From MCC to CMM:

Technology Transfers Bright and Dim

Bill Curtis
TeraQuest
P.O. Box 200195
Austin, Texas 78720-0490 USA
+1-512-219-9152
curtis@acm.org

ABSTRACT

This paper describes lessons learned during the author's five lives in technology transfer. The author's first life came in General Electric's Space Division where he performed research on software metrics and structured programming, and transferred technology to the pages of technical journals. His second life came at ITT's Programming Technology Center where he was responsible for transferring software measurement practices into common use across ITT's worldwide software operations. Some measurement initiatives survived, but most were short-lived. His third life came in MCC's Human Interface Laboratory and Software Technology Program. MCC's member companies were only occasionally able to transfer the advanced technology they challenged MCC to produce. His fourth life came in directing the Software Process Program at the Software Engineering Institute where he led the team that produced the Capability Maturity Model. Although the CMM's transfer was occasionally too rapid to control, the CMM suggested that you should transfer no technology before its time. The author's fifth life came in co-founding TeraQuest, and working with companies to improve their software development capability. The critical skill in transferring technology often becomes understanding and managing the organization's dynamics. A career model of technology transfer is proposed.

Keywords

Technology transfer, technology adoption, change management, MCC, software measurement, design process, Software Engineering Institute, Capability Maturity Model, CMM, software process improvement.

Life 0—Innocence

They never told me the whole story when they awarded my doctorate. They only told me the cheery side, "You will publish research and increase human knowledge." Years later I would come to understand the dark side, "On judgement day, you will be held accountable for whether anyone ever used your results!"

After my doctoral defense in September 1975, I crammed all my books, clothes, and TV into my Chevy and drove into the sunset (Texans are always supposed to ride into the sunset) heading for the University of Washington and my new Research Assistant Professorship. I had been hired to work in a research project on organizational management. When I arrived, untimely fluctuations in research funding left me teaching statistics and scrambling for a new research topic.

However, any young researcher who could twirl tape and punch cards (like twirling lassoes and punching doggies, these were daily practices of our ancestors) would not suffer long without research colleagues. When a professor in an adjacent office learned I could sling statistics and navigate a computer center, he invited me to join his sports psychology research on coaching behavior.

We spent pleasant spring days studying how little league coaches behaved (or mis-behaved) and what effect this had on their players. We collected bathtubs full of data, I lived in the computer center, and we published articles as fast as we could write them. One afternoon while watching the NCAA basketball finals, we drafted guidelines for coaching kids and later developed a Coach Effectiveness Training program. We began training coaches, became popular speakers at coaching conferences, and had a national organization publish our guidelines. Life was good and research was both rewarding and fun. I came to appreciate that my professors had prepared me well—"You will publish research and increase human knowledge!"

Lesson 1: *If there is strong demand for the results of your first research project, you have learned nothing about technology transfer.*

With no tenure track position in sight, my idyll in academia succumbed to the whimsies of research funding. I was offered a research position in General Electric Space Division, but it was supported by the 'soft research funding' toward which I was now quite skittish. Instead I took a stable position in a stable corporation in a stable industry. Six months into my stability my highly regarded boss was fired. Five months later my department was dissolved and I was appointed to an irrelevant position at a forgotten outpost in the corporate hinterland. As a popular corporate sport, 'downsizing' was still a decade away. Yet I will always be grateful for my opportunity to get in some early practice.

Lesson 2: *Your research position is stable only if you are paid essentially nothing by a financially strong university with a winning football team.*

Life 1—General Electric Space Division

My teetering career was salvaged by an out-of-the-blue call from Tom Love who had offered me the job at General Electric a year before. His soft research funding had outlived my stable position in a stable corporation. So I stopped trying to find 'hinterland' on a map and instead joined Information Systems Programs of General Electric's Space Division in Arlington, Virginia on February 6, 1978. With Tom's promotion two months later, I began managing the research team and foraging for soft research funds.

The group was initially supported by funding from the Office of Naval Research to study whether you could predict anything useful from software metrics. When I arrived the data from the first experiment was supporting the dreaded null hypothesis. Realizing that my new appointment would not last a fortnight if the null hypothesis were left standing, I ransacked the data in search of anything significant. Knowing very little about the substance of the experiment, I naively asked why, if you cocked your head a-kilter, the scatterplot relating metrics to performance looked like waves washing up on a beach. Tom was starting to fear his 'out-of-the-blue call' had landed him a space cadet.

Yet, the next morning Tom bounded into the office rejoicing that if you adjusted the data to connect the ends of my erstwhile waves, the metrics started predicting performance. In fact, he found that what I had perceived as waves were actually sets of datapoints for different programs, each of which had its own characteristic relationship between complexity and performance. Science was served, but more importantly, funding was saved.

In addition to validating that metrics predicted performance, Sylvia Sheppard, Phil Milliman, and the other members of the research group also demonstrated through a series of experiments that structured programming and several other modern programming practices enhanced programmer performance. Our research was well received because our experiments were conducted with real programmers rather than the usual lineup of college sophomores.

We were pretty proud of ourselves. We had demonstrated empirically that metrics could predict performance and that structured programming mattered. Even so, the world was moving at its own pace in adopting structured programming, and our results did little more than offer some refreshment along the way.

Lesson 3: *If your research validates the benefits of technologies people have already decided to adopt, you are merely a gnat on the hindquarters of technology transfer.*

Metrics was a greater conundrum. In a field that claims a kinship to mathematics, why could we find so few numbers on projects? We showed our impressive correlations to all sorts of people, and they were impressed until we left the room and they had to get back to coding. Could it be that computer science was where mathematicians escaped when they got tired of numbers?

We had learned a lot about what precautions had to be taken to use metrics to predict performance. We had proven the predictions worked in experiment after experiment. We had published these results in several of the most prestigious scientific journals in software engineering. Still, when programmers talked about 'alphanumeric' they 'alpha' more than they meant 'numeric'.

Lesson 4: *More people watch professional wrestling than read scientific journals*.

Space Division had created a small group to figure out what to do about software, since software was swelling inside the systems we were delivering. This group created a Software Engineering And Management (SEAM) process that, by policy, all projects were supposed to use. Collecting and archiving measures was an important part of this process. This policy caused managers to adopt a new behavior at the beginning of a project, the behavior of writing a waiver from the policy. It appeared that measurement was perceived as a threat, rather than an aid to project management. I was as popular as a dentist with the ever helpful message of brushing and flossing.

When people talked about technology they usually meant machines and the software that ran them. Technology transfer seemed to imply the ability to get people to use some design method or software tool that was purported to benefit them. Only years later would I see in the dictionary that 'technology' was the *knowledge and means for*

producing something. Providing people with measures or knowledge about how to use tools and methods properly just did not seem to justify the cachet of 'technology transfer'. Measurement and knowledge were commodities you would expect to get from a consultant, not a technologist. Or so it seemed back then.

Since the job of my group was to win and perform research contracts as a profit and loss center, we were only asked whether we would meet our schedule and budget numbers. We were never evaluated on whether our research had any effect on practice, only on whether it was funded.

Lesson 5: *If your performance is not measured on whether someone adopts your technologies, you can accurately predict your transfer rate.*

Life 2—ITT Programming Technology Center

My idyll in General Electric succumbed to an offer from ITT to establish a corporate software measurement program in their Programming Technology Center. ITT began life as an international telecommunications company, but had grown into the world's largest corporate hodge-podge. ITT's hook was a salary offer that ensured the only way I could return to a university would be to coach football. Childhood was over. No longer could I just study software measurement, I had to get folks to adopt it. 'Technology transfer' had morphed from an interesting research topic into a salary continuation plan.

I arrived at ITT's Programming Technology Center in Stratford, Connecticut on August 11, 1980. Before I could learn to spell 'telephony', I was shipped to ITT's European headquarters in Brussels to review the top level design comments on ITT's 'bet-the-farm' project, System 1240. I was to analyze the 3000+ review comments and make a recommendation on what, if any, action should be taken.

Although I learned to spell telephony, after struggling through the jargon of the first two dozen comment forms I knew that a content analysis was laughable. I retreated to simple frequency counts, leaving an education in telephony to a later day. I plotted the approximately 900 documents according to their comments-per-page proneness. Most documents seemed to have attracted a stable, but not alarming number of comments. However, the inflection point in the curve indicated there were about 40 documents needing major surgery. I listed these documents and dutifully made my report.

There was one document that drew such wrath it had three times the comments-per-page of the next most comment-prone document. When asked which one it was, I surveyed my list and announced, "It's the *Human to Machine Communication Design* document." This drew the quick reply, "Don't worry about it, that's just the user interface." This was the first half of a lesson I would fully learn at the end of my next life.

Lesson 6: *Some areas of technology get no respect--let someone else worry about transferring them.*

The development of System 1240 was consuming all the profits ITT made on Hostess Twinkies and *The Joy of Cooking*, but no one could account for why since it was being developed in multiple design centers spread across two continents. In mid-1981 I was ordered again to Brussels to help develop a measurement standard to be applied to all aspects of System 1240 development. We wrote and revised for days, and when we were tired of revising I got on planes and trains and visited the affected design centers to get their concurrence on the definitions.

In one country I was seated across the table from 3 Ph.D. mathematicians who argued the fine details of the definitions with me for 8 consecutive hours. When we reached agreement we shook hands and they reported data monthly using exactly the definitions as agreed. In another country I was received graciously, and following some pleasant socializing, we held a short discussion of the definitions. We reached quick agreement, shook hands, and I returned to Brussels. Apparently their data must have gotten lost in the mail.

Lesson 7: *The extent to which they argue the technical details is directly related to the likelihood of their adoption*.

Later we developed a set of 10 measures to use in tracking projects at various phases of development. We developed a simple tool to aid in collecting and reporting the measures. We provided training and assistance in getting started. We had a mandate from headquarters that the data be reported monthly. Even so, many centers remained stubbornly consistent in their unwillingness to collect, use, and report measures.

We had started some initial collection of project data from ITT units in 1980. We wanted to establish an initial baseline, but the quality of the data we gathered was mixed. Every possible definition of a line of code, or of a person-hour, or of a defect had been used somewhere in ITT. Even so, the CEO had been promised a baseline and a baseline we would create. After some measure-magic we published a report estimating an initial baseline. We pointed out the limitations in the quality and consistency of the data, but were assured this was no concern since executives were delighted to see baselining begin.

The following year the baseline activity was taken much more seriously and the CEO expected to see improvements that justified the treasure he was investing in the Programming Technology Center. All centers were asked to report data using the agreed definitions. When they asked why they should spend time replying to yet another information request from corporate, the reply "Because the CEO asked for it" seemed to work best.

One third of the data created interocular trauma in that it was so unreasonable it struck you between the eyes. All sources of questionable data were called to verify their submissions. In half of these cases—fully one sixth of the submitted data—the results were found to be inaccurate by as much as an order of magnitude.

Lesson 8: *Even when you think their attention is focused on your technology, they may have attention deficit disorder.*

When we got the data sorted out, a sorting far too extensive to discuss here, they indicated that ITT was 24% more productive in 1981 than it had been in 1980. The CEO was delighted and we got to keep our jobs. Caveats in the report about fluctuations in future baseline results caused by changes in the mix of projects being completed in subsequent years was removed over the protest of the authors. Future generations of baseline authors would have to explain to executives why ITT was 10% less productive in 1982.

Lesson 9: *Executives are unable to learn the concepts of random variation, sampling theory, and standard error of measurement while viewing data that support their position.*

In 1982 there arose a distress signal from System 1240 that operators were rejecting the user interface. As the report filtered down to my level I pointed to a forlorn datum in a long forgotten report on System 1240 top level design comments. Our failure to interest anyone in the usability of the interface had left festering a significant threat to product adoption.

Lesson 10: *Murphy's Law of Technology Transfer—The technology you failed to transfer would have prevented the biggest defects.*

Based on the company's dearth of user interface guidance, and buoyed by some work I had done at General Electric, I started a user interface activity in 1983. I consulted with several product groups to make their user interfaces more usable. The technical people quickly grasped the issues, but it was never clear that the management would make the commitment to build usability into the budget or schedule.

By the end of 1983, the Programming Technology Center was beginning to succumb to a swirl of business events overtaking ITT. Since I had been one of the people peering deeply into the data, I knew too well what was coming. After talking with friends I applied and was invited to become the Technical Director of a user interface group at a new research center back home in Texas. Capers Jones who had been an inspiring colleague through the years at ITT left about the same time to start his new endeavors in Boston. Although collecting and analyzing software measures can be frustrating, one of its benefits is that you are among the first to know when it is time to leave.

Lesson 11: *If you cannot transfer your technology, it is best to transfer yourself.*

Life 3—MCC

I joined the founding team at Microelectronics and Computer Technology Corporation (MCC) on December 5, 1983. MCC was born when CEOs of several American computer manufacturers realized that if the Japanese were successful with their government-funded Fifth Generation Computer project, their current generation computer products would become obsolete. MCC represented a new concept, an industrial consortium among competitors to create new technology they could all use in their product lines. The concept required the U.S. Congress to pass a law exempting such pre-competitive ventures from traditional anti-trust laws. It is always thrilling to see the U.S. Congress organize itself for sufficiently long to pass a helpful law.

From the start the new CEO, Bobby Ray Inman, emphasized that the real challenge was to transfer the breakthrough technologies into product line use at the many companies that were supporting our research. This seemed like a great thing to say to keep the member companies satisfied that we cared about them. However, we knew that the real challenges lay in solving the mysteries of symbolic processing that stood between us and breakthroughs.

MCC had review groups stacked on top of review groups to ensure that the member companies had oversight and approval of every component of the research program. Each of the original seven research programs had a Technical Advisory Committee composed of technical experts from each company funding the program. Above them was management oversight committee composed of senior sponsors from each of the companies involved in MCC. Far above them was a Board of Directors.

The sponsors and technical experts were tasked with representing each company's interests in the research program. They were supposed to link units in their own companies into the technologies that would most benefit their product lines. In addition, each company was asked to send a technically qualified person to act as a technical liaison between a MCC research program and the target units in their home companies. These liaisons were to be full members of the research staff, but a portion of their time was to be devoted to transferring the technology back into their company. In addition each company was expected to create an advanced development group for each program it supported to track MCC research and begin converting it for company use. Never had such a labyrinth been laid between a research lab and its beneficiaries.

The first sign that the labyrinth was flooding came as MCC was staffing and launching the research programs with a decision by the members companies to require MCC to pay for the liaisons out of its research budget rather than cover

their costs from budgets at home. In addition, few of the advanced development groups tasked to track MCC technology were ever created. The liaisons were now burdened with the full responsibility of technology transfer while holding up a position that was expected to contribute to the research.

Lesson 12: *Executives who are not willing to pay for technology transfer probably still believe in Santa Claus.*

In the early reviews of our proposed research the member companies consistently exhorted us to accept more risk in our research plans, and to push faster into unexplored approaches. Many agreed that information about promising ideas that lead to a dead end were often as valuable as new technologies since they saved wasted effort in advanced development. In addition they agreed that our primary product was a finished prototype that they could redesign into their own architectures. Since we were to push as fast as possible into prototyping, Lisp machines were the vehicle of choice.

Then AI winter set in. The fundamental challenges of artificial intelligence proved far harder than they had been described in the Business Week cover articles that hailed this new technology in 1983. Reasoning within a domain is had already been conquered, but when you have to reason across domains, the human brain still dominates its electronic impersonators. Artificial intelligence would make a commercial contribution, but not one large enough to justify the investment in MCC. We were creating more permeations than breakthroughs.

When the Japanese Fifth Generation project proved less of a threat than originally perceived, MCC's support further dwindled. The liaisons generally did not know Lisp. Without advanced development projects working to translate MCC prototypes into a company's proprietary architectures, the full burden fell on the research staff at MCC. The new CEO who replaced Inman soon blamed the problems on Lisp as a poor language for supporting MCC's mission.

Lesson 13: *Regardless of their cause, all problems in transferring software technology can be blamed on the programming language.*

By 1987 it was politically correct to condemn Lisp and forswear machines devoted to it. Times were so strange that the concept of a breakthrough research prototype that did not stray to far from conventional technology stopped sounding ridiculous. The research pace began crawling as prototypes were painfully built in languages and environments meant for production. MCC proposed a new technology transfer organization that would translate MCC prototypes into standard production ready components for delivery to the member companies. This proposal received as much funding as the liaisons had a few years earlier.

The star in MCC's orb was its lone venture into hardware technology, an effort to accelerate the pace of packaging technology for semiconductors. This program set more modest and predictable goals than the other research programs, because the technology was easier to quantify. They targeted and generally delivered technologies with a six-month lead on the market, large by semiconductor standards.

Lesson 14: *It is easier to transfer deliverables that people can hold in their hand.*

In 1985 I transferred to help Les Belady build the Software Technology Program. Belady rapidly focused his program on creating technology to aid requirements and design processes. He gave his vision a name, Leonardo. The member companies loved anything that had a name because it provided the illusion of results. The first step in technology transfer is to settle on a name, since frequent reference to it buys time to determine if the name stands for something other than the emperor's new clothes.

Lesson 15: *If there are no technologies to transfer, transfer a proper noun to buy time.*

Leonardo would emerge from four research projects that were chartered to build the underlying design engine, the design information base, the interface for designers, and tools for managing the design process. I was offered the design interface project, but I had grown weary of interfaces, and the design process sounded like a new frontier. I chose to focus a research team on methods and tools to aid the process of design.

For the first year we heard the same refrain from the member companies at each quarterly review, "You do not understand our problems!" Growing tired of receiving hackneyed explanations in response to questions about these infamous problems, we decided to venture into their domains and study their problems. Herb Krasner, Neil Iscoe, and I interviewed 19 design teams on large projects and found that the big problems were the thin spread of application domain knowledge, requirements volatility, and communication and coordination breakdowns. These were not exactly the problems everyone expected we would find, but once we started talking about them, we never again heard that we did not understand their problems. In fact it now appeared we understood their problems better than they did. Interviews with member company executives several years later found that this report had been read more than any other.

Lesson 16: *If you can not transfer a solution, maybe you can transfer a diagnosis.*

The prevailing model of software design in the 1980s was of a top-down, breadth-first process. No-one believed it, but to believe otherwise would desecrate the sacred design methods that had been handed down across generations of

software gurus (of which unfortunately we were now in the first). To determine which design religion we would claim as our ancient heritage, Herb and others spent hours videotaping designers and teams designing software. Just as we had feared, you could always prove top-down, breadth-first processes with textbook problems. Add the kind of complexity they never dare to breach in textbooks, and the prevailing model tumbles top down.

As we peered into the videos, we saw an all-out scramble for whatever lifelines designers could find. Rather than top-down analysis, they appeared to be opportunistically searching for recognizable patterns in either the problem or solution space upon which they could anchor their design. Yet, being able to recognize patterns and relationships in the shards of information about the problem and potential solutions put the responsibility on the designer, not the tools. The raw material was knowledge, and all our popular technologies could do was capture and organize it. The research results became far more popular than their implications. Our results became a form of Bible that were cited as support for all manner of conflicting opinions.

The Software Technology Program transferred prototypes of many tools and technologies. However, we were never able to transfer a full design environment. The cost of producing the infrastructure needed to transfer an environment exceeded the research budget, and was never accepted by the program or its members as the primary value of the program. Worse, few of the members worked seriously enough with the program to integrate our results into their standard corporate environments. Consequently, researchers scrambled to make personal links with developers at the member companies who believed in the potential of our technologies. The more successful we were in establishing these individual links, the less likely we would ever integrate and transfer Leonardo.

One of the most promising technologies was called gIBIS for 'graphical Issue-Based Information System'. The objective of Jeff Conklin and Michael Begeman in producing this system was to aid designers in capturing the argumentation and rationale produced by the design process. This system was at the forefront of the movement that emerged in the 1980s to focus on design rationale. Jeff found a dedicated user in one of the member companies and supported her intensely as a demonstration project for using gIBIS for capturing the design information her team created. She ultimately attributed considerable cost savings to the mistakes this design discipline allowed her to avoid.

Unfortunately we were unable to find other disciples with the same fervor for gIBIS. Design rationale ended up being far more interesting to researchers trying to understand design than to practitioners who had mastered it. I found it far easier to interest people in adopting gIBIS if they visualized it as a way to capture the information produced in technical reviews.

Lesson 17: *The Principle of Process Sloth—It is far easier to transfer technologies that support processes people already perform, than to transfer anything that forces them to adopt a new process.*

The organizations that were the most successful in transferring MCC technology had single individuals who assumed a personal mission to link MCC researchers together with advanced development groups that could use their technology. Unfortunately few companies assigned an individual into the breach between MCC and its members who was capable of initiating and sustaining these relationships.

At the end of 1989 I was asked to return to the Human Interface Laboratory to try and save the program's funding. The Lab had succeeded in developing an integration of knowledge-based and media-based interface technologies called HITS, the Human Interface Tool Set. Although the members found the HITS architecture innovative, they were unhappy that it was hosted on the now dreaded AI machines. 'Lisp' was harder to spell than 'C'.

I began visiting companies near and far to sustain the existing members and try to add new ones. For their part, the researchers ported the core components of HITS to C++. Now we found that even porting to C++ was not enough. Too many product groups wanted the prototypes ported to the specific architecture of their product line. For too many of the members 'advanced development' only meant development in advance of the delivery date.

Our last hope was to find executives whose products needed improved user interfaces to compete more successfully in their markets. I found Vice Presidents in charge of every conceivable component of a system except the user interface. There were VPs in charge of operating systems, applications, databases, and development tools, but nary a one claimed responsibility for usability. Inspired by an old Spanish mission 75 miles to the South, the Lab hung on far longer than expected. Yet we learned that usability was as much a bastard child in corporations as it was in computer science departments. In late 1990 the Human Interface Laboratory was closed, completing my lesson begun in ITT on user interface's niche at the bottom of the computational pecking order.

Lesson 18: *Corporations will not adopt a technology in which they will not invest the lowly sum of a Vice President.*

Having seen the energy we were putting into design process research between 1986 and 1989, Watts Humphrey had asked me to join the advisory board of the Software Process Program at the Software Engineering Institute (SEI) in Pittsburgh. Three times a year I flew to Pittsburgh and joined several others in spending two and a half days in constructively debating some new ideas Watts had for

improving software organizations.

Although the board meetings were spirited, Watts came to believe I believed, and he believed this more than I. Consequently, in 1989 he began asking if I would eventually be interested in taking his position. As MCC's fortunes waned I became more receptive, but was still not sure I wanted to leap back into the technology transfer fray with something as nebulous as a 'maturity framework'.

My last trip to attract a new member into MCC's research programs took me to a large financial institution's software technology meeting in Omaha, Nebraska. One of the hosts had been a former Air Force officer and arranged for a dinner in the Officers' Club at Offut Air Force Base, the home of the Strategic Air Command. After dinner a Colonel Dean Briscoe spent an hour describing to us how his group managed the software that replanned the destruction of the entire world every 3 days. He admitted that it was far easier to destroy civilization than to maintain software. Then he went on to describe how this Watts Humphrey fellow had come over from Pittsburgh and helped them gain control of their software operations. He had been skeptical of Humphrey's methods, but they had worked. For the first time I realized that hypotheticals in Pittsburgh were beginning to bear fruit out in the country.

I had now turned 40 and the measure of my career rested on a collection of technical papers that in combination did not generate as much attention as Wrestlemania VII. Every attempt to transfer a technology reminded me of a line from Shelley's Ozymandias, "Nothing beside remains." Hopefully Watts would call again.

Life 4—Software Engineering Institute

On February 1, 1991 I replaced Watts as the Director of the SEI's Software Process Program. For the first few weeks I busied myself with whatever new Directors are supposed to busy themselves with. Yet the title 'Director' implies something more than 'busy', so I looked for a sign.

The sign came in burning emails from a system acquisition executive in the Air Force who owned the SEI's contract and from a group of experienced project managers Watts had assembled to provide feedback on the maturity framework. Both wanted Watts' book, '*Managing the Software Process*' transformed into a set of process improvement guidelines, and they wanted it now.

The maturity framework grew from ideas Watts and his colleagues had developed in IBM during the 1980s. However, these ideas blossomed when the U.S. Department of Defense (DoD) asked for a way to evaluate the capability of software contractors. DoD was tired of having to accept the lowest bidder knowing that the bid too often reflected total ignorance of the effort required to build the system. In response to DoD's request, Watts worked with a small group to complete the maturity framework. This new model was presented in an *IEEE Software* article and a questionnaire.

The maturity questionnaire was not statistically rigorous, but for better or worse, it caught the imagination. The model as it stood at the turn of the decade was best expressed in '*Managing the Software Process*'. However, detractors found the book much harder to attack than the inviting target presented by the questionnaire. Many found great sport in criticizing the questionnaire, and a couple were granted their 15 minutes of fame.

When DoD started using the maturity framework in selecting contractors, an unfortunate checklist mentality overwhelmed the questionnaire. When DoD started requiring that bidders achieve maturity level 3 in order to receive contracts, the dearth of level 3 bidders appeared to be a clever solution for reducing the national debt.

From March through August of 1991, Mark Paulk, Charlie Weber, and many other members of the Process Program struggled valiantly to produce the Capability Maturity Model for Software (CMM) from the foundation laid in Watts' maturity framework. To formulate some implementation guidelines, the Process Program had collected bathtubs of software practices in workshops. I had participated in one such workshop in the late 1980s and had been asked by Watts to list every measurement practice I had actually seen work (fortunately he forgot to require that they still use it). The task for the spring and summer was to order these myriad practices into the maturity framework, produce a model from them, shepherd it through two national reviews, and publish it for use. It seemed straightforward when we planned the project in late March, a plan whose half-life was 7 days.

We delivered version 1 of the CMM at the SEI's annual symposium at the end of August 1991. Upon announcing this delivery, the attendance doubled from the previous year, and then kept growing until the fire marshal refused to allow more than 900. When DoD started using the CMM in awarding business, Watts had found the mechanism for changing at least one segment of industry.

Lesson 19: *Although it is good to have an innovative technology with demand-side pull—it is better to have a hammer.*

As aerospace companies began struggling with the improvements recommended in the CMM, we began to get telephone calls. The usual scenario was, "Hi, we've just completed an assessment of our software practices, now what?" "Fix your biggest problem" was not their preferred response. Although we assumed most organizations could quickly address the findings of the assessment, in truth many had no idea how to conduct an improvement program. Unfortunately, our insights at the SEI were little deeper than theirs. Transferring the CMM was not enough, we had to figure out how to help folks ingest it.

We were now under extreme pressure to upgrade the process assessment and capability evaluation methods, concoct a roadmap for improvement programs, develop a process definition method and process asset library, solve the ancient mystery of measurement definitions, and train every computationally literate soul in process improvement. Although canceling one's private life until further notice is politically incorrect, the experience of finally having demand for your technology is energizing.

The SEI wanted to maintain balance across its various programs. It was not designed to handle the national (and ultimately international) demand of a successful product line. The stresses were occasionally intense as we struggled to meet growing demands on an institute that was organized to advance the ideas and transfer the transition responsibility to others. The idyllic life of an academic institute had collided with frenetic pace of the software industry.

Lesson 20: *Having your technology adopted nowhere is a tragedy, having it adopted everywhere is a nightmare.*

Even with broad adoption in the aerospace industry there were many who remained unconvinced that the CMM held any benefit beyond large government systems. We were constantly harangued by companies to present data validating the benefits of improvement programs that we could only report if they collected it. Finally validating data started appearing from software organizations that occasionally considered them as much a marketing bonanza as a scientific obligation.

We received endless calls from people who were crafting briefings to convince their executives of the benefits from software process improvement. As we counseled them it became apparent that too many executives were using the request for benefit analyses as a tactic for stalling an improvement program. How come so many executives who never supported software measurement suddenly wanted all these data? My only retort was that a low maturity organization had so little data that it could not even defend the Return-On-Investment for having Vice Presidents.

Lesson 21: *Executives' passion for Return-On-Investment analyses of a technology's benefits is directly inverse to their interest in adopting it.*

The CMM harbored a model for technology transfer that we did not fully understand as we developed it. In 1991 we did not comprehend that Watts' genius lay in formulating a new theory of organizational development, and one that contradicted much of the conventional wisdom. Within this theory was a new way to effect technology transfer.

Traditional organizational development called for problem-solving teams to analyze problems and propose solutions that could be adopted across the organization. It sounded good in theory, but it was a sound that could not be heard above the chaotic din of most software organizations. Watts had understood that postulating organization-wide solutions was the wrong way to start because people were too busy trying to meet unachievable schedules to worry about adopting anything. First you had to control the chaos. This was not the job of problem-solving teams, it was the job of project management.

Rather than focusing first on the technical work that the rest of us had focused on for two decades, Watts focused on management. Long before we worried about organization-wide improvements, Watts wanted improvements in the management of projects that would control project commitments and baselines. He did not care that this control might be established in different ways on different projects. He just wanted project management to accept responsibility for stabilizing the project so that capable technical folks could do their work in a disciplined and orderly way. He never assumed that software developers had genetic pre-dispositions toward anarchy or disorder. Rather he assumed they were proud professionals who wanted a chance to employ the professional practices to produce quality products, even under pressure.

The initial focus of the CMM is on project management, and the first technologies the organization may benefit from are those that assist managers in planning, tracking, and controlling baselines. We had spent two decades transferring technologies to solve problems we could only address effectively after we had eliminated the chaos of over-committed schedules and volatile requirements. Our problem was not that developers could not solve technical problems, it was that we threw them into a mad rush and they started making mistakes they did not have time to detect and correct until the project was in crisis. In the 1980s most of us had focused on design, but Watts had focused on project management.

The evidence had always been there. Every time we tried to transfer a technology managers and developers were too busy fighting fires to take notice. They did not have time to take the training, read the manuals, or update to current versions of the tools. In the 1970s and 1980s we had been too busy trying to solve the cool technical problems we were trained to solve rather than solve the problem that was right in front of our eyes. We blamed technology transfer stalemates on the intransigence of developers rather than on our failure to solve the problems standing between them and our purported breakthroughs.

Watts had his own envisaged solution—Shewart and Deming's Plan-Do-Check-Act approach to continuous improvement based on statistical control of process and quality performance. Yet he realized that most software organizations suffered too much daily crisis to adopt a solution so alien to their experience. It was in this

realization that the maturity framework was conceived. The staging of the CMM was predicated on Watts' understanding of the ordering of problems he had to help an organization solve if they were to ultimately achieve the promised benefits of continuous improvement.

Lesson 22: *The real benefit of your technology may lay in the barriers you helped someone overcome in adopting it.*

As organizations mature, they start adopting technologies we had spent years trying to transfer. Once they have a reasonably consistent process constructed from what they believe to be their best practices, they find it easier to select the methods, tools, and environments that provide the best support. The common process makes it easier to match technologies with actual development needs. Before this point they had too often been trying to match technologies to imagined needs. This is called marketing.

Improvement data emerging from companies like Raytheon, Motorola, Hughes, Boeing, Ericsson, Tata, Telcordia, and others, showed encouraging productivity gains from eliminating rework as they achieving levels 2 and 3. When developers participated in making achievable commitments they avoided the crisis atmosphere, made fewer mistakes, and caught them early when they took less time to fix. This benefit we anticipated.

However, there was surprising leap in productivity as organizations passed beyond level 3 that we never anticipated. This second productivity gain resulted from an explosion in reuse. Reuse had been the holy grail of software productivity since I entered the field in 1978. It was the pot of gold just around the corner. It was still just around the corner in 1988. Finally in 1998 some companies finally turned the corner.

The reason some made the turn would not have been surprising if we had been analyzing problems rather than transferring proper nouns and visions. It was not that we had not known how to craft reusable components. Rather it was that we were in such a rush that we did not have time to produce something we would want to reuse. We were in crisis mode and knew that our components had not been properly designed or tested. We did not have time to properly document them or to update whatever documentation we scribbled out. Frankly there were components so rushed through development that we did not even want to admit we developed them.

However, as organizations passed beyond level 3 they finally had a development process they trusted. They knew from their measures how much time it took to design and build a component, and they knew its quality at delivery. If they needed additional time to design a more generic component, they understood how and where to alter their process. Reuse happened, and sometimes naturally, when developers shared common, trusted processes and a commitment to professional discipline.

Lesson 23: *If you cannot transfer your favorite technology, transfer a framework that helps folks figure things out on their own.*

Thanks to the real estate crash in Texas at the turn of the decade, I was unable to move my family to Pittsburgh. For two years I endured a 1200 mile commute and was sustained by the four basic food groups of American Airlines—peanuts, pretzels, chocolate-chip cookies, and Dr. Pepper. Having regularly transferred myself for two years, it was time to come home.

Life 5—TeraQuest

On February 1, 1993 I opened operations as an independent consultant and advised several companies on their improvement programs. Although being an independent consultant is a heady experience if the phone rings, I quickly learned that I would never have the leverage needed to help an organization conduct large scale improvements. So when Don Oxley and Joyce Statz called to talk about a company they were in the process of forming I saw an opportunity to help create something with the leverage I needed. TeraQuest (no, I do not know what it means either) was born in 1993 to work with organizations on improving their software operations.

Although it is easy to say that consultants get paid regardless of whether the organization gets better or not, in truth consultants are not retained if they do not produce results. I no longer had to wait for judgement day to be held accountable for whether someone used my results, an accounting came monthly when we had to meet a payroll.

One of our earliest lessons was that Watts really had gotten it right in the maturity framework. In some of our earliest engagements we tried to roll out the classically designed improvement program with all the ingredients suggested in the organizational development and technology transfer literatures. We trained the Software Engineering Process Group, organized a Management Steering Committee, and helped coach Process Action Teams that had been assigned to solve specific problems. This was the improvement mantra, and even the SEI was formulating an improvement model called IDEAL that institutionalized this approach. It was a fine model and nothing appeared wrong with it.

Until nothing happened. In several of our earliest engagements the action teams and process groups would take the results of an assessment and disappear for months designing solutions to the most significant problems. They were trained to do this, they were engineers. All the improvement models said—one, pick an important problem—two, design a solution—three, pilot the solution—four, revise the solution—five, transfer the solution across the organization. The only problem was that after 6 months we were barely into step three. It

sounds okay until we realized that the rest of the organization was burning.

Lesson 24: *Executives will give you six months to make something better—period.*

At the six month mark executives appear to have been genetically programmed to get very nervous about overhead expenditures. It they have not seen any evidence that something is improving, regardless of how small, they will assume the program has failed and withdraw its funding. The most successful improvement programs all seemed to have a common characteristic, they began working with projects to make improvements very early. By the dreaded six-month mark, several projects had plans and were able to provide a reasonably accurate accounting of their progress. We could not show productivity and quality results yet, but when executives see the beginning of competent project management they will give you credit for making a beneficial change. Once you have been given this credit, you will have time to deploy organization-wide solutions. But first things first (projects primarily).

When your livelihood and your reputation depend on your results, you learn at a much quicker pace. We came to realize that Charles Darwin knew more about organizational development than most social scientists, and both knew more than computer scientists. The change management pundits keep calling these improvement programs 'the quality journey'. Rubbish, this is not a journey, it is war. The more we are able to make executives understand that their ability to transfer technology is key to their survival, the better results we achieve.

The toughest lesson has been that the organizations that most need your help are the ones least capable of sustaining an effective relationship. Immature organizations are characterized by fire-fighting and off-again-on-again decisions. Every significant step forward is subject to an unanticipated delay. The executive sponsor is the key, and to understand their orientation I like to ask, "What is your job?" If they respond with. "My job is to get products out for my customers", I know we are in trouble. I try to find a diplomatic way to say, "NO! Your job is to build an organization that gets products out for your customers, your managers' jobs are to get the products out." When they cease thinking like managers and start thinking like executives, they see improvement and technology transfer as their responsibility, rather than that of the improvement group.

Lesson 25: *Transferring technology is like riding a wild bronco—you figure you can do it, but when he starts bucking you are in for some serious ups and downs.*

A Career Model of Technology Transfer

Every responsible academic paper must propose a model, and I would regret to fail this expectation. The sum of my 25 lessons leads less to a model than to a confessional. However, the historical pattern of my technical supplications suggests an unexpected model. If I can not formulate the transcendent model of technology transfer, then perhaps I can at least model what it felt like along the way. Accordingly I propose the following 5 stage model of careers in technology transfer (forgive me, it seems everything I am involved in these days has 5 levels). The model represents the maturing (there I go again) approaches technology transferists (or is it transferians?) adopt in transferring improved methods and tools across their careers.

The Technology Transfer Career Model

STAGE	*APPROACH*
1–Youth	I will present it to them and they will adopt it because it is good
2–Adolescence	I will work with them to adopt it because I can prove it is good
3–Young Adult	I will understand their needs and help them adopt an appropriate solution
4–Adult	I will find a hammer and they will adopt my solution
5–Elder	Having once used a hammer, they will listen to my wisdom and adopt it

Scientifically useful models require predictions that can be tested. According, this model predicts that you will either succeed in a career of transfer, or in a transfer of career.

From personal experience I can only attest to the existence of the first four stages. However, I have determined through observation that the fifth stage exists. The lower three stages are measured in ohms, the upper two stages are measured in Watts.

And so we must ultimately come to the preparation for final judgement. The bright side of increasing knowledge inspired a career. But the fear of the dark side of accountability drove it through many fitful redirections, none ever leaving a full measure of satisfaction, but a few holding promise and pointing the way onward. If technology constitutes know-how, is not then the ultimate technology transfer the transition of oneself from innocent education into harsh practice, and from the cauldron of experience into wisdom—all in preparation for the final accounting.