Pathways to Process Maturity: The Personal Software Process and Team Software Process

Watts S. Humphrey

Editor's Note: The following article is a condensed version of a three-part series that appeared in successive issues of *CrossTalk* from February to April 1998. Those issues are available online at http://stsc.hill.af.mil/CrossTalk.

Moving from "what" to "how"

Although the Capability Maturity Model® (CMM®) provides a powerful improvement framework, its focus is necessarily on "what" organizations should do and not "how" they should do it. This is a direct result of the CMM's original motivation to support the Department of Defense acquisition community. We knew management should set goals for their software work but we also knew that there were many ways to accomplish these goals. Above all, we knew no one was smart enough to define how to manage all software organizations. We thus kept the CMM focus on goals, with only generalized examples of the practices the goals implied.

As organizations used the CMM, many had trouble applying the CMM principles. In small groups, for example, it is not generally possible to have dedicated process specialists, so every engineer must participate at least part time in process improvement. We kept describing to engineers what they ought to do and they kept asking us how to do it. Not only did this imply a need for much greater process detail, it also required that we deal more explicitly with the real practices of development engineers. We needed to show them precisely how to apply the CMM process principles.

Improvement requires change, and changing the behavior of software engineers is a nontrivial problem. The reasons for this explain why process improvement is difficult and illustrate the logic behind the Personal Software Process^{SM} (PSP^{SM}).

A question of conviction

Software engineers develop their personal practices when they first learn to write programs. Since they are given little or no professional guidance on how to do the work, most engineers start off with exceedingly poor personal practices. As they gain experience, some engineers may change and improve their practices, but many do not. In general, the highly varied ways in which individual software engineers work are rarely based on a sound analysis of available methods and practices.

Engineers are understandably skeptical about changes to their work habits; although they may be willing to make a few minor changes, they will generally stick fairly closely to

what has worked for them in the past until they are convinced a new method will be more effective. This, however, is a chicken-and-egg problem: engineers only believe new methods work after they use them and see the results, but they will not use the methods until they believe they work.

The Personal Software Process

Given all this, how could we possibly convince engineers that a new method would work for them? The only way we could think of to change their behavior was with a major intervention. We had to directly expose the engineers to the new way of working. We thus decided to remove them from their day-to-day environment and put them through a rigorous training course. As shown in Figure 1, the engineers follow prescribed methods, represented as levels PSP0 through PSP3, and write a defined set of 10 programming exercises and five reports[1]. With each exercise, they are gradually introduced to various advanced software engineering methods. By measuring their own performance, the engineers can see the effect of these methods on their work.

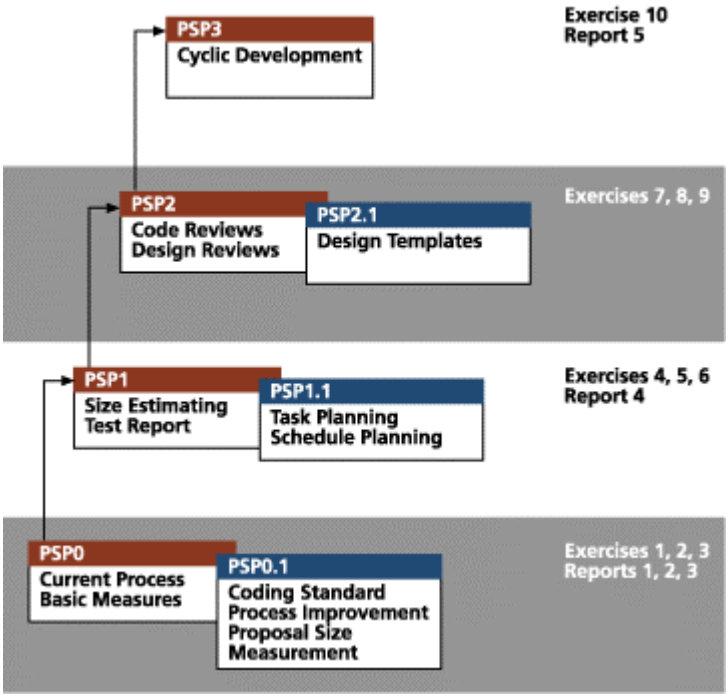

Figure 1: The PSP process evolution

Figures 2 through 4 show some of the benefits engineers experience[2] [3]. Figure 2 shows an improvement from a 55 percent estimating error to a 27 percent error or a factor of about two. As shown in Figure 3, the improvement in compile and test defects is most dramatic. From PSP0 to PSP3, the engineers' compile and test defects dropped from 110 defects per 1,000 lines of code (KLOC) to 20 defects per KLOC, or over five times. Figure 4

shows that even with their greatly improved planning and quality performance, the engineers' lines of code productivity was more or less constant.

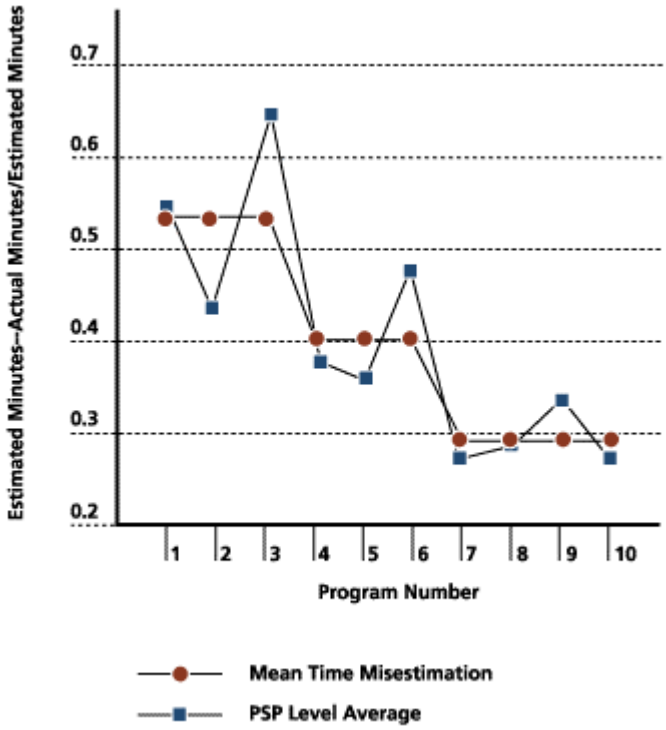

Figure 2: Effort estimation results

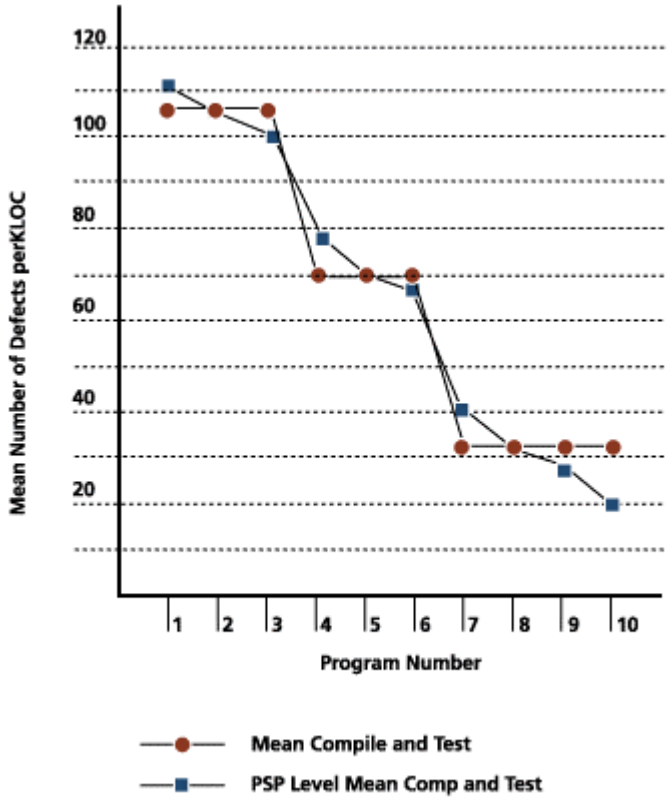

Figure 3: Quality results

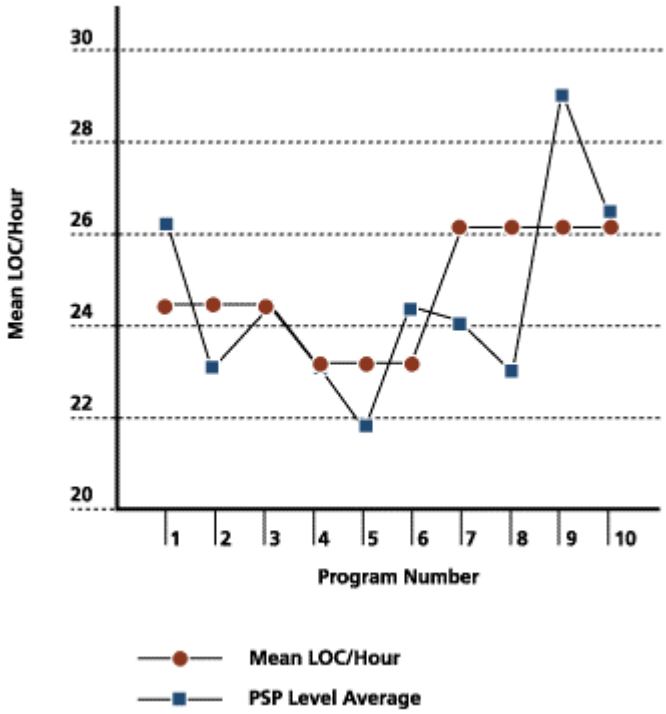

Figure 4: Productivity results

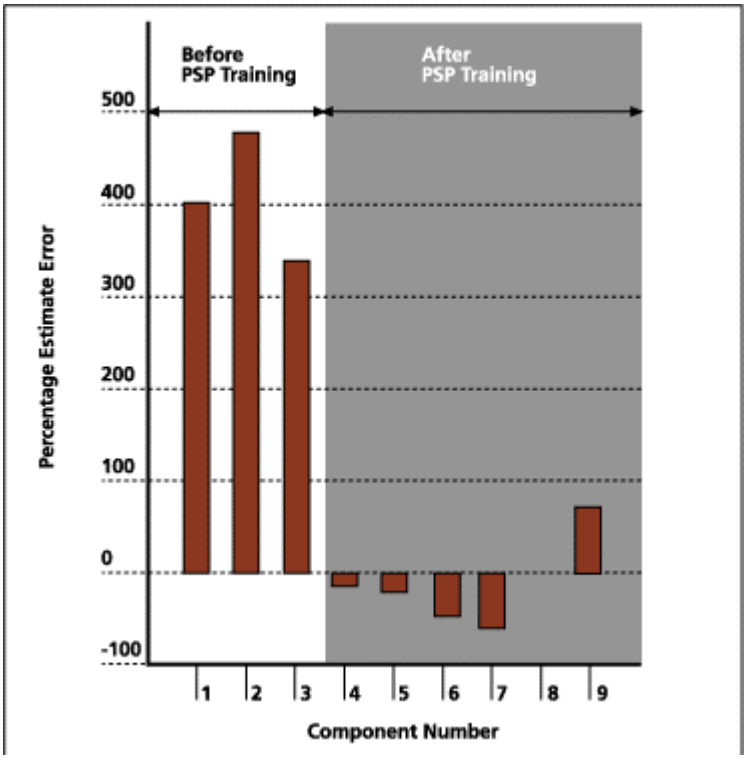

Figure 5: Schedule estimating error

Industrial results with the PSP

A growing number of organizations are using the PSP, such as Baan, Boeing, Motorola, and Teradyne. Data from some early users clearly demonstrate the benefits of PSP training[4]. Figure 5 shows data from a team at Advanced Information Services (AIS) in Peoria, Ill. Team members were PSP trained in the middle of their project. The three bars on the left of the chart show the engineers' time estimates for the weeks it would take them to develop the first three components. For Component 1, for example, the original estimate was four weeks, but the job took 20 weeks. Their average estimating error was 394 percent. After PSP training, these same engineers completed the remaining six components. As shown on the right, their average estimating error was -10.6 percent. The original estimate for Component 8, for example, was 14 weeks and the work was completed in 14 weeks.

Table 1 shows acceptance test data on products from one group of AIS engineers. Before PSP training, they had a substantial number of acceptance test defects and their products were uniformly late. After PSP training, the next product was nearly on schedule, and it had only one acceptance test defect. Table 2 shows the savings in system testing time for nine PSP projects. At the top of the chart, system test time is shown for several products that were completed before PSP training. At the bottom, system test time is shown for products the same AIS engineers completed after PSP training. Note that A1 and A2 are

two parts of the same product, so testing for them was done together in one and one-half months.

Not Using PSP	**KLOC**	**Months Late**	**Acceptance Test Defects**
1	24.6	9	N/A
2	20.8	4	168
3	19.9	3	21
4	13.4	8+	53
5	4.5	8+	25
Using PSP	**KLOC**	**Months Late**	**Acceptance Test Defects**
1	22.9	1	1

Table 1: Acceptance test improvement

System test time before PSP training

Project	Size	Test Time
A1	15,800 LOC	1.5 months
C	19 requirements	3 test cycles
D	30 requirements	2 months
H	30 requirements	2 months

System test time after PSP training

Project	Size	Test Time
A2	11,700 LOC	1.5 months
B	24 requirements	5 days
E	2,300 LOC	2 days
F	1,400 LOC	4 days
G	6,200 LOC	4 days
I	13,300 LOC	2 days

Table 2: System test time savings

Introducing the PSP

Although the PSP can be introduced quickly, it must also be done properly. First, the engineers need to be trained by a qualified PSP instructor. The SEI trains and authorizes PSP instructors and provides limited on-site PSP training. There is also a growing number of SEI-trained PSP instructors who offer commercial PSP training. [For more on

PSP training, please see the SEI Web site at http://www.sei.cmu.edu/psp/psp.html, or contact SEI Customer Relations at customer-relations@sei.cmu.edu.]

The second important step in PSP introduction is to train in groups or teams. When organizations ask for volunteers for PSP training, they get a sparse sprinkling of PSP skills that will generally have no impact on the performance of any project.

Third, effective PSP introduction requires strong management support. This, in turn, requires that management understand the PSP, know how to support their workers once they are trained, and regularly monitor their performance. Without proper management attention, many engineers gradually slip back into their old habits. The problem is that software engineers, like most professionals, find it difficult to consistently do disciplined work when nobody notices or cares. Software engineers need regular coaching and support to sustain high levels of personal performance.

The final issue is that even when a team of engineers are all PSP trained and properly supported, they still have to figure out how to combine their personal processes into an overall team process. We have found this to be a problem even at higher CMM levels. These are the reasons we are developing the Team Software Process^{SM} (TSP^{SM}).

Building a supportive team environment: the Team Software Process

The Team Software Process (TSP) extends and refines the CMM and PSP methods to guide engineers in their work on development and maintenance teams. It shows them how to build a self-directed team and how to perform as an effective team member. It also shows management how to guide and support these teams and how to maintain an environment that fosters high team performance. The TSP has five objectives:

- Build self-directed teams that plan and track their work, establish goals, and own their processes and plans. These can be pure software teams or integrated product teams (IPT) of three to about 20 engineers.
- Show managers how to coach and motivate their teams and how to help them sustain peak performance.
- Accelerate software process improvement by making CMM Level 5 behavior normal and expected.
- Provide improvement guidance to high-maturity organizations.
- Facilitate university teaching of industrial-grade team skills.

The principal benefit of the TSP is that it shows engineers how to produce quality products for planned costs and on aggressive schedules. It does this by showing engineers how to manage their work and by making them owners of their plans and processes.

Team-building strategies are not obvious

Generally, when a group of engineers starts a project, they get little or no guidance on how to proceed. If they are lucky, their manager or one or two of the experienced engineers will have worked on well-run teams and have some ideas on how to proceed. In most cases, however, the teams have to muddle through a host of issues on their own. Following are some of the questions every software team must address:

- What are our goals?
- What are the team roles and who will fill them?
- What are the responsibilities of these roles?
- How will the team make decisions and settle issues?
- What standards and procedures does the team need and how do we establish them?
- What are our quality objectives?
- How will we track quality performance, and what should we do if it falls short?
- What processes should we use to develop the product?
- What should be our development strategy?
- How should we produce the design?
- How should we integrate and test the product?
- How do we produce our development plan?
- How can we minimize the development schedule?
- What do we do if our plan does not meet management's objectives?
- How do we assess, track, and manage project risks?
- How can we determine project status?
- How do we report status to management and the customer?

Most teams waste a great deal of time and creative energy struggling with these questions. This is unfortunate, since none of these questions is new and there are known and proven answers for every one.

The TSP process

The TSP provides team projects with explicit guidance on how to accomplish their objectives. As shown in Figure 6, the TSP guides teams through the four typical phases of a project. These projects may start or end on any phase, or they can run from beginning to end. Before each phase, the team goes through a complete launch or

relaunch, where they plan and organize their work. Generally, once team members are PSP trained, a four-day launch workshop provides enough guidance for the team to complete a full project phase. Teams then need a two-day relaunch workshop to kick off the second and each subsequent phase. These launches are not training; they are part of the project.

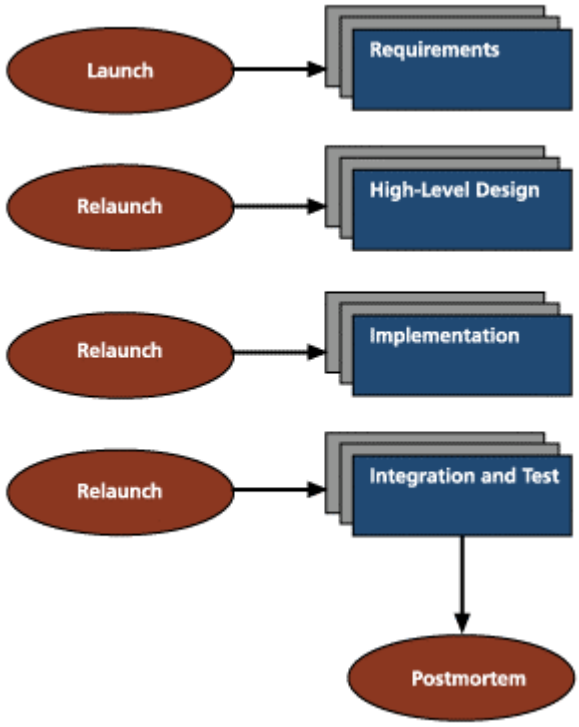

Figure 6: TSP structure

The TSP launch process

To start a TSP project, the launch process script leads teams through the following steps:

- Review project objectives with management.
- Establish team roles.
- Agree on and document the team's goals.
- Produce an overall development strategy.
- Define the team's development process.
- Plan for the needed support facilities.
- Make a development plan for the entire project.

- Make a quality plan and set quality targets.
- Make detailed plans for each engineer for the next phase.
- Merge the individual plans into a team plan.
- Rebalance team workload to achieve a minimum overall schedule.
- Assess project risks and assign tracking responsibility for each key risk.
- Hold a launch postmortem.

In the final launch step, the team reviews its plans and the project's key risks with management. Once the project starts, the team conducts weekly team meetings and periodically reports its status to management and to the customer.

In the four-day launch workshop, TSP teams produce

- written team goals
- defined team roles
- a process development plan
- the team quality plan
- the project's support plan
- an overall development plan and schedule
- detailed next-phase plans for each engineer
- a project risk assessment
- a project status report

Early TSP results

While the TSP is still in development, the early results are encouraging. One team at Embry-Riddle Aeronautical University removed more than 99 percent of development defects before system test entry. Their defect-removal profile is shown in Figure 7.

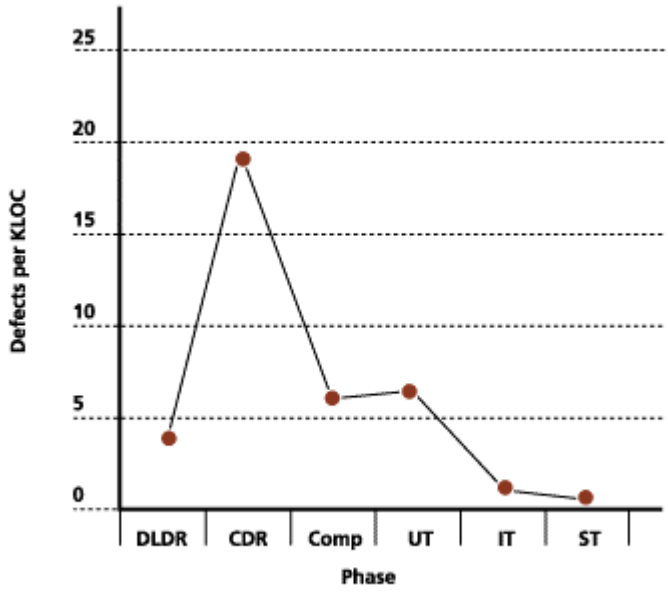

Figure 7: Defects per thousand lines of code (KLOC) removed by phase

Another team at Hill Air Force Base reduced testing time by eight times and more than doubled their productivity. The team's customer has since found no defects in using the product. [For more information about this case, see "Using the TSP on the TaskView Project" in the February 1999 issue of *CrossTalk*.]

TSP teams also gather the data they need to analyze component quality before integration and system testing.

How the TSP helps teams behave professionally

Perhaps the most powerful consequence of the TSP is the way it helps teams manage their working environment. The most common problem product teams face is unreasonable schedule pressure. Although this is normal, it can also be destructive. When teams are forced to work to unreasonable schedules, they are unable to make useful plans. Every plan they produce misses management's edicted schedule and is therefore unacceptable. As a result, they must work without the guidance of an orderly plan. Under these conditions, the team will generally take much longer to complete the project than they otherwise would.

The TSP team's responsibility is to plan and produce a quality product as rapidly and effectively as they can. Conversely, it is management's responsibility to start projects in time to finish when needed. When similar projects have taken 18 months and

management demands a nine-month schedule, this is clearly unrealistic. Where was management nine months ago when the project should have started? Although the business need may be real, the team's schedule is only part of the problem. Under these conditions, it is essential that management and the team work together to rationally determine the most effective course of action. This will often involve added resources, periodic replanning, or early attention to high-risk components.

While TSP teams must consider every rational means for accelerating their work, in the last analysis, they must defend their plan and resist edicts that they cannot devise a plan to meet. If management wants to change job scope, add resources, or suggest alternate approaches, the team will gladly develop a new plan. In the end, however, if the team cannot produce a plan to meet the desired schedule, they must not agree to the date. So far, most TSP teams have been able to do this. Teams have found that the TSP provides them convincing data to demonstrate that their plans are aggressive but achievable.

The TSP manager-coach

Perhaps the most serious problem with complex and challenging work is maintaining the discipline to consistently perform at your best. In sports and the performing arts, for example, we have long recognized the need for skilled trainers, conductors, and directors. Their job is to motivate and guide the performers and also to insist that everyone meet high personal standards. Although skilled players are essential, it is the coaches who consistently produce winning teams. There are many differences between software teams and athletic or artistic groups, but they all share a common need for sustained high performance. This requires coaching and support.

Software managers have not traditionally acted as coaches, but this is their role in the TSP. The manager's job is to provide the resources, interface to higher management, and resolve issues. But most important, the manager must motivate the team and maintain a relentless focus on quality and excellence. This requires daily interaction with the team and an absolute requirement that the process be followed, the data gathered, and the results analyzed. With these data, the manager and the team meet regularly to review their performance and to ensure their work meets their standards of excellence.

Conclusion

The CMM, PSP, and TSP provide an integrated three-dimensional framework for process improvement. As shown in Table 3, the CMM has 18 key process areas, and the PSP and TSP guide engineers in addressing almost all of them. These methods not only help engineers be more effective but also provide the in-depth understanding needed to accelerate organizational process improvement.

Level	Focus	Key Process Area	PSP	TSP
5 Optimizing	Continuous Process Improvement	Defect Prevention	X	X
		Technology Change Management	X	X
		Process Change Management	X	X
4 Managed	Product and Process Quality	Quantitative Process Management	X	X
		Software Quality Management		
3 Defined	Engineering Process	Organization Process Focus	X	X
		Organization Process Definition	X	X
		Training Program		
		Integrated Software Management	X	X
		Software Product Engineering	X	X
		Intergroup Coordination		X
		Peer Reviews	X	X
2 Repeatable	Project Management	Requirements Management	X	X
		Software Project Planning	X	X
		Software Project Tracking	X	X
		Software Quality Assurance		X
		Software Configuration Management		X
		Software Subcontract Management		

Table 3: PSP and TSP coverage of CMM key process areas

The CMM was originally developed to help the Department of Defense (DoD) identify competent software contractors. It has provided a useful framework for organizational assessment and a powerful stimulus for process improvement even beyond the DoD. The experiences of many organizations show that the CMM is effective in helping software organizations improve their performance.

Once groups have started process improvement and are on their way toward CMM Level 2, the PSP shows engineers how to address their tasks in a professional way. Although relatively new, the PSP has already shown its potential to improve engineers' ability to plan and track their work and to produce quality products.

Once engineering teams are PSP trained, they generally need help in applying advanced process methods to their projects. The TSP guides these teams in launching their projects

and in planning and managing their work. Perhaps most important, the TSP shows managers how to guide and coach their software teams to consistently perform at their best.

About the author

Watts S. Humphrey founded the Software Process Program at the SEI. He is a fellow of the institute and is a research scientist on its staff. From 1959 to 1986, he was associated with IBM Corporation, where he was director of programming quality and process. His publications include many technical papers and seven books. His most recent books are: Managing the Software Process (1989), A Discipline for Software Engineering (1995), Managing Technical People (1996), Introduction to the Personal Software Process[SM] (1997), and Introduction to the Team Software Process[SM] (in press). He holds five U.S. patents. He is a member of the Association for Computing Machinery, a fellow of the Institute for Electrical and Electronics Engineers, and a past member of the Malcolm Baldridge National Quality Award Board of Examiners. He holds a BS in physics from the University of Chicago, an MS in physics from the Illinois Institute of Technology, an MBA from the University of Chicago, and has been awarded an honorary Ph.D. in software engineering from Embry-Riddle Aeronautical University.

---

[1] Humphrey, W.S., *A Discipline for Software Engineering*, Addison-Wesley, Reading, Mass., 1995.

[2] Hayes, Will, "The Personal Software Process: An Empirical Study of the Impact of PSP on Individual Engineers," CMU/SEI-97-TR-001.

[3] Humphrey, W.S., "Using a Defined and Measured Personal Software Process," *IEEE Software*, May 1996.

[4] Ferguson, Pat, Watts S. Humphrey, Soheil Khajenoori, Susan Macke, and Annette Matvya, "Introducing the Personal Software Process: Three Industry Case Studies," *IEEE Computer, Vol. 30*, No. 5, May 1997, pp. 24-31.
