, the first Task/Master at $18,000, but over time this price increased steadily. It was close to $40,000 when we discontinued it in 1978.
16. The day's sales outstanding, or the amount of accounts receivable, was expressed as a number of days of average sales.
17. Only Bob, after a few years away, stayed at TSI and worked in the successor company. Leslie went to Rolm as a senior development manager, Joe went first to ADP as vice president of research and development, then to RJR Nabisco and Seagrams as CTO, and Bill became CEO of Gartner Group.
18. An interesting side note is that back in my IBM days I had been asked to evaluate the then brand-new National CSS as a credit risk for its purchase of two IBM System 360/67's, a huge order. I met the founders, heard their business plans, and concluded that they would not last six months, which I reported back to IBM—thus almost making my prophesy come true.
19. See R. Weissman, "CCITT Meeting Recommendations," *IEEE Annals of the History of Computing*, vol. 24, no. 4, Oct.–Dec. 2002, pp. 43-43.
20. We had long before installed a National CSS terminal (a converted Selectric typewriter that IBM called the 2740) to let our people write Cobol programs interactively. The system improved productivity, but our customers would not pay for it, and we could not afford the monthly charges, which frequently ran into thousands of dollars. Also, the system was so slow that Hal Feinberg, one of the VP/CSS developers, added a "tic" that made the Selectric ball spin occasionally, just to let the users think the system was working and not asleep.



Ernest E. "Lee" Keet, cofounder of Turnkey Systems in 1967, is president of a private equity firm, Vanguard Atlantic Ltd. Before founding Vanguard Atlantic in 1985, he was president of the Software Products Group at Dun & Bradstreet. He holds a combined BS and MS degree in mechanical engineering from Cornell University and an MS in operations research from New York University. He currently sits on a number of boards, including the Trudeau Institute and the Charles Babbage Foundation. He resides with his wife in New York, San Francisco, and Paris, with a permanent address in Saranac Lake, New York.

Readers may contact Lee Keet about this article at 62 Moir Rd., Box 1199, Saranac Lake, NY 12983; Lee@VanguardAtlantic.com.