



Three Process Perspectives: Organizations, Teams, and People

WATTS S. HUMPHREY

watts@sei.cmu.edu

*The Software Engineering Institute, Carnegie Mellon University, 4500 Fifth Ave., Pittsburgh, PA 15213,
USA*

Abstract. This paper provides the author's personal views and perspectives on software process improvement. Starting with his first work on technology assessment in IBM over 20 years ago, Watts Humphrey describes the process improvement work he has been directly involved in. This includes the development of the early process assessment methods, the original design of the CMM, and the introduction of the Personal Software Process (PSP)SM and Team Software Process (TSP)^{SM*}. In addition to describing the original motivation for this work, the author also reviews many of the problems he and his associates encountered and why they solved them the way they did. He also comments on the outstanding issues and likely directions for future work. Finally, this work has built on the experiences and contributions of many people. Mr. Humphrey only describes work that he was personally involved in and he names many of the key contributors. However, so many people have been involved in this work that a full list of the important participants would be impractical.

Keywords: assessment, education, improvement, management, personal, process, productivity, programming, PSP, quality, software, team, training, TSP

The first software process improvement work I am aware of was at IBM over 20 years ago. In around 1980, Al Pietrasanta was made IBM's Director of Programming Quality and Process. His job was to improve the productivity and quality of IBM's programming. Even earlier, Mike Fagan had introduced many of the concepts of software process improvement with his first pioneering paper on programming inspections [Fagan 1976]. During this same period, Harlan Mills in IBM's Federal Systems Division was describing improved software design, testing, and quality management methods [Mills 1987]. These three, Mike Fagan, Al Pietrasanta, and Harlan Mills, provided the intellectual foundation for much of what the process improvement community has later accomplished.

In 1983, Al Pietrasanta was made director of IBM's Systems Research Institute and I assumed his quality and process job. At that time, the volume of IBM's programming work was growing rapidly and software maintenance and repair then cost the company \$250,000,000 a year. Management believed that software would be increasingly important to the company and that we needed a special focus on software quality. The

* Personal Software Process and PSP are service marks of Carnegie Mellon University. Team Software Process and TSP are service marks of Carnegie Mellon University.



Figure 1.

company had previously established a company-wide quality policy and the hardware engineers had made dramatic improvements. However, the software groups had made very little progress.

1. “We can program SDI”

Based on what we learned over the next several years, I wrote a brief editorial, or Speak-out, for *IEEE Spectrum*. It was entitled “Yes We Can Program SDI” (see figure 1) [Humphrey 1986]. SDI was the Strategic Defense Initiative, and it was popularly called

When initially applied, these disciplines produce rapid improvement. Once schedules and costs are under control, a plateau is reached and major changes are needed for further progress. While the team is now ready for a full-scale attack on quality, this first requires a formal definition of the programming development process. This is done by dividing it into several steps, assigning entry and exit criteria to each, and carefully tracking progress. With such comprehensive data, the cost and frequency of each type of error can be found and resources can be deployed to maximize quality and minimize the impact on schedule and cost.

Next, error causes are assessed and preventive methods established. This may require improved tools or modified standards, for example, or better development coordination. Every step in the development process must be carefully managed, for poor design cannot be repaired through tests. The objective is to improve the effectiveness of each step. The limit on the quality of a large program is the most erroneous step in the process that produced it. When exhaustive testing is not possible, defect-free programs can be produced only when the output of every development step is itself defect-free.

There are many supporting elements for high-quality software development. People must be trained, an architecture is needed, and adequate tools and support are essential. But there is no magic tool by which programmers can intuitively produce the quality required. The SDI programs must be developed by strong technical teams that use a highly disciplined development process.

There is growing evidence that this approach works. When process management methods were applied to semiconductors, quality, as measured by process yield, surprised even the technologists.

In programming, the results to date have not been as dramatic nor as visible but they are encouraging. For some leading manufacturers, the quality as measured by defects per thousand lines of code has improved by three to five times in the last 10 years and, at the current rate, will improve a full order of magnitude in the next decade.

While the current trends are headed in the right direction, even the anticipated high quality level of the best commercial programs will be insufficient for the SDI. What is needed is dedicated teams with an unswerving focus on quality and a disciplined and measured development process. Estimates are that 10 million or more lines of error-free code are needed. If the estimates are correct, the challenge is enormous. But if even a modest degree of error tolerance can be built into the SDI, the job can be done.

The time and resources required to program the SDI will likely be substantial. While estimates of time depend on the anticipated size for the programs and the degree of error tolerance permitted, three to five years is not enough. New development teams will likely be needed, and it will take several years just to assemble them and build the required development disciplines. With a national commitment, however, the job could probably be done in 10 to 15 years.

If the SDI is found necessary, it should not be delayed by concerns about programming quality. The methods and technology are known and can be applied today. The benefits of this venture would be enormous. While it would be senseless to build the SDI solely for its technological side effects, the resulting discipline would lift programming from its "stone age" and launch the next industrial revolution.

– Watts S. Humphrey – 1986

Figure 1. (Continued.)

Star Wars. Its objective was to protect the US from a nuclear missile attack by the Soviet Union.

The feasibility of the SDI system was hotly debated, and the principal concern was writing the required software. The contention was that since the total system could not be exhaustively tested it would be impossible to produce software of the required quality. I agreed that, with the software practices commonly used at the time, we could

not develop large complex programs of the quality required. However, there were more effective ways to develop software.

While I did not debate the desirability or feasibility of the SDI system, I believed that the decision to develop this system should not be based on these software concerns. In my Speakout, I explained that the quality required could be achieved, but only if the software developers consistently used the very best quality methods. However, I also said that, to develop the required software capability, a major improvement initiative was needed and that it would probably take at least 10 years.

The recommended improvement initiative was not launched, but we have learned a great deal in the intervening 15 years. We now know that the quality principles described in my article of 15 years ago do work. We also know how to build disciplined and productive software teams and we know that such teams can build large and complex software systems that are essentially defect free. We also know that disciplined software teams can do quality work on predictable schedules and within planned costs. However, we have also learned that software process improvement requires a concerted effort and that this effort must address three distinct levels:

- The organization.
- The team.
- The individual.

In this paper, I review this improvement journey, what we have done, what we have learned, what worked, and what did not work. At the end, I discuss the issues we currently face and the challenges ahead.

2. Software process improvement at IBM

A few years before I retired from IBM, I was the Director of Technology Assessment. In this job, I reported to Art Anderson, the senior vice president for large systems manufacturing and development. This group developed and produced almost all of IBM's data processing products. My role was to assess the technology used in these products. The objective was to ensure that IBM had a premier position in all the significant technologies for its systems. While programming was important, Art's major concern was semiconductor development and manufacturing.

2.1. Early assessments

Art Anderson had previously used organizational assessments at IBM Research and he suggested that we use them to review the semiconductor manufacturing and engineering groups. In an assessment, the assessment team evaluates the organization's performance against defined criteria. In assessing IBM's technologies, we considered patent coverage, patent activity, technology roadmaps, technical roadblocks, professional accomplishments, and competitive capability.

The first group we assessed was IBM's semiconductor facility in Burlington, Vermont. We found that IBM could then import semiconductor chips from Japan at lower cost than manufacturing them in Burlington. We said that if IBM Burlington could not be cost competitive, we would close the facility and purchase imported chips. This got management's attention. They assessed their entire operation and, in their final report to Art, they described significant cost improvements. They had used process measurements to identify quality problems and to more than double the number of "good" chips produced from a single silicon wafer. This cut manufacturing costs by over 50%. Since IBM Burlington then produced most of IBM's memory chips, this was very significant. In just two years, IBM became the industry leader in chip design and manufacture.

2.2. *Yield management*

This experience provided several powerful lessons:

- First, product cost was a direct function of manufacturing quality.
- Second, quality was measured by product yield, or the percent of produced chips that were good.
- Finally, product yield, and thus quality, could only be improved by improving the process that produced that product.

With chips, it was obviously impossible to improve quality by testing; you had to fix the process. While extensive testing was essential, you could not test and repair chips at the end of the line. Unless quality was built in during development and production, there was no way to consistently improve yield and reduce costs.

The critical need was for precise process measurements to use in improving the chip manufacturing process. To manage yield, the engineers had to examine every defect, identify its cause, and then change the process to eliminate the major defect causes. This required detailed and precise data so the engineers could identify the most important problems and establish improvement priorities. It also required a defined chip-production process and a comprehensive measurement and analysis program.

Finally, the assessment motivated the entire effort. In the Burlington assessment, the local engineers analyzed the problems and reviewed the results with management. Soon the entire organization understood the problems and why they had to be fixed. The assessment provided the best way I had yet seen for understanding an organization's capability and getting broad agreement on the need to improve. It also gave management useful guidance on what and how to improve.

2.3. *Software process assessments*

When I moved to the software quality and process job, I wanted to continue using assessments. Ron Radice and Jack Harding were in my new group and I asked them to devise an assessment method for software [Radice 1985]. The first software laboratory we approached was San Jose. While the laboratory director was interested in having us do an assessment, his immediate managers were not.

Ron, Jack, and I met with the entire San Jose management team. It took a full day to convince them that the assessment would help them. What changed their minds was our promise to only report the assessment results to them. We agreed not to publish a score card or report anything to corporate headquarters. We had stated this position at the very beginning but it took all day before they believed us. These department managers did not want any more auditors. They simply could not believe that anyone from headquarters would not take their findings to senior management or tell corporate how bad a job they were doing. We agreed to only report the assessment results to the laboratory director. He could use that report in any way that he chose.

The first assessment was very successful. The San Jose managers told the other laboratories and soon they too were asking for assessments. IBM corporate management respected our confidentiality commitment, but there was a lot of curiosity about our findings. Since management really wanted better software work and not laboratory grades, we convinced them that scores were irrelevant.

2.4. Crosby's Quality College

The assessment method we used was based on a 5-level maturity model used in Phil Crosby's Quality College [Crosby 1979]. Crosby calls them stages and he describes them as follows.

- *Uncertainty*: Management is confused and has no knowledge of quality. Managers regularly speak about quality but have no understanding or commitment to it.
- *Awakening*: Management begins to recognize that a quality program could help but they are unwilling to devote time or money to quality improvement.
- *Enlightenment*: Management decides to launch an improvement effort and makes a commitment and investment in the quality improvement program.
- *Wisdom*: At this stage, quality is recognized as important and a quality executive is named. Quality is now measured and managed.
- *Certainty*: At this stage, the quality problems are largely solved and quality has become a vital corporate activity.

Ron and Jack followed the general concept of Crosby's levels but renamed them as traditional, awareness, knowledge, skill and wisdom, and integrated management system. While these levels were useful, they concerned subjective judgments of people's attitudes about quality. Therefore, our assessment evaluations were largely subjective. However, since we had nothing better, we continued to use Crosby's maturity structure.

2.5. Assessment problems

The initial laboratory assessments were very successful. Management could see where improvement was needed and they could make changes to address the problems we identified. However, when we made a second assessment of a laboratory, we had problems. The source of the trouble was that the subjective maturity levels had no relationship to

specific software activities. At the Kingston laboratory, for example, the first assessment rated the laboratory strong in some areas and weak in others. The managers then implemented an improvement plan to address the identified weaknesses.

In the second assessment a year later, our evaluation found that the laboratory was rated worse in several areas where they had made substantial improvements. In the assessments, we interviewed the engineers from four to six typical projects. If these projects did not appear adequately focused on quality, for example, that is what we reported. However, we did not report which engineers or projects had which problems. While we could explain that we had studied different projects in the two assessments, the fundamental problem was that we could not relate the assessment findings to specific software activities. With the Crosby maturity framework, we could not connect our ratings to needed improvement actions.

2.6. The management commitment problem

While these maturity levels were not perfect, they were useful and did result in significant improvement. The biggest problem we faced was maintaining management's commitment. To launch an assessment, we first met with the appropriate laboratory director and his¹ entire management team. We first had to convince them that we would not report the assessment results to anyone else before we could persuade them to do the assessment. Unfortunately, IBM's laboratory directors changed jobs about every 18 months so it was nearly impossible to visit every laboratory often enough to keep everybody committed. When I retired from IBM in 1986, my two biggest problems were the subjective assessment framework and continuing management commitment.

3. An outrageous commitment

As I approached IBM's normal retirement age, I started thinking about what to do next. I knew that the software quality problem was critically important, not just to companies but to our entire society. The software job is to precisely define human problems so they can be solved by machines. Since human problems are inherently unlimited, the software business is inherently unlimited as well. As human ingenuity devises solutions to its most pressing problems, we can expect newer and more challenging problems that will need even more advanced solutions.

My conclusion was that society's rate of progress was at least partly limited by software technology. If we did not master this technology, industrial growth would be stunted and human progress constrained. This seemed like a challenge worth devoting my life to. This is what I called an outrageous commitment. While I had no illusions that I could change the world of software, I decided it would be an exciting challenge and it would certainly be worth the effort. In tackling this challenge, I went to see Eric Bloch in early 1986. Eric was an old friend who was then the head of the National Science

¹ There were no women lab directors in IBM at the time.

Foundation (NSF). He suggested that I join the Software Engineering Institute (SEI) at Carnegie Mellon University, and he even wrote me a letter of introduction.

3.1. The Mitre study

Shortly after starting at the SEI, Larry Druffel, the director, asked me to work with a team at the Mitre corporation on a way to evaluate software vendors. The Air Force had studied 17 major system contracts and found that every project was late and over budget. The average delay was 75%. This meant that one 4-year contract took 7 years. In every case, the source of the problem was software. What was most discouraging from an Air Force perspective was that all 17 contracts were late and over budget. These contractors were the Air Force's premier suppliers. Therefore, merely selecting from the available vendors would not produce the desired results. The Air Force had to motivate better work. This was exactly what I was looking for so I gladly participated in the study.

Bill Sweet and I were the two SEI team members and the Mitre people were Martin Owens, Rodney Edwards, Gerry LaCroix, and Herman Schultz. Bob Kent of the Air Force had originally suggested the study and Dick Sylvester of Mitre was our principal sponsor and advisor. In the study, the first step was to convince the team that we should evaluate the software process the bidders were using, not just their proposals. Everybody agreed this was a good idea but they felt that the standard proposal evaluation process should continue. We ended up adding a capability evaluation step to an otherwise standard Air Force proposal evaluation process.

The principal assumption of this work was that the process an organization used in the future would be much like what it had done in the past. Therefore, the Air Force could reasonably expect that the organization's future software development performance would be much the same as in the past. If the organization planned to change its process, it should describe the changes in the proposal. What was most important to me was that this motivated management to think about process quality every time they submitted a proposal. Since this work was in a different environment and had different objectives, I did not initially see how it related to my earlier IBM assessments. It was several months before we made that connection.

3.2. The maturity model

In deciding how to evaluate software organizations, we first listed the characteristics of capable software groups. This produced a set of about 100 questions. Since we expected these evaluations to be performed by Air Force personnel and not by software experts, the questions dealt with specific activities that the assessors could readily understand and verify. Once we had the questions we faced another problem: how could people who were not software experts use the questionnaire to evaluate and rank organizations? For example, if the evaluators got answers to the 100 questions from each of a dozen vendors, they would have 1,200 answers. How could they possibly use these data to rank the vendors in the order of their software capabilities?

One day, while we were struggling with this question, I was stranded in Atlanta Airport. It was early October in 1986 and I had lots of time on my hands, so I decided to work on this problem. It occurred to me to list these 100 questions against Crosby's maturity framework. To my surprise, they fit very well. While I had to redefine the maturity levels, it was not hard to rank the questions in maturity order.

From my experience managing large and small software groups, I knew that the first step in managing a software operation was to develop plans and to only make commitments when you had plans and the staff to do the work. A comprehensive requirements effort was essential, as was independent quality assurance and configuration management. These were the basic requirements for the first maturity step, or level 2. Since I had found that organizations with these level 2 capabilities repeatedly met their commitments, I called level 2 repeatable. Once an organization achieved level 2, it would be tracking and controlling its work with plans rather than reacting to crises.

The next maturity step concerned identifying and adopting best practices across the organization. To be a learning organization, the best practices must be defined and broadly used. I therefore called level 3 defined. The few remaining questions concerned measurement, statistical control, technology adoption, and defect prevention. I broke these questions into a level 4 that addressed measurement and level 5 for continuing process improvement. I called level 4 the managed level and level 5 the optimized level. I had initially picked chaotic as the name for level 1 but the Mitre people convinced me that most groups would be rated at level 1, at least at first, and that they would probably object to being called chaotic. We agreed to call level 1 the initial level. We also realized that level 5 would never be fully optimized since there was much to learn about software. We therefore called level 5 the optimizing level [Humphrey 1988].

4. The capability maturity model

We refined this maturity model and soon published it in an SEI technical report [Humphrey 1987]. Since the US Air Force had announced that it would evaluate potential contractors against this maturity framework, this technical report was used by many military contractors to evaluate their organizations. We were also asked to assess many software organizations and to suggest how they could improve their software capabilities. Soon, winning bidders realized that their improved software capability was helping them to win competitive contracts.

Many organizations have now used the Capability Maturity Model (CMM)² and experience has shown that it addresses the concerns I earlier had with getting management attention, maintaining a long-term improvement focus, and guiding the improvement work. Because it has provided these benefits, CMM use has grown steadily. Results of an early SEI study of CMM assessment results for 13 organizations are shown in table 1 [Herbsleb 1994].

² Registered in the US Patent and Trademark Office.

Table 1
Summary of CMM results.

| Category | Range | Median |
|---|--------------------|-----------|
| Total yearly cost of SPI activities | \$49,000–1,202,000 | \$245,000 |
| Years engaged in SPI | 1–9 | 3.5 |
| Cost of SPI per software engineer | \$490–2,004 | \$1,375 |
| Productivity gain per year | 9%–67% | 35% |
| Early defect detection gain per year (defects discovered before test) | 6%–25% | 22% |
| Yearly reduction in time to market | 15%–23% | 19% |
| Yearly reduction in post-release defect reports | 10%–94% | 39% |
| Business value of investment in SPI (value returned on each dollar invested) | 4.0–8.8 | 5.0 |

4.1. Getting management attention

The CMM has been very effective in getting management's attention. Initially this was because of the support we got from the Department of Defense. Col. Jack Ferguson led the very first contractor evaluation, and his group was instrumental in getting the CMM used by progressively more groups in the Air Force and then in the other military services.

The US Air Force announced that it would use the CMM to make contractor risk assessments and consider these assessments in source selection. The major contractors then realized that software process improvement was important. With few exceptions, they asked the SEI to help them improve their software capabilities. This was what we wanted and it worked exceedingly well.

4.2. Maintaining the improvement focus

The CMM also helped to maintain management's focus on process improvement. This was both because the DoD used the CMM to evaluate contractors and because companies found that process improvement was good for their businesses. Contracts were now more likely to finish on time and within planned costs. The workers were also more satisfied and business performance improved [Humphrey 1991]. Soon, other branches of government and industry started using the CMM to evaluate their own software work as well as that of their suppliers.

While the management motivation problem appeared solved, we worried about two problems which had yet to surface: the misuse of the maturity framework and early success. The misuse problem could arise when organizations focused on the maturity levels instead of on improvement. This primarily concerned the government's use of the CMM with software capability evaluations (SCE). Here, government auditors evaluate an organization's capability as part of making a vendor selection. Since organizations would want to get high marks from these auditors, the SCE could easily motivate management to get a high grade rather than to improve the organization's capability. We felt that this

problem would be exacerbated if the DoD set some criteria like level 3 as a requirement for a contract.

The problem of early success relates to improvement motivation. If the process improvement work had been started because of severe software cost and schedule problems, once these problems were solved, the motivation for improvement would be gone. As long as the Air Force or other contracting agency did a competent job of evaluating organizations, the risk of this problem was minimized. While we initially worried about these problems, they did not surface until later.

4.3. Guiding the improvement work

The CMM addressed another of our previous problems: some way to guide the improvement work. The Crosby maturity model did not directly relate to software work, so its guidance was subjective. By defining the maturity levels in software specific terms, the CMM provided a defined set of priorities. For example, an organization at level 1 had the immediate challenge of getting to level 2. The CMM then helped management to identify where improvement was needed. Similarly, in getting to levels 3, 4, or 5, management could see what had to be done and establish plans, provide resources, and track the work. The CMM thus addressed the key improvement problems we had faced at IBM.

The reason this guidance was important was because process improvement has prerequisites. For example, until commitments are properly managed, unrealistic schedule pressures will prevent engineers from following any defined process. Similarly, without a defined process, a measurement program is largely useless because, without a specific process, there is no way to precisely define the measures. Therefore, without the guidance of a maturity framework, groups were likely to waste time on changes that did not actually improve the organization. This would destroy the value of the improvement work and ultimately destroy the entire process improvement effort.

4.4. Building the CMM

Many people were involved in refining and formalizing the CMM. I first wrote a book describing the logic for the maturity framework [Humphrey 1989]. In that book I included an Appendix that described the maturity structure in general terms. To make the CMM truly useful, this framework had to be better defined and structured. At about that time, Bill Curtis joined the SEI as Director of the Process Program and I was made an SEI Fellow. Mark Paulk took over the CMM work and led the team that produced a more precise process definition. While a great many people were involved, the principal contributors were Mark Paulk, Bill Curtis, Charlie Weber and Mary Beth Chrissis. The final CMM structure is shown in conceptual form in figure 2.

As the CMM work progressed, the SEI held an industry/government workshop where Mark's team explained how it had defined the new CMM version. Since the CMM had been initially embraced by both industry and government with little contention, we were surprised at the uproar. The industry attendees were very upset, not so much with

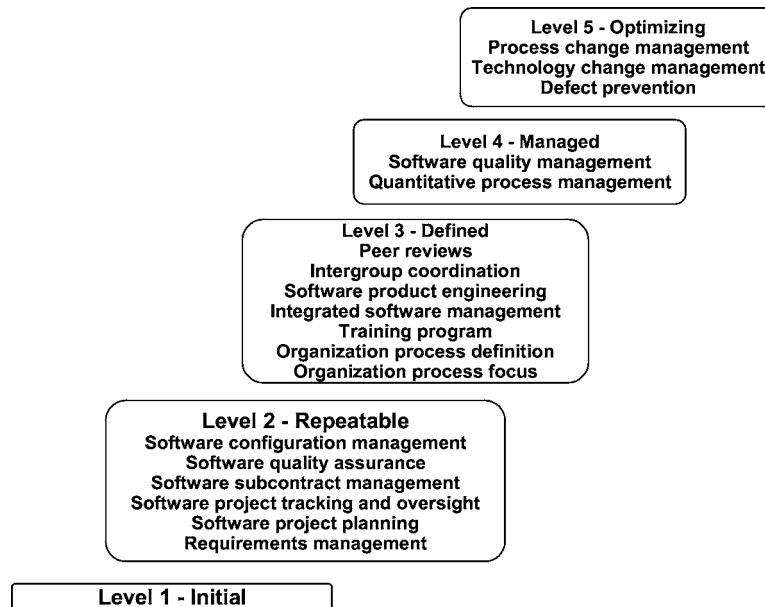


Figure 2. The capability maturity model.

the specific changes as with the way we had made them. While we had received many suggestions, we had worked pretty much by ourselves with little industry interaction.

This was a valid complaint and we realized that the CMM had to be broadly accepted by its industrial users. It would be used to evaluate their work so they should be involved in its definition. Mark and his team then assembled a CMM advisory board and formed committees with SEI, industry, and government representatives to review, revise and complete the CMM definition. More than anything else we did, this inclusive approach was responsible for the quality of the resulting CMM model and for its rapid acceptance [Paulk 1995].

4.5. Other related efforts

One of the great disappointments of this process improvement experience has been the failure of the community to break out of its NIH (not invented here) mentality. Every time software people have faced a new problem, instead of building on prior work, we have invented some new language, tool, or method. This forces us to start over, and it also forces our users to start over. We would have made no progress in physics if everyone invented their own calculus or laws of motion. Chemistry would be nowhere if we had to work with dozens of different periodic tables or chemical symbols. Can you imagine what electronics would be like if no one had agreed on Ohm's law, units of resistance, or standardized component values? When faced with a problem, software people generally find their own solutions, even when the problem has been solved many

times before. That fact that it is so hard to build on other people's work is the single most important reason why software has made so little progress in the last 50 years.

The first example of the NIH problem for the CMM was a European group that came to the SEI for a visit in about 1990. They were launching an effort to build a European version of the CMM. I told them this would be a mistake. I explained that they would have to relearn everything that we had learned and that they would both confuse and fragment the process improvement effort. I said that we would gladly incorporate them into our steering committees and that we would work with them to enhance the CMM to better address our mutual concerns.

They explained that to get European Commission funding, they could not build on US work. They needed a purely European solution. The parallel SPICE (software process improvement) effort was then launched to do precisely the same job as the CMM. This fragmented the process improvement work in Europe and has badly slowed their progress.

Unfortunately, the same thing happened again with the integrated CMM (CMMI). The SEI administered this work, but the principal decisions were made by a DoD directed committee. The CMMI was launched to address precisely the same problems the CMM addressed, only for a broader range of technologies. Even where the CMM and CMMI address identical issues, which is almost everywhere, this group made them different. As a result, the CMMI is not fully compatible with the CMM and any CMM users that want to move to CMMI must learn a new model.

As Harry Truman once said, "The only thing new in the world is the history you haven't learned" [Miller 1973]. The history in this case concerns compatibility. History teaches that improvements are great, as long as they are true improvements, and as long as the ultimate customers will see them as worth the required conversion and retraining effort. While every individual improvement may seem like a good idea in isolation, the risk of making the CMMI too different from the CMM is that the existing CMM users will see it as incompatible. Then many will find it too hard to move and will not change.

4.6. CMM problems

Soon after we defined the initial CMM framework, the Air Force drafted a ruling that all software contractors must be at level 3. This was a problem we had previously worried about and we strongly opposed the rule. The reason assessments have been so successful is because organizations do them to learn about their own capabilities and to guide improvement. While the assessments were often done to prepare for a government SCE evaluation, the motivation was still improvement. If the government required a fixed maturity level, this motivation was likely to change.

No simple model could precisely measure process maturity and complex models are not useful in guiding improvement. Simplicity is essential so the engineers, managers, executives, and acquisition people can all understand the framework, agree on where the organization stands, and understand the needed improvements. Therefore, maturity ratings had to be simplistic summaries of an overall evaluation. An organiza-

tion could not be considered at level 2, for example, unless it met all the level 2 criteria. However, there is a continuous range of capabilities between levels 1, 2, 3, 4, and 5 and there are corresponding degrees of risk at each point on this continuum.

While level 3 might be a useful management goal, it should not be used as a precise guideline for contract awards. There must be room for judgment about the degree to which any criterion is satisfied. With millions of dollars at stake, a hard and fast guideline puts too much stress on the evaluation system. Then an imprecise rating system could not survive. To handle contract disputes, you would have to distinguish between levels 2.76 and 3.04. This would not only be impossible with the CMM, it would be counter productive. People should not view the maturity level ratings as the objective and concentrate on passing an evaluation. Since it is generally easier to pass a superficial evaluation than to actually change organizational behavior, the predictable result would be special process groups whose objective was to pass a level 3 assessment, not to improve the process.

In preparing for these reviews, these process groups would produce all the required paperwork and prepare answers to all the likely questions. They would then make sure that all the projects were prepared to answer the expected questions and that all the paperwork and work products were in place to satisfy a cursory evaluation. Then, with a little practice, an organization could pass any but the most detailed level 3 evaluation without actually changing the way the engineers and managers did their work.

While we convinced the Air Force not to issue its initial level 3 guideline, the DoD has since issued such a ruling. The risk is that this will destroy the usefulness of the CMM as a process improvement guide. If it does, CMM level ratings will no longer accomplish their original purpose: to distinguish between capable and poorly performing organizations. Unfortunately, since the DoD has incorporated the CMM into its acquisition guidelines, we are now on this path and, unless current trends are reversed, the CMM will ultimately not be useful as a guide to contractor performance or to process improvement.

4.7. *What not how*

A second problem with the CMM concerns the original objective I stated in my *IEEE Spectrum* editorial [Humphrey 1986]. This was to build disciplined and motivated software teams that could consistently and predictably produce large and complex programs that were essentially defect free. The CMM was a major stride in this direction since it provided the management environment needed for quality work. However, the CMM consciously focused on *what* organization should do, not on *how* they should do it. From the very beginning, we knew that the CMM would be used to evaluate Air Force contractors. We therefore decided that the CMM should specify *what* software contractors should do and not *how* they should do it. We did not want any government evaluation system telling industry how to develop software. The problem this caused was that the teamwork practices and personal disciplines required for quality software work are almost entirely issues of *how*, and not just *what*.

By ensuring that the CMM did not specify how the work was to be done, we consciously stayed away from engineering practices. As a result, the CMM does not deal with engineering disciplines and it does not address the teamwork practices and personal disciplines required for quality work. Because engineers will not change the way they work without very specific guidance, the CMM does not consistently or predictably change engineering behavior. For that, it was soon clear that we needed something else.

5. The personal software process

As the CMM became more successful, I grew more concerned about the lack of engineering guidance on how to do quality work. I was still focused on my outrageous commitment to change the world of software, but I had concluded that the CMM alone was not going to get us there. Engineers liked working in the more ordered and better managed environment of a mature organization because they more consistently met commitments and had fewer late night crash efforts. However, without more specific guidance, it took organizations a long time to move from one CMM level to the next. Further, even when they did, the engineers' personal practices did not change appreciably between levels 1 and 5.

I have met with many engineering groups and talked with lots of engineers at all maturity levels. When I ask about their personal work, I hear essentially the same answers at all maturity levels. While the management system is different and the engineers are much happier in high maturity organizations, they still do their personal work much as they always have. Personal practices are largely independent of maturity level and I have seen sound personal practices in low maturity organizations and poor personal practices in high maturity organizations.

When we launched the CMM work, I had hoped that levels 4 and 5 would motivate a disciplined, measured, and statistically managed process. While the CMM could not specify how this would be done, I expected that engineers in high maturity organizations would understand the principles involved and change the way they did their work. After all, if the engineers continued to work precisely as they always did, no statistical studies or evaluations could substantially improve the productivity, quality, or predictability of their work. The CMM definitions of levels 4 and 5 focused on the organizational procedures for gathering data and on the management practices for using these data. While sound management practices are essential, they alone do not address the engineers' use of the data. In essence, the need was not for lots of process data but for engineers who gathered and used that data.

I attempted to explain this distinction to the CMM working groups but could not describe what I meant. When asked how these practices would affect the engineers or what specific questions to add to the CMM framework, I could not be specific. That meant that I could not change the definitions of CMM levels 4 and 5 to achieve my longer term objective. While this left me dissatisfied with the CMM definitions, it was entirely my own problem. I did not understand what I wanted well enough to explain it to anybody else. I believed that if engineers personally understood and followed all the

principles of CMM level 5, their performance would improve and they would produce very high quality products. However, I could not prove it or even explain precisely what I meant.

5.1. Developing the PSP

Fortunately, this was when Larry Druffel, the SEI Director at the time, nominated me to be an SEI Fellow. He told me that I could work on any subject I choose. I decided to work on what I soon called the Personal Software Process (PSP)SM. My objective was to see how engineers would work if they applied all the principles of CMM level 5 to their personal work. This meant that they would personally plan each project, track their progress, use a defined process, measure their work, and measure and manage the quality of their products. By working this way, I believed that engineers would consistently produce quality products on predictable schedules. I also believed that their and their team's productivity would improve and that they would find the work truly rewarding. Further, I realized that to get people to work in this way, we would have to show them precisely how to do it.

To understand how the PSP would work, I spent the next 3 ½ years writing programs in Pascal, Object Pascal, and C++. I followed all the practices that I believed a disciplined engineer working at CMM level 5 should use. In the end, I found that my intuition had been correct and that the PSP did indeed help me to consistently produce essentially defect free programs on predictable schedules and with higher productivity than before. Now that I knew the PSP worked for me, my next challenge was to prove that it would work for others.

5.2. The PSP Course

To get people to use the PSP, I talked to many groups. Everybody was polite but I had no takers. Finally, an engineer in a research laboratory in new Jersey contacted me and said his organization would like to try the PSP. I visited this laboratory, talked with the director, and met with the 5-engineer team they had selected. They agreed to try the PSP. The laboratory director even told the team that using the PSP was more important than meeting their project's schedule. For the next few months, I checked with this team every week to see how they were coming. Every time I got the same story: "Right now we're too busy but we'll try it as soon as we have the time." They never had the time.

By now, it was February 1993 and I was at a process improvement conference in Berlin, Germany. I chatted about this problem with Nazim Madhavji, a professor at McGill University in Montreal, and he suggested that I teach a PSP course. That was obviously the answer. Nazim and I agreed to jointly teach the course the next January at McGill. This meant that I needed to design a PSP course and write a textbook. When I got home, I started the writing and course development work. While this sounds easy, it was not. The PSP text turned out to have 800 pages and I had to write it without knowing if anybody would ever use it [Humphrey 1995].

While I was writing the PSP text, Howie Dow and another student in the Masters of Software Engineering program (MSE) at Carnegie Mellon University (CMU) chose to develop a small PSP data gathering tool for their masters project. As he developed the tool, Howie learned about the PSP and became convinced it would work. After getting his MS degree and returning to Digital Equipment Corporation (DEC), Howie asked if he could teach a PSP course at the University of Massachusetts in Lowell in September. This was good news, but the timing was a challenge. I had used the PSP to plan and track my work while I wrote the PSP textbook and, according to my plan, I would not finish until January of the next year. This was April and the PSP data on my first month's work showed that I was ahead of schedule and would finish in September. Of course this assumed that I continued to work at the same rate. Howie agreed that he could take the book in parts if necessary, so with some misgivings, I agreed to get the manuscript to him by the end of August.

5.3. *The AIS company*

It was now late Spring of 1993 and I got a call from Girish Seshagiri, the CEO of Advanced Information Services (AIS). This is a small software firm in Peoria, IL. Girish asked me to visit his company. He explained that he was so excited about the CMM that he had bought a copy of my book *Managing the Software Process*, for every one of his people [Humphrey 1989]. While I complimented him on his taste in textbooks, I said that I was now working on the PSP. When I explained what that was, Girish said he would like to try it at his company. That July, I visited AIS and Pat Ferguson, one of the project managers, agreed to use the PSP on her project. I gave her a copy of my personal PSP process and agreed to come back in November to see how they were doing. When I returned, I was disappointed to find that they had not started to use the PSP. Pat had decided to get all the CMM level 2 practices in place first. By this time, I had finished the PSP manuscript and suggested that Pat and Girish teach their people the PSP using my draft textbook. They agreed and arranged to have a professor at Purdue University teach the course.

5.4. *ERAU*

At the SEI Symposium in September 1993, professors Soheil Khajenoori and Iraj Hirmanpour contacted me. Iraj was the head of the Computer Science Department at Embry Riddle Aeronautical University (ERAU). They explained that this university had the largest aeronautical engineering program in the world and that it worked closely with NASA, Boeing, Lockheed, and others on many research and student projects. It was located in Daytona Beach, Florida, right near Cape Canaveral.

Soheil and Iraj were starting an ERAU graduate program in software engineering and wanted to base it on the CMM. They asked for my help. While I agreed, I suggested that they include a PSP course and that it be the first required course in the MS program. They not only agreed but they wanted to teach this first PSP course the following January.

While I could not teach the PSP course with Nazim Madhavji at McGill, he taught it in January and I taught it at CMU. I now had five customers for my book: AIS, McGill, ERAU, U. Mass, and CMU. This was a marvelous testing ground for the course and a source of data on the PSP process. I finished the book on time and the planned PSP courses were all taught. When the professors sent me the class data, I found that the results were better than I had hoped.

My first class was a blessing in a way that I had not expected. Three people from the SEI were taking the course: Dan Roy, Julia Mullaney (then Julia Gale), and Jim Over. Subsequently, they all decided to work with me in transitioning the PSP into general practice. They have been doing so every since. Neil Reizer, an MSE student who worked for us part time, also took the course. Afterwards, Neil suggested that the SEI offer the PSP as an industrial course. That seemed like a good idea so I asked him to get the course launched. When he said that he wouldn't know where to start, I suggested that he figure that out. He next complained that he didn't have the authority. I told him to tell everybody what he was doing so no one would be surprised, and then just do it. Nobody would stop him. Nobody did. The SEI has been teaching the PSP course ever since.

5.5. PSP course results

As we taught more industrial PSP courses, we accumulated a substantial amount of data on course results [Hayes 1997]. Figures 3–5 show the average results for 298 mostly working engineers who took the PSP course. The point at the left in each figure is the result for program 1 where the engineers used their prior software practices. By program

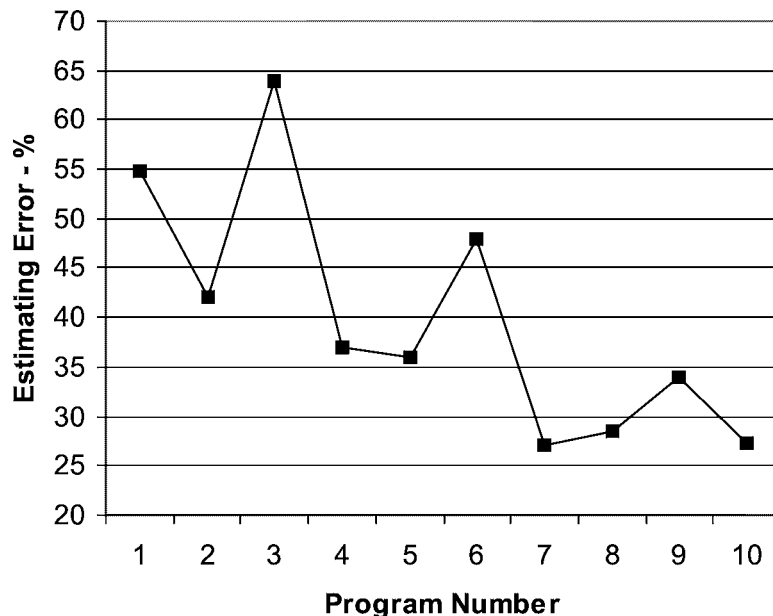


Figure 3. Time estimating error.

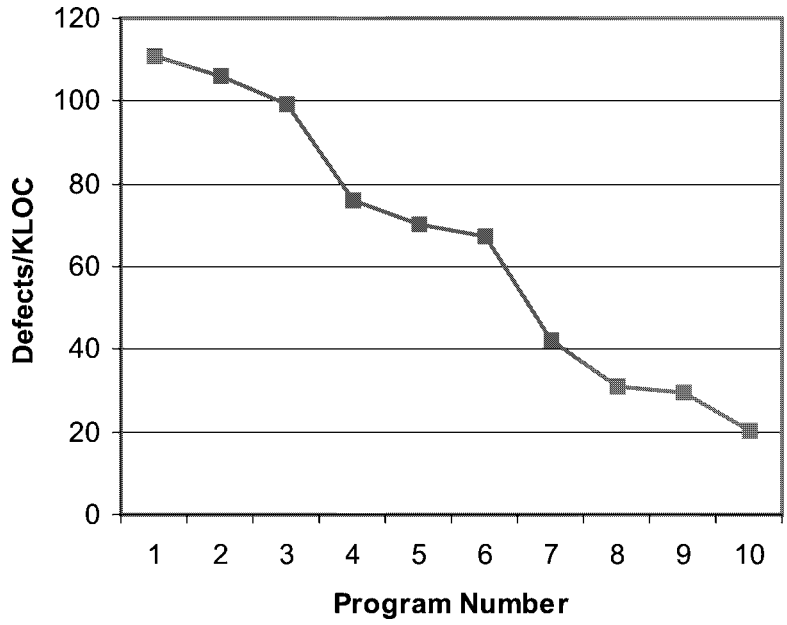


Figure 4. Defects removed in compile and test.

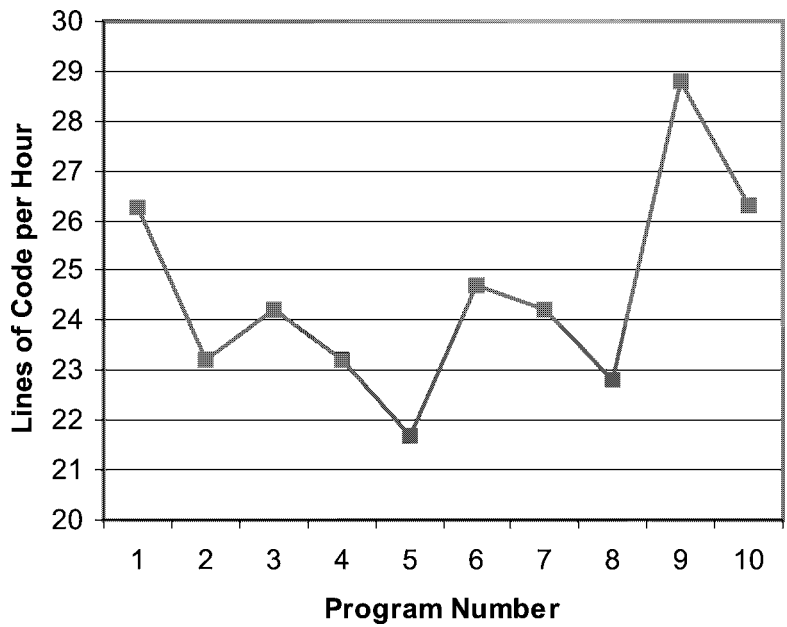


Figure 5. Productivity.

10 at the right, they were using the full set of PSP practices. As shown in these figures, without significantly changing productivity, estimating accuracy improved two times and compile and test defects were reduced by over five times. As we later found, early defect reductions of this magnitude reduce project test times from months to weeks.

While I had expected planning and quality improvements, I had never dreamed that they would be this dramatic. These data showed that by using the PSP, engineers could do better work without any performance sacrifice. Of course, they had to take the PSP course and consistently use the PSP methods.

From the course data, I could see that the PSP did accomplish what I had thought it would. I now had PSP data from a mix of students and experienced engineers and I could see that the PSP methods were effective for just about any software professional who completed the full course. I also found that the PSP was such a significant change in working practice that engineers had to complete the entire PSP course and write the 10 exercise programs to truly appreciate its benefits. What was most surprising to me was that the PSP substantially improved the performance of just about all software engineers, whether they were new and inexperienced or whether they had done software work for 20 or more years.

The PSP was not just a couple of new techniques, it required that engineers completely change the way they developed programs. This degree of change required that they truly understand and believe in these new methods. To achieve this level of understanding, engineers had to use the PSP to write a substantial number of programs, gather data on their work, and see from their personal data that the PSP helped them to do a better job. Since they had personally gathered these data, they could not argue that the results did not apply to them.

5.6. The consequences of the PSP

With few exceptions, when engineers complete the entire PSP course, they can see the benefits and are persuaded to apply the PSP to their personal work. In my first PSP course at CMU, I was fortunate to have a marvelous group of students. I had their data when I revised the PSP manuscript for publication and all the book examples used the course data. I learned a great deal from this class and revised the manuscript based on their suggestions. Howie Dow also made many useful suggestions that substantially improved the course and textbook.

The PSP addressed two of the problems I had encountered with the CMM. First, it showed engineers that process improvement can help them personally. It also showed them how to implement most of the CMM practices. This in turn made it easier for PSP-trained groups to get to level 3. Then government guidelines requiring CMM level 3 would be more likely to facilitate real and not superficial process improvement.

The second PSP benefit was that it directly addressed the need for engineers to use disciplined methods and to measure, plan, and statistically manage their work. The PSP training showed engineers how to define processes; how to use a defined process; how to plan, measure, and track their work; and how to measure and manage quality.

This would not only accelerate process improvement, it would help both large and small organizations. In fact, PSP practices would even help software engineers when they worked by themselves

5.7. *PSP problems*

After Pat and Girish had trained several AIS engineers in the spring of 1994, they started using the PSP. These engineers produced some truly exciting results [Ferguson 1997]. In fact, AIS has continued to use the PSP. While this was a marvelous outcome, it was somewhat misleading. Pat Ferguson and Girish Seshagiri are marvelous managers and leaders. They provided the guidance and support needed to get the PSP used consistently. It wasn't long before I realized that there are few managers or executives who naturally provide the kind of coaching, leadership, and support that the PSP requires.

In helping organizations introduce the PSP, we found that few engineers would use it in their work. It wasn't that they didn't believe the PSP would work or that it was particularly hard to use, but that the working environment discouraged them from using it. The PSP requires disciplined personal behavior and, as in many other fields, highly disciplined behavior requires a supportive working environment. Musicians have conductors, actors have directors, and athletes have coaches. In medicine and the sciences, there are rigorous laboratory courses and internship programs that demonstrate and instill the necessary skills and disciplines. There are also regulations, professional standards, and oversight bodies to ensure that the practices are consistently used.

In short, disciplined behavior is not normal or natural for most people. It requires extensive training, a supportive environment, and a change in management style. Because of Girish and Pat's capable leadership, the engineers at AIS have continued to use the PSP in their work. This proved that the methods could work in industry [Ferguson 1997]. However, it was also clear that a broader transition program was required to get the PSP more widely used.

6. **The team software process**

While the PSP was an important step, it was only one step in a longer journey. There were still three important issues to be identified and addressed:

1. Getting engineers trained in the PSP.
2. Convincing management to invest the time required for PSP training.
3. Getting PSP trained engineers to use the PSP methods on the job.

These were not really PSP issues, because the PSP clearly worked. To address these issues, we needed to take an entirely new approach. This new approach was the Team Software Process (TSP)SM.

As a first step, we made PSP training a prerequisite for the TSP. Next, we set the TSP objective as providing the environment needed for teams to consistently follow the

PSP disciplines. It took us a little longer to realize that the TSP had to do much more than provide an environment to support disciplined work. After all, the team is the most powerful tool mankind has devised for doing large-scale intellectual work. The TSP had to be a vehicle to help organizations capitalize on the potential benefits of disciplined teamwork. To do this, we needed motivated teams that made their own plans, managed their own work, and were committed to producing quality products. We also needed managers who knew how to guide, support, and motivate such teamwork.

6.1. Developing the TSP

In developing a process to show engineering teams how to consistently follow the required disciplines, we had to work with teams that did industrial software work. While we knew what the teams should do, we needed to understand how to get them to do it. I had been able to develop the PSP by myself, but this was not possible with the TSP. It was soon obvious, that, to develop improved teamworking methods, we had to work with teams.

I also faced another problem. I had now been working with the PSP and CMM for over ten years and these practices now seemed obvious and natural to me. I needed to work with practicing engineers and managers to understand their problems and issues. So I looked for organizations to work with. Fortunately, I had continued to work closely with ERAU on their academic program and they agreed to use the TSP in a one-semester graduate course. While these would be students and not practicing engineers, it seemed like a low risk way to start. The ERAU students would have previously taken the PSP course and the TSP project would show them how to apply the PSP to an industrial-grade software project.

6.2. The first TSP project

By now it was July 1996 and I was committed to providing a process and a draft users' manual to the ERAU faculty for the TSP project laboratory course in September. I had also been contacted by an industrial group in upstate New York that wanted to try the TSP. They had already started introducing the PSP and saw the need for the TSP. I then developed the TSP0 process and wrote a supporting manual. However, I had a trip planned to Australia and New Zealand and could not help either of these first teams start their projects. I hoped that the PSP training, the TSP process, and the users' manual would be sufficient.

When I returned from Australia, I visited both teams. At ERAU, five graduate students had started the project but two had to leave part way through. Eric Dauth, Steve Lehr, and Beth Levine finished the project and did a marvelous job. Beth was also working for a company as a software engineer in industry while getting her MS degree, but Eric and Steve had no previous industrial experience.

When I first met with these students in October, they were closely following the process and making good progress. However, shortly after developing their initial plan in September, they had started to worry about the tight schedule and the slow pace of

the requirements work. They were developing a flight scheduling system for the ERAU flight line and had to work with flight operations to understand their needs. Progress was very slow.

Within a week, this team got so concerned about the schedule that they abandoned the TSP process. They then worked without a process or a plan for a couple of weeks. They told me later that the project was totally out of control and they soon realized they were lost. They then returned to the TSP process and followed it religiously for the rest of the job. They had made this decision a couple of weeks before my visit and they were now closely following their defined process. While they were slightly behind schedule, they finished only a few weeks late. They produced 4,540 lines of code (LOC) at a rate of 18.08 LOC per hour. This included all the planning, data gathering, requirements, quality, testing, and postmortem activities. They removed 79.9% of the defects before unit test and 99.43% of the defects before system test. System test defect density was 0.22 defects/KLOC. This was substantially better than any industrial team I had yet seen.

While these students were a few weeks late, the late delivery was not a surprise. They had used earned value to track their work and had accurately projected their completion date. I concluded from this team that the TSP process would work if it was properly used. I didn't realize until later how effective the ERAU faculty had been in helping this team achieve its impressive results. Professors Iraj Hirmanpour and Soheil Khajenoori regularly met with the students, reviewed their work, and provided encouragement, guidance, and support. What was most important, they told the students that, while they had to finish the project, their grades depended more on the quality of their process than on their completion date.

6.3. The first TSP project failure

The experiences of the industrial team were not as encouraging. They had started reasonably well but had not continued to follow the process. Their manager had not insisted that the engineers gather their time, size, or defect data and they did not. While she had agreed that data gathering was important, she did not have the training or experience to get the engineers to properly follow the process. Shortly after my first visit, the company had a major reorganization. The subsequent layoff included many engineers, including some in the same department. Many engineers then got nervous and quit. The project manager was the last person left on the project and she called me just before leaving to explain what had happened.

While management's actions would have destroyed this first TSP project anyway, it was clear to me that, for engineers to use the TSP process, their managers must require that they do so. While the manager need not look at every engineer's data every week, he or she must know that the process is being followed. Even though this first industrial project failed, my experience at ERAU convinced me that the TSP could work. The next challenge was to get it consistently followed.

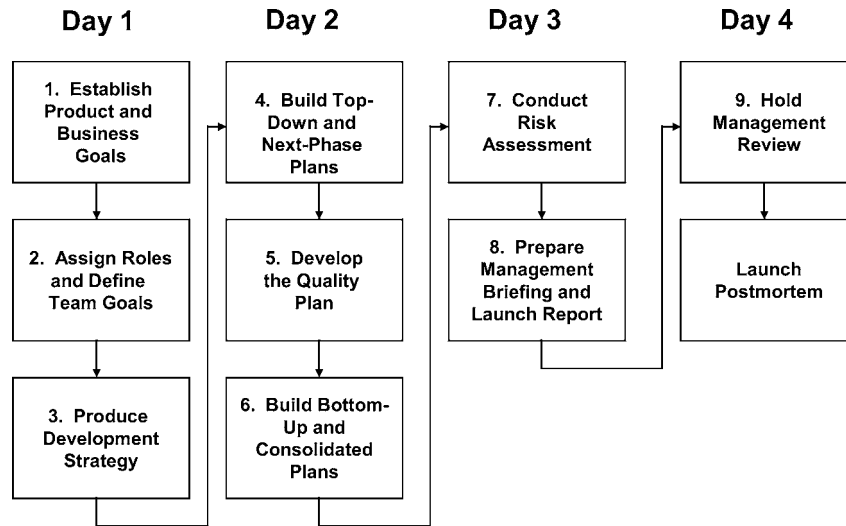


Figure 6. The TSP launch process.

6.4. The TSP project

After these initial TSP trials, we expanded the SEI staff on the PSP project to include the TSP work. Dan Burton joined Jim Over and I and somewhat later so did Noopur Davis, Marcia Pomeroy-Huff, Jim McHale, Don McAndrews, and Janice Marchock Ryan. Julia Mullaney, who had left to be in Japan with her husband, soon rejoined the group as well. Bob Musson, who had managed some early industry TSP projects, also joined the team a little later.

Over the next several years, we enhanced the TSP process to address these initial problems. While the PSP taught the engineers basic planning, data gathering, and quality management methods, the initial teams had not done a thorough job of planning their projects. It wasn't obvious to them how to apply the methods they had learned in the PSP course to a team project. To do quality work, engineers needed detailed plans and defined processes. Without the process, they cannot make detailed plans, take consistent measurements, or track their work against the plan. However, when engineers have a project to deliver, they are rarely willing to take the time to define a complex process, even when they know how to do so. We needed a TSP launch process to guide teams through defining their processes and making complete, precise, and detailed plans. This four-day TSP launch follows the steps shown in figure 6.

As we gained TSP experience, we added steps to address goals, plan and manage quality, assess risks, and track and report on the work. We also developed a family of training programs and a prototype support tool to help teams and their management introduce and use the TSP. Even with all this support, the discipline of the team's work remained a problem. While all the teams that used the TSP performed substantially better than before, they did not follow the process as faithfully as the first ERAU team.

As a consequence, their quality, cost, and cycle time performance was not what it could be. There was clearly room for improvement.

6.5. *TSP problems*

The ERAU students had done a good job because the faculty had told them that their grades depended on their following the process. While finishing the job was important, the key was to do the job properly. It is difficult to convince industrial managers to take this position. However, to deliver quality products on schedule and for their planned costs, the engineers must consistently do disciplined work. When schedule is management's principle measure, the engineers will not define their processes, plan their work, track their progress, or manage quality. Then, as history demonstrates, teams produce poor quality products on late schedules and for far more than their planned costs.

To convince management of the importance of process discipline, the SEI now offers a family of courses for every engineer and manager involved in a TSP project. While this has normally been enough to get initial support, the only way for managers to be truly convinced that process discipline is important is to see the results of their own TSP teams.

The biggest single problem with the TSP is training. With few exceptions, managers want the benefits the TSP provides but are reluctant to invest in the required training. Several groups have tried to use the TSP with untrained or partially trained teams and they have always failed. So far, we have not found a quicker or easier way that works. We now recommend that organizations implement the TSP properly or not even try it.

6.6. *TSP results*

We now know that the TSP process works when teams follow it. We also know that TSP teams will generally follow the process when the engineers and their managers are properly trained and when the teams are guided and supported by a qualified coach. The PSP shows the engineers how to do disciplined personal work and it provides the data to convince them that these practices will help them do better work. The TSP then provides the environment engineers need to consistently follow disciplined practices. We have also found that team performance improves dramatically the first time engineers use the TSP. Then, on second and subsequent projects, their performance improves even further. We don't yet have the data to show long term improvement trends or to judge how long this improvement trend should last.

Getting comparative data on TSP performance has been difficult. The problem is not the lack of TSP data but that few organizations have data on their current software groups. A before-and-after analysis is only possible for organizations that have before-and-after data. The TSP has now been used by many teams and the results have been surprisingly good. Figures 7–11 show quartile data from 28 teams in 4 organizations [McAndrews 2000]. That is, the bars in these figures show the performance range from

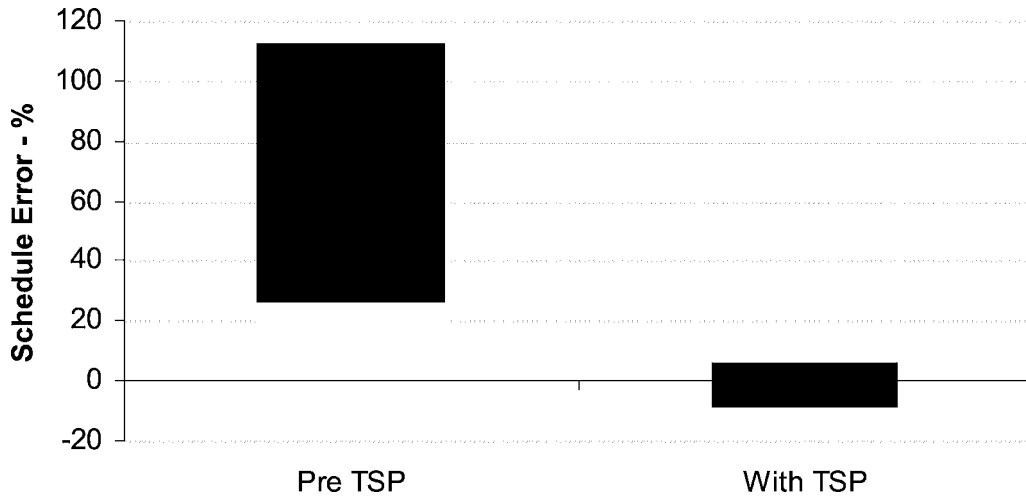


Figure 7. Schedule error.

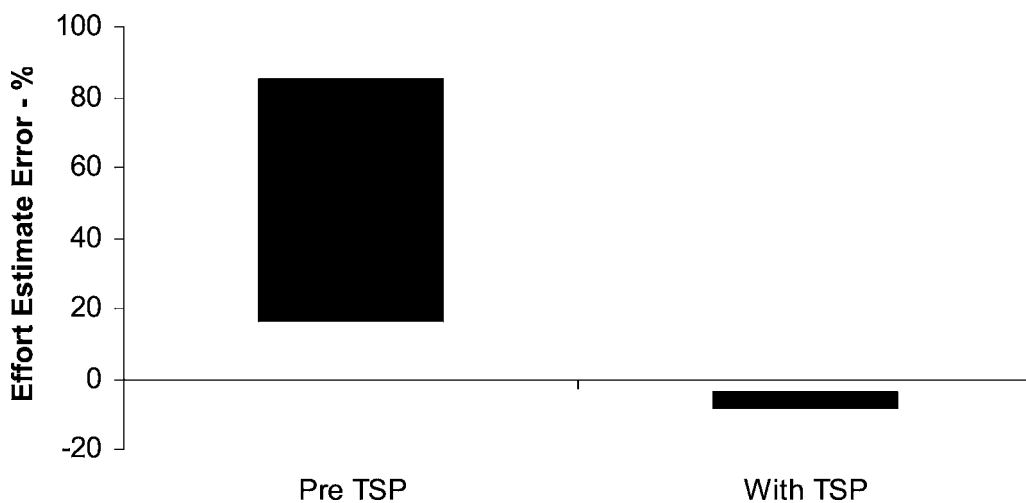


Figure 8. Effort estimate error.

the 25th percentile to the 75th percentile of the TSP and prior projects. Since these data are for first time TSP projects, we can expect even better results in the future.

Figure 7 shows that, before the TSP, schedule performance was typically late by 27% to 112% and with the TSP, schedules ranged from 8% early to 5% late. In figure 8, prior effort estimates were 17% to 85% low and with the TSP projects under expended by 4% to 8%. In figure 9, prior acceptance test defects ranged from 0.1 to 0.7 per KLOC and the TSP was from 0.02 to 0.1. In figure 10, test times were typically cut by 10 or more times and in figure 11, the delivered products often had no reported defects.

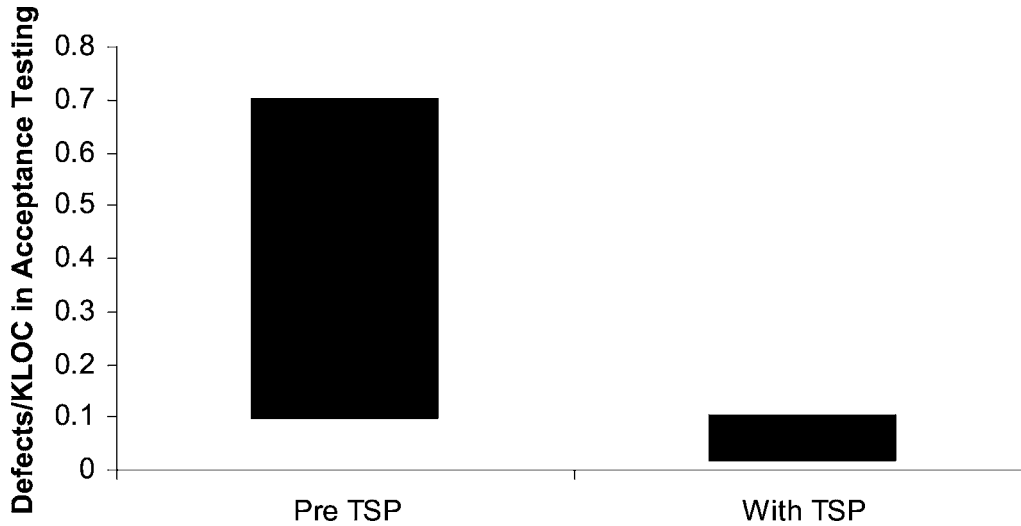


Figure 9. Reduced test defects.

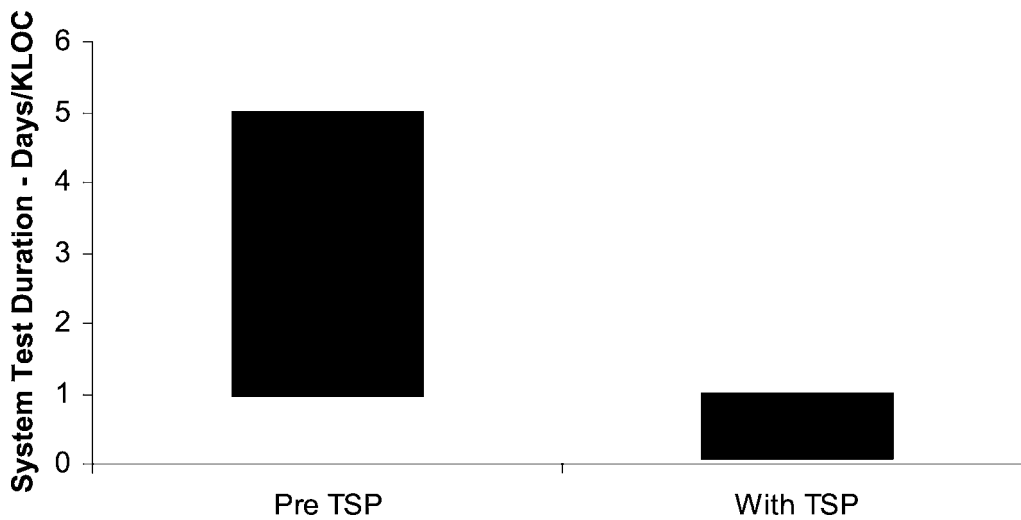


Figure 10. Reduced system test time.

7. Current status and remaining problems

While early TSP teams have been successful, there are remaining problems. The principal problems are getting the TSP more widely used, training time, building and sustaining excellence, and maintaining management priority. The following paragraphs address these topics. TSP enhancements and the longer term outlook are discussed in the succeeding sections.

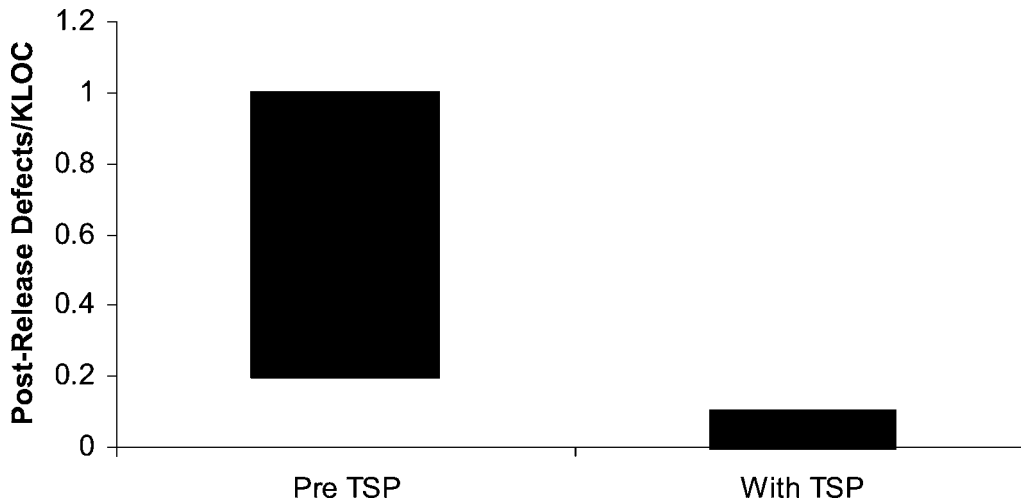


Figure 11. Improved product quality.

7.1. Training time

The length of a training program generally depends on the amount and complexity of the material, the time required to build new skills, and the practice required to overcome bad habits. The technical materials in the PSP and TSP are not complex and could be learned from reading a book or taking a brief tutorial. However, this is true for most technical subjects. Experience shows that new design methods, languages, tools, quality practices, or planning methods cannot be learned from a brief tutorial or an instruction manual. While people can learn from lectures and books, they will not generally apply what they have learned without practice, reinforcement, and coaching.

Most programmers learn to program in high school or college. When they take a first programming course, however, it is generally a programming language course. While the instructor may describe good methods, the students are graded on the programs and not on how they wrote them. This is like grading chemistry students on the compounds produced regardless of how the work was done. Can you imagine grading medical students on how the scar looked and not on the entire procedure?

Method is critical and the disciplined practice of sound methods is the only way to learn to do consistently high quality work. This is why it takes time to teach engineers the PSP. Until software engineering curricula require sound practices from the very beginning, we will continue to have software quality problems. Under these conditions, training in the disciplines required for the PSP and TSP will continue to take the two weeks it does today. This time is primarily required to overcome years of bad habits and to provide compelling evidence that sound methods work.

7.2. *Building and sustaining excellence*

The next challenge is to build the engineers' disciplines after they have been trained. Following disciplined practices is difficult, even for the most dedicated people. Witness the problems of championship athletes. That is why first class coaches are in such demand and why the coach is the key to a winning team. That is also why the support and guidance that Iraj and Soheil provided to the first TSP team at ERAU was so important.

Capable and skilled professionals are critical but the key is getting them to consistently follow sound practices. Even when engineers have the requisite training and even when they have a supportive working environment, few people can consistently do disciplined work. Someone must review their work, complement them on successes, and help them overcome problems. This is the role of the coach, and it is crucial.

TSP coaches are trained to work with managers and engineers, to monitor team performance, and to help team members do the work the way they know they should do it. Nobody likes to be criticized, so the coach must be supportive and help the engineers to do better work. To date this coaching strategy has been successful. However, the longer term challenge is for managers to learn more of a coaching style. This will take time because it requires a major change in the way managers work. Over time, as more coaches are promoted to management positions, this evolution will naturally follow. Then, instead of having frustrated designers as managers, we will have experienced coaches who know how to guide, motivate, and coach teams of software professionals. Perhaps, when TSP teams have managers with coaching experience, they will not need separate coaches. While this would be desirable in theory, only time will tell if it is practical.

7.3. *Maintaining management focus*

The problem of continuing management support stems from the original improvement motivation. If management launched the improvement effort to address cost and schedule problems, once these problems have been solved, the original improvement motivation will be gone. Then management will likely turn to other more pressing concerns. While this reaction is normal, the first evidence that an improvement method works is only an early indication of success. Building the discipline to continue using these new methods takes considerably more time.

Once organizations reach CMM levels 2, 3, 4, or even 5, management often feels that the problems have been solved and that they can reduce the improvement effort. Based on my experience, process improvement is a slippery slope: hard to climb up but easy to slide down. The challenge is to capitalize on the problems that motivated improvement. As soon as the improvement work is underway, start trying to convince management to maintain the improvement effort.

Various executive level training programs can maintain management's support through the TSP introduction steps. However, after the first projects have been launched, management must hold regular project reviews. The review data will demonstrate to them that, with the TSP, engineers make accurate plans, precisely track their work, and

accurately report on their progress. Such reviews will both maintain team motivation and help to sustain management's interest and support.

To maintain the required level of support, we need a way to maintain management commitment to improve. While no single action could completely solve this problem, success stories are most important [Humphrey 2002]. Another powerful motivator is a competitor's improvement efforts. The third and most reliable way to maintain management commitment is the customer. When the customer demands concrete evidence of process improvement, management is most likely to respond.

This motivation issue was addressed with the CMM when the US Air Force said it would use the CMM to select vendors for major system contracts. This got management's attention. To the extent that customers require quality work, improvement priorities will be maintained and improvement programs are most likely to continue. When customers don't care about quality or don't make their concerns known, process improvement will be a difficult proposition. Then the improvement must be tied to corporate quality policies, cost reduction programs, competitive capability, or growth opportunities.

Sophisticated customers are now recognizing the value of process improvement. That is why developing countries are so interested in the CMM, PSP, and TSP. They expect that better software methods will make them more competitive. They also know that better quality methods helped Toyota and other Japanese manufacturers become highly successful 20 years ago. As the PSP and TSP data show, such a strategy could also work for software.

8. Continuing the TSP transition

As we look to the future, the TSP must be enhanced to cover a broader range of engineering and non-engineering activities. We need to address the transition inhibitors and to build for the longer term. This requires a focus on education. The next paragraphs address TSP enhancements and some needed changes in how we educate software professionals.

8.1. Enhancing the TSP

The CMM, PSP, and TSP methods have worked surprisingly well. However, they do not yet form a complete and coherent package. The current PSP and TSP methods cover a range from the individual engineer to teams of 2 to 20 or so members. The TSPm process supports multiple teams of up to 100 to 150 members at single or multiple locations. The next step is to expand the TSP to truly large project teams with hundreds to thousands of members from multiple professional disciplines. It is important to expand the TSP to such massive programs because that is the only way to ensure that disciplined methods are consistently and successfully used. This is essential because these large programs are critical for military systems, have the widest economic impact, and are most likely

to cause loss of life or major economic damage. The work on these projects should be of the highest quality.

A further need is to broaden the TSP to more closely mesh with important areas like security, privacy, and COTS³-based development. A closer coupling with the CMM is also needed. While TSP enhancement can help to make these connections, these areas also need to couple more directly with the TSP. For example, the CMM needs process areas that address personal and team disciplines.

8.2. *The long term solution*

Because we must change ingrained working habits, almost any transition program will be time consuming and expensive. The longer term solution is to enlist the academic community. Then engineers will be properly trained when they start work and will not have to overcome bad habits. In working with ERAU, I have seen that sound methods can be taught from the very beginning of an academic program. Then the students understand the basic principles, appreciate the importance of quality, know how to make personal plans, and have the skill to do disciplined personal work.

To pursue this education strategy, I have worked with professors Iraj Hirmanpour and Tom Hilburn at ERAU. Tom taught an introductory programming course and agreed to introduce a simplified PSP course for college freshmen. We designed a simple course and I wrote a textbook to use with a first programming course [Humphrey 1997]. Tom has since used this material to teach freshman programming classes.

While there were some initial learning problems, the course has been very successful [Hilburn]. We initially made the mistake of describing the freshman PSP course as an experiment. Since students do not like being treated as guinea pigs, they complained. When the course was subsequently offered as the normal way to learn programming, these problems went away. The text has since been published and is being adopted by a number of universities and colleges.

PSP principles are now incorporated throughout the ERAU undergraduate software curriculum. The requirements, design, and database courses all use PSP methods. These are followed by an introductory TSP project course (called TSPi) in the Junior or Senior year [Humphrey 2000]. Professor Hilburn also helped to design this course which is now offered every year. At the end of the course, the student teams present brief final reports of their work. These reports are more concise and factual than any I have seen from traditional projects, even in industry. Out of the first two classes of about 40 students each, the 16 teams all completed their projects on time and produced working programs. In their final reports, they presented their plan and actual data and described their team roles and experiences.

While teaching students sound disciplines is effective, it requires a knowledgeable and committed faculty. Unfortunately, this is a long term problem that will require a long term solution. Getting disciplined methods adapted by the academic community requires

³ COTS stands for commercial off-the-shelf software.

that faculty members learn these methods and become committed to introducing them. This will take time.

While it is popular to think of universities as leaders in technology, that is generally only true for technologies that originate in the academic community. User driven advances like process improvement take longer to penetrate academia. These subjects do not generally involve exciting technical material but focus on the more pragmatic issues of teamwork, planning, and quality practices. Since current computer science curricula do not typically address such people-related topics, this change will likely take considerable time.

The best way to accelerate this academic transition would be for industry to fund process-related academic fellowships and research projects. We could then expect substantial academic interest. We know that the PSP and TSP disciplines can be taught and that the students are then better prepared for industrial work. Further, by starting with proper training, the industrial costs of introducing disciplined methods would be greatly reduced. Industry would greatly benefit from academic participation in this improvement crusade.

9. The future

As we look to a future when disciplined software methods are widely adopted, there will be new opportunities. In moving from intuitive software practices to a disciplined and scientifically based process, decisions can more often be based on facts and data. While it is impossible to predict where this will lead, the nature of software work suggests that the impact will be substantial. The best analogy I know is the flowering of the industrial age. Once disciplined and quantitative methods were adopted for manual work, the industrial revolution followed. As Peter Drucker has said, in the 100 years since Frederick Winslow Taylor introduced disciplined methods for manual work, the productivity of manual labor increased 50 times [Drucker 1999]. He adds “On this achievement rests all the economic and social gains of the 20th century.” It would be surprising if the introduction of defined and quantitative methods for intellectual work did not have a similarly profound effect.

Another way to view the software business is as a human based technology; what I refer to as a performing science. Here, substantial improvements depend on improving human performance. An enlightening sports example is running. The fastest time for the mile was 4.5 min in 1865. In 1954, the record was 4 min and it is now about 3 min 40 s. A plot of the world record for the mile is a straight line. At the current rate of improvement, the mile will be run in 3 min in 2116.

The key point is that human performance is essentially unlimited. This is not just for the mile but for every measured human activity. Can you imagine how much runners would have improved if there were no stop watch or measured track? Human performance steadily improves, but only with timely and measured feedback. Unmeasured performance does not improve. This applies to sports, manufacturing, and software. No

sport has yet found ultimate human limits. With the PSP and TSP we may find ultimate human limits, but I suspect they are farther away than anyone imagines.

Acknowledgements

The work described in this paper was built on the creative contributions of many of my predecessors. In particular, my IBM associates Mike Fagan, Harlan Mills, and Al Pietrasanta provided much of the intellectual foundation for everything I have since done in the software field and for much of what we have accomplished at the SEI. For that, I am eternally grateful.

In addition, many other people at IBM supported and guided my work through 27 years with the company. George Kennard was an inspirational quality leader and Bill Florac, Jack Harding, and Ron Radice were creative contributors to the quality and process work. They originated many of the ideas that have later been so successful in both process measurement and process improvement.

At the SEI, Larry Druffel led the institute during much of my early process improvement work and I am deeply indebted to him for his support and for his nominating me to be an SEI Fellow. That kindness provided the opportunity to develop and prove the concepts which have later become the PSP and TSP. Steve Cross, our current SEI Director, has continued to support my work, as have John Goodenough, Clyde Chittister, and Bill Peterson. For that I thank them all.

Most of all, I want to thank those who have joined with me to develop and introduce the PSP and TSP. Jim Over was the first to join in this work and his support has been invaluable. In fact, without his support, I do not believe we could have succeeded. I am also deeply grateful for the support of the entire TSP team. This included Dan Burton, Nooper Davis, Don McAndrews, Jim McHale, Julia Mullaney, Bob Musson, and Marsha Pomeroy-Huff.

Finally, this work would not have been possible without the efforts of countless people throughout the software community. I have already mentioned my associates at ERAU who have been so helpful, as well as the SEI team working on the PSP and TSP. The many software professionals who have studied and learned the PSP and worked with us as we have developed and improved the TSP have also been critically important. Without their support and participation, this work could not have been accomplished.

Last, I must thank the reviewers of this paper. Dan Burton, Nooper Davis, Don McAndrews, Julia Mullaney, and Bill Peterson have read drafts and made valuable comments and suggestions. My thanks to them all.

References

- Crosby, P.B. (1979), *Quality is Free*, New American Library, Mentor, New York.
Drucker, P.F. (1999), *Management Challenges for the 21st Century*, Harper Collins Publishers, New York.
Fagan, M. (1976), "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems Journal* 15, 3, 182-211.

- Ferguson, P., W.S. Humphrey, S. Khajenoori, S. Macke, and A. Matvya (1997), "Introducing the Personal Software Process: Three Industry Case Studies," *IEEE Computer* 30, 5, 24–31.
- Hayes, W. and J.W. Over (1997), "The Personal Software Process: An Empirical Study of the Impact of PSP on Individual Engineers," CMU/SEI-97-TR-001.
- Herbsleb, J., A. Carleton, J. Rozum, J. Siegel, and D. Zubrow (1994), "Benefits of CMM-Based Software Process," CMU/SEI-94-TR-13.
- Hilburn, T.B., "PSP in Academia and Training," FASE (Forum for Academic Software Engineering), <http://www.cs.ttu.edu/fase>
- Humphrey, W.S. (1986), "SPEAKOUT: We Can Program SDI," *IEEE Spectrum*, April, 16.
- Humphrey, W.S. (1988), "Characterizing the Software Process: A Maturity Framework," *IEEE Software*, March, 73–79.
- Humphrey, W.S. (1989), *Managing the Software Process*, Addison-Wesley, Reading, MA.
- Humphrey, W.S. (1995), *A Discipline for Software Engineering*, Addison-Wesley, Reading, MA.
- Humphrey, W.S. (1997), *Introduction to the Personal Software Process*, Addison-Wesley, Reading, MA.
- Humphrey, W.S. (2000), *Introduction to the Team Software Process*, Addison-Wesley, Reading, MA.
- Humphrey, W.S. (2002), *Winning with Software: An Executive Strategy*, Addison-Wesley, Reading, MA.
- Humphrey, W.S., T.R. Snyder, and R.R. Willis (1991), "Software Process Improvement at Hughes Aircraft," *IEEE Software*, July, 11–23.
- Humphrey, W.S. and W.L. Sweet (1987), "A Method for Assessing the Software Engineering Capability of Contractors," Technical Report CMU/SEI-87-TR-23, Software Engineering Institute, Carnegie Mellon University.
- Miller, M. (1973), *Plain Speaking: An Oral Biography of Harry S. Truman*, The Putnam Publishing Group, Berkley Publishing Group, New York.
- Mills, H.D., V.R. Basili, J.D. Gannon, and R.G. Hamlet (1987), *Principles of Computer Programming, A Mathematical Approach*, Allyn and Bacon, Newton, MA.
- McAndrews, D.R. (2000), "The Team Software Process (TSP): An Overview and Preliminary Results of Using Disciplined Practices," Technical Report CMU/SEI-2000-TR-015, ESC-TR-2000-105.
- Paulk, M.C., C.V. Weber, B. Curtis, and M.B. Chrissis (1995), *The Capability Maturity Model: Guidelines for Improving the Software Process*, Addison-Wesley, Reading, MA.
- Radice, R.A., J.T. Harding, P.E. Munnis, and R.W. Phillips (1985), "A Programming Process Study," *IBM Systems Journal* 24, 2, 91–101.