Personal Software Process Technology: An Experiential Report

Iraj Hirmanpour[1]
Soheil Khajenoori[2]
Department of Computing and Mathematics
Embry-Riddle Aeronautical University
600 Clyde Morris Boulevard
Daytona Beach, Florida 32114

ABSTRACT

Process improvements within software development occur at three different levels: the organizational level, the project/team level, and at the individual engineer -- or personal -- level. The Software Engineering Institute (SEI) of Carnegie Mellon University has developed process improvement models tailored to each of these levels. The Capability Maturity Model (CMM)[3] deals with organization issues, the Team Software Process (TSP)[1], currently under validation testing, address improvements in project or team development processes, and the Personal Software Process (PSP)[1]. The focus of this paper is on individual software engineer's issues addressed by the PSP. The Personal Software Process (PSP) provides a framework that individual software engineers can use to define, instrument, and continuously improve their individual processes.

After five years of experience in teaching PSP in both academic and industry settings, we have gained some insights into the challenges and rewards of transitioning this technology into an organization's software development practices. Our industrial experiences included work with the Motorola Paging Products Group; Boeing Corporation's Space Division and the Naval Oceanographic Office (NAVO).

In this paper, we will relate our experiences with the transition of PSP technology into these three organizations. We will describe various approaches taken with industrial PSP training, and report data to validate the benefits of PSP. We will further describe some barriers to PSP training, the challenges of post-training activities, and offer conclusions about the transition process.

Keywords: Personal Software Process (PSP), Defect management, Defect Prevention, CMM, TSP

1. INTRODUCTION

There is a notion that the cornerstone of the software quality movement lies in quantitative process management. This belief has been promulgated by many in the software development field through the introduction of improvement models such as CMM [Paulk 1995], SPICE, and Trillium [Bell 1994], to mention a few. Process improvement, however, commonly occurs within software development at three different levels: the organizational level, the project/team level, and/or at the individual engineer's level. The Software Engineering Institute (SEI) of Carnegie Mellon University has developed process improvement models for each of the three levels. The Capability Maturity Model (CMM) deals with organizational issues [Paulk 1995], the Team Software Process (TSP), currently under test, deals with project/team issues [Humphrey 1999], and the Personal Software Process (PSP) deals with individual software engineer's issues [Humphrey 1995].

1 Ih@db.erau.edu

2 soheil@db.erau.edu

3 CMM, TSP, and PSP are service marks of Carnegie Mellon University

The CMM embodies process improvement at the organizational level, and tends to address quality issues more from a managerial perspective rather than at the work level of individual software engineers. However, the application of CMM has proven effective [Herbsleb 1997] in improving the quality, schedule, and costs of software [Herbsleb 1997].

" Personal work processes are rarely even considered part of the management business process, let alone analyzed and perfected."

Kerry Gleeson

The PSP method was developed by Watts Humphrey, introduced by SEI in 1995, and published in " A Discipline for Software Engineering" [Humphrey 1995]. The goal of PSP is to provide individuals with the same type of tools CMM offers to organizations, in other words, a roadmap for a disciplined approach to software development. TSP, also developed by Watts Humphrey and soon to be released by SEI, will complete the cycle by providing team-oriented processes for PSP-practitioners, thereby enabling them to use level five practices in a team project setting. CMM describes *what* software development practices an organization should implement to reach a higher level of maturity and effectiveness. PSP shows individual engineers *how* to deploy CMM level 5 practices for the same beneficial outcomes.

Our premise is that further improvements can be achieved when a *personal* quality model such as PSP is used. We hypothesize that the Personal Software Process provides an effective quality model for individual software engineers to employ. The notion of personal quality, new to software development, is not new in other disciplines. Bob Galvin, a former Motorola CEO, promoted the practice by espousing, "*quality improvement is not just an institutional assignment-- it is a daily personal priority obligation*."

In this paper, we share our experiences with the transition of PSP technology into the aforementioned organizations, describing various approaches taken with the industrial PSP training, and reporting data to validate the benefits of PSP. We go one step further to describe barriers experienced during the PSP training, the challenges of post-training activities and summarize with conclusions.

2. WHAT IS PSP?

Personal Software Process (PSP) provides a framework that individual engineers can use to define, instrument, and continuously improve, their individual software development processes. It consists of a family of defined, measurement-based processes, organized in an evolutionary path that teaches software engineers to practice personal quality management and personal project management. Individuals are introduced to the PSP through a series of seven process steps. Following these steps, engineers generate up to ten small programs and prepare five reports using the evolving PSP methods. They are also responsible for gathering and analyzing data from their own work. Thus, the PSP framework provides a mechanism for software engineers to apply CMM level-5 process principles to their individual work. These principles include:

- Time management,
- Defect management,
- Estimation,
- Planning and tracking,
- Establishment and utilization of standards,
- Data analysis,
- Quantitative process improvement,

Covering principals and practices of 12 of the 18 key process areas of the CMM.

3. BARRIERS TO PSP IMPLEMENTATION

The focus of the PSP technology is on personal work processes, which traditionally have not been considered as a part of management business processes. Since there are, thus far, few quantitative process improvement experiences at the organizational level, and even fewer at the personal level, convincing engineers and managers to learn new practices is a difficult task. A cultural change of this nature is slow and requires a long-term commitment from both management and engineers alike.

Therefore, the education and practice of PSP presents a considerable challenge both to the individuals and to their organization. The most common barriers are the reluctance of many engineers to accept a new method of developing software and a lack of long-term commitment by management to support PSP efforts. The challenge of transition to using PSP is far greater than just delivering the training course; it is affecting a cultural and behavioral change from a "common culture" to a "PSP culture". The chart below illustrates the various components related to such a change.

Common Culture *(Engineers favor)*	**PSP Culture** *(PSP-practitioners favor)*
• Test over reviews	• Reviews over test
• Code over design	• Design over code
• "Jumping-in" over planning	• Planning over "jumping-in"
• "Guesstimates" over estimates	• Estimates over guestimates"
• Opinion over facts	• Facts over opinion

The question arises of how to move from a "common culture" to a "PSP culture." Training and education are the primary components of such a journey. In what follows, we describe our experiences and share lessons learned while completing the PSP training sequence.

4. EXPERIENCES WITH PSP TRAINING

The industrial PSP training began in the summer of 1995 at Motorola Paging Product Group with a pilot course taught to a group of managers and engineers [Macke 1996]. Since then, four additional courses have been taught at Motorola PPG, two courses at the Boeing Corporation at Kennedy Space Center, and one course at the Naval Oceanographic Office (NAVO).

The first offering of the full PSP course was taught over twelve weeks, one day per week. During the duration of the course, engineers completed ten programs and prepared five reports. Data has shown that students spent over 150 hours to complete the entire course. This sizable utilization of resources was deemed excessive by both management and engineers. As time went on, we experimented with several different approaches such as shortening the duration of the course, reducing the number of programming assignments from ten to seven, providing additional learning aids to clarify the assignment requirements, and offering individual assistance to students in completing their assignments. These implemented changes have reduced the amount of time spent by students on the course.

The most effective reduction in time spent by students was found to occur with a shortened format we call the "3-3-4 format". This format was most recently implemented at NAVO. The 3-3-4 format consists of offering the course three consecutive days one week, three days several weeks later with a four day "wind up" several weeks later**.** Total time spent by students for the entire course was reduced by almost 50 hours. This means that students spend approximately 100 hours on the course, including class time. The data collected from the shortened version of the course suggests that there is no reduction in the quality and effectiveness using this format over the more demanding twelve-week course**.** This shortened 3-3-4 format divides the course into three logical "fragments" as shown below:

- Module I: PSP0 Software Measurement – 3 days
- Module II: PSP1 Planning and Tracking – 3 days
- Module IIIa: PSP2 Quality Management - 3 days
- Module IIIb: Final reports & graduation ceremony – 1 day

This 3-3-4 format is considered advantageous for the following reasons. It separates three associated modules. This provides students with ample opportunity to learn the concepts of each module and allows them to practice the concepts of the module for a period of time before going on to the next one. This logical segmentation of the course has exhibited benefits to both the learner and the instructor.

5. PSP TRAINING BENEFITS

As most software engineers are aware, the most troublesome issue of software engineering research is collection and analysis of accurate data that will show the effectiveness of a new method or tool. In an effort to study the effectiveness of PSP training, three sets of data were analyzed: 1) Data collected from the first two classes taught at Motorola in Summer 95 and Winter 96 consisting of 24 engineers[1], 2) Data drawn from actual post-training projects by six of the above 24 engineers; and, 3) Data collected from a survey instrument administered six months after each Motorola class to evaluate the respondents attitudes toward the concepts learned. While this type of research effort is more similar to a case study than a statistical sampling, an effort is being made after each course is completed to validate whether concepts taught are being learned and then finally implemented. We realize the further study and validation of this type of data by other colleagues in the discipline is sorely needed.

PSP practices are designed to assist software engineers with quality and predictability issues. The quality component of the PSP strategy focuses on managing the defects in the software being produced. It is assumed that by improving defect management, engineers can produce more consistently reliable components of their software [Humphrey 95]. Using this limited definition of quality, one of the research questions became "*Do PSP- practitioners produce higher quality software*?" To answer this question, we can examine the data collected by PSP-students during the two Motorola classes and from the post-training projects.

Class data:

The population of the group under study included the previously mentioned 24 Motorola engineers enrolled in the first two of twelve training classes. The data presented compares the student engineers' work at the beginning of the course (the first three programming assignments) and at the end of the course (the last three programming assignments).

Table 1 shows data for in-process defects per thousand line of code (KLOC) from both classes. The average defect rate during the first three PSP programming assignments (where students are still relying heavily on their past software process) are compared to the last three PSP programming assignments (where students are now more effectively using the PSP processes they have learned).

1 This type of data is a by-product of the course and its collection

	Def/KLOC Prog. 1, 2, 3	Def/KLOC Prog. 8, 9, 10	% I
Class#1	93	66	29
Class#2	50	40	20
Total (Δ)	72	52	27.7
			I = Improvement

Table 1: Total Defects

As shown in Table 1, the defect rates declined in both classes. The average number of defects of the two classes represented in KLOC for the first three assignments in two classes was 72 and the average defects per KLOC for the last three assignments was 52; this represents over 27% improvement in overall defects. This trend, by the way, is consistent in all classes we have analyzed.

Table 2 shows compile defects found for the same two classes as in Table I.

	Comp. Defects Prog. 1, 2, 3	Comp. Defects Prog. 8, 9, 10	% I
Class#1	32	20	37.5
Class#2	40	8	80
Total (Δ)	36	14	61
			I = Improvement

Table 2: Compile Defects

As shown in Table 2, the average of the compile defects for the two classes was found to be 36 KLOC at the beginning of the course after the first three assignments and only 14 KLOC at the end of the course while completing the last three assignments. This is an improvement in early compile defect removal of over 60%. Since the compile defect rate is an early indicator of the quality of the software under development, obtaining such information early on is a valuable tool in deciding whether to proceed onto testing or to return to a previous phase for re-work which would avoid being bogged down in the testing phase.

Finally, Table 3 shows the test defect rate for the same two classes.

	Test Defects Prog. 1, 2, 3	Test Defects Prog. 8, 9, 10	% I
Class#1	44	16	63.6
Class#2	32	12	62.5
Total (Δ)	38	14	63.2
			I = Improvement

Table 3: Test Defects

As shown in Table 3, the average of the test defects found for the two classes was 38 KLOC at the beginning of the course, reducing to 14 KLOC at the end of the course; this represents an improvement of 63.2%.

If one agrees with the premise that there is a direct relationship between the in-process defect rate and the post-deployment defect rate, such as the higher in-process defect rate correlates with the higher probability of post-deployment difficulties, then one can conclude from the results presented in the tables that PSP teaches a set of practices that enable engineers to build higher quality (lower defect) software. In other words, they learn to build quality into the software instead of relying on testing to remove existing defects. Considering the sample size of the data, it is evident that there is a necessity for further studies. However, the notion of substantial improvement as observed in the present data can be further confirmed by a second set of data collected from real software projects at the conclusion of the training.

Project Data:

Six Motorola engineers from the two classes in this study volunteered to collect and share data on their current software projects. The data collected from these six engineers comes from one or two-person small software projects, mostly in a maintenance environment, giving a total of 18 projects. These projects collectively consisted of 25,114 lines of code (LOC) using 2,597 hours of engineer time, as shown in Table 4.

Proj.	LOC	Develop. Hours	Test Defects	Total Defects
18	25,114	2,597	136	575

Total Defect Density	22.8 defects/KLOC
Test Defect Density	5.4 defects/KLOC
Productivity	9.6 LOC/hour

Table 4: Profile of 24 Projects by 6 Engineers

Table 4 shows that the defect rate for eighteen actual projects as 22.8 defects/KLOC at the beginning of the project compared to 52 defects/KLOC at the end of the class (Table 1). This represents over a 60% improvement. Also, the test defect rate at the end of class was 14 defects/KLOC (Table 2), whereas the test defect rate over the eighteen projects is 5.4; again, over a 60% improvement. Both sets of data suggest significant improvement in quality taking place as engineers' master PSP practices and use them repeatedly in their software practices.

Survey Data:

In order to determine the effect of PSP education on attitude and work habits of engineers, a survey was conducted six months after completion of each class. The population surveyed consisted of the same 24 Motorola engineers who provided the data reported in Tables 1, 2, and 3 for the first two classes, including the six engineers who volunteered the project data reported in Table 4. The survey instrument was designed to query the engineers' opinion about the value of the course and to gauge behavioral changes in their work habits. Table 5 shows partial results from this opinion survey.

Sample Survey Questions	Agree	%
I have better quantitative knowledge of how to improve my work habits.	21/24	87
I pay more attention to defect management than before	19/24	79
I conduct personal code reviews	21/24	87
I have better insight into how my projects are progressing	17/24	70

Table 5: Engineer's Observations and Opinions

As shown in Table 5, 87% of the PSP-trained engineers polled in the survey stated that they have acquired a better understanding of the value of quantitative software engineering. A similar percentage indicated that they practice personal code reviews. Seventy percent of the surveyed engineers report better insight into project progress, while 79% report they paid more attention to defect management, after the course, than they did before taking part in the course. These reported increases suggest that PSP-trained engineers are adopting more of software "best practices," with a potential concomitant increase in efficiency and quality.

6. SUCCESS FACTORS

As mentioned earlier, industrial implementation of PSP is difficult and there are many barriers to overcome. Through experience, we have learned a number of factors that contributes to successful training and transition. These include training support, coaching and follow-ups after the training, as well as visibility and management support for continued use of the learned practices.

Training support means availability of equipment and software tools, allocation of sufficient time for completing course assignments, and mentoring and coaching support. A measure of successful training is the student completion rate. A student is considered to have completed the course when he/she has completed all programming assignments, prepared all reports requested, and been evaluated by the instructor as "satisfactory". Only completed assignments are accepted; incomplete assignments are returned to the student for re-work. Interestingly, regardless of course format, we have seen no significant differences in the completion rate of the course across the three organizations studied. We have consistently achieved over 90% completion rate, irrespective of the format used or the type of organization in which the training took place. We attribute this success rate to the process model used to implement the training/education program. The model consists of a detailed plan prepared in coordination with the managers of the corporate sponsors, an awareness seminar conducted with the prospective students, establishing expectations, requiring commitments from both managers and engineers, and finally not accepting incomplete work. Management plays an intregal part during the entire training course. They are briefed regularly on the engineers' progress and asked to intervene when assignments are not handed in on time. During the last day of the class, the trained engineers present their data before the entire class and management and are asked to reflect upon their learning experiences and describe how they intend to utilize the PSP concepts they have learned in their future individual projects.

In some cases, we continue to work with the trained engineers via coaching and community building. *Coaching* is conducted to encourage and assist the engineers to continue to use the PSP concepts in developing software. A pro-active coaching strategy promotes adaptation of the PSP concepts in the work environment. PSP graduates are contacted on a regular basis to determine the extent to which they are using the PSP concepts and to suggest new approaches for improving their work processes.

Visibility and management support is another success factor. In order to ensure continued interaction and the exchange of information among PSP-trained engineers and managers, a PSP users' group is formed by PSP graduates in an effort to *build a community* of PSP practitioners. In this case, the group meets once a month for lunch to share their experiences and the lessons they have learned. Often an engineer who has just completed a project using PSP makes a presentation and shares his/her data, experiences, and success/failures. The management supports these activities by attending, providing refreshments, and giving letters of commendation for exemplary work. As more engineers complete the course, this cycle of "train and join" eventually builds the community, and as a final end product, the "new culture".

7. CONCLUSIONS

Our experience and the limited data collected thus far suggests that applying PSP practices improves product quality and reduces cycle time. This assertion is further verified by a study conducted and published by SEI [Hays 1997]. Our survey data also suggests the trend of acceptance of the PSP methods by software engineers. It indicates improvement in the quality of work habits deployed by the PSP-trained engineers and suggests that engineers who learn PSP are better able to follow and reap the benefits of a defined, measured process.

However, as noted, introducing the PSP technology into the software development practices of an organization often proves to be a difficult task. It requires extensive resources and a long-term commitment in order to see a tangible benefit.

Further studies are needed to answer questions such as: What can be done to reduce the effort required in learning PSP? How can PSP be deployed into an organization's present software development practices?

What is the Return on Investment (ROI), among others?

8. ACKNOWLEDGEMENT

We would like to acknowledge influence, guidance and inspiration of Watts Humphrey. We have had the privilege of working closely with Dr. Humphrey Watts and have benefited from his insights into the software development process.

9. REFERENCES

Bell Canada, 1994, Trillium: Model for Telecom Product Development & Support Process Capability, Bell Canada.

Gleeson, K., 1994, The Personal Efficiency Program. John Wiley & Sons, New York.

Hayes, Will and James, W. Over, 1997, The Personal Software Process (PSP): An Empirical study of the Impact of PSP on Individual Engineers, (CMU/SEI-97-TR-001), Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.

Herbsleb, T., 1997, Benefits of CMM-Based Software Process Improvement, (CMU/SEI-97-TR-001), Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.

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., May, 1996, Using a Defined and Measured Personal Process, IEEE *Software*.

Humphrey, W.S., 1999, Introduction to Team Software Process, Addison Wesley, Reading, MA.

Khajenoori, S and I. Hirmanpour, April, 1995, "Personal Software Process: An Experimental Report," Proceedings of 8th Conference on Software Engineering Education, New Orleans, LA.

Khajenoori, S and I. Hirmanpour, October, 1995, "An Experiential Report On The Implications of Personal Software Process For Software Quality Improvement," Proceedings of the Fifth International Conference on Software Quality, Austin, TX.

Macke, S., Khajenoori, S and I. Hirmanpour, April, 1996, .An Industry/Academic Partnership that Worked: An In-Progress Report, Proceedings of 9th Conference on Software Engineering Education, Daytona Beach, FL.

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.

Roberst, H.V. and B.F. Sergesketter, 1993, Quality is Personal, The Free Press, New York, NY.