Suggestions for an Alternate Introduction Method to the SEI PSP™

Steven Teleki
Director, Software Development
Webify Solutions, Inc.
teleki@computer.org

***Abstract*- The Personal Software Process™ (PSP™) framework designed by Watts Humphrey from the Software Engineering Institute (SEI) enables software engineers to achieve outstanding software development performance. Introducing the PSP into a software group following the SEI recommended introduction method is challenging. By changing the introduction method: to account for the software engineers' individual learning speeds and styles, and to incorporate the wisdom of the ages of focusing on one skill at a time and practicing it until it is learned, all of the knowledge required to implement the PSP framework can still be learned. Once the PSP framework is established the team can work according to the Team Software Process™ (TSP™) principles. With this knowledge even people in small organizations can achieve high levels of software development performance.**

I. Introduction

In this paper I review results from the Personal Software Process (PSP) class to show that the classes are effective for improving software development performance as it is applied to common software problems used in the class; review results for Team Software Process (TSP) projects from the industry to show that once the TSP is introduced into an organization the results are very good; describe problems encountered during introducing the PSP; and make suggestions toward an alternate introduction method.

According to Humphrey "The Personal Software Process™ (PSP™) is a self-improvement process designed to help you control, manage, and improve the way your work. It is a structured framework of forms, guidelines, and procedures for developing software." [1] Also according to Humphrey [2] the need for the TSP arose because most of today's software problems can only be solved feasibly by a team of software developers. The goal of TSP is to enable a group of software engineers that understand their performance (because they apply the PSP framework) to be formed into a jelled, high-performance software team. A spreadsheet [3] accompanies the TSP book [2] that can help practitioners with the data collection and analysis.

There are two assumptions made in this paper:

1. The PSP and TSP are effective at producing great results. Once engineers apply the PSP framework and form TSP teams, to their everyday work, they get extraordinary results.

2. The SEI-recommended PSP introduction method fails to work under certain circumstances. The recommended method fails mostly for organizations that consider themselves small and on the leading edge. These are the organizations that cannot afford dedicated people to devise new exercises and tools to make necessary customizations to the PSP introduction method. Therefore these types of organizations need guidance to introduce the concepts embodied in the PSP and TSP into their everyday work.

II. PSP and TSP Results

The first book on PSP [1] was published in 1995, but several organizations have been using the PSP even before that. There was a PSP class taught at Carnegie Mellon University during 1994. The PSP is now over a decade old.

A. *Personal Software Process (PSP) Class Results*

The following chart from [4] shows a typical outcome of engineers taking the PSP class. Their performance at writing the programming assignments improves dramatically over the course of the class.

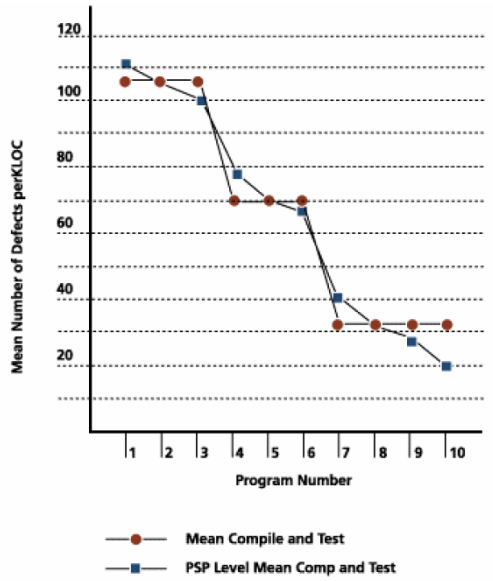

Fig.1. Mean # of Defects/KLOC in Compile and Test [4]

The data I have from teaching the PSP class to over 100 software engineers supports the data published by the SEI.

B. *Team Software Process (TSP) Industry Results*

In the late nineties the SEI started pilot projects for the TSP. I was part of one of these early pilot projects and I experienced first hand the benefits of both PSP and TSP.

The use of PSP and TSP significantly reduces acceptance test defect density (Fig.2) and reduces the average schedule deviation (Fig.3).

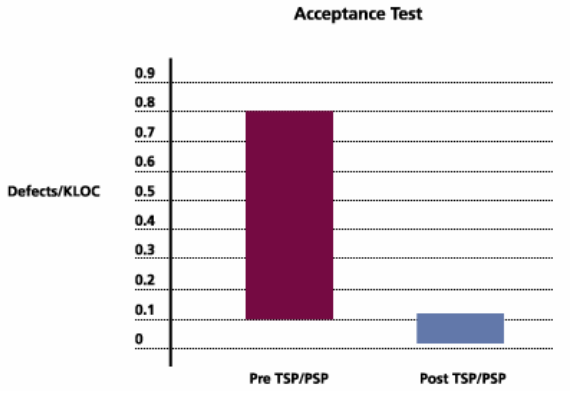

Fig.2. Acceptance Test Defect Density [5]

One of the difficulties in assessing some of the benefits of TSP is that some organizations that adopted the TSP had no prior data on their performance. Of those organizations that had data, many of them have shown impressive improvements like the ones shown in Fig.2 & Fig.3.

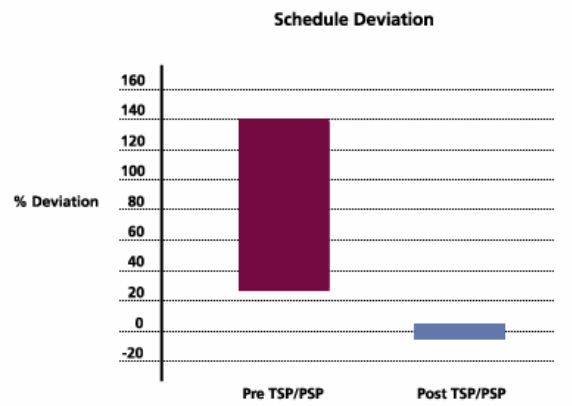

Fig.3. Average Schedule Deviation [5]

For a more complete list of results consult [4] and [5] and visit the SEI website at www.sei.cmu.edu/tsp.

III. PSP Introduction Difficulties

The PSP works once the practitioners have learned how to apply the principles to their day-to-day work. Unfortunately, getting to the point where the engineers have figured out how to apply the PSP framework is difficult.

A. The PSP Introduction Method Recommended by the SEI

The PSP introduction method recommended by the SEI follows Humphrey's original recommendation [1]. The PSP is introduced through a set of 7 processes (PSP0 through PSP3) applied to the 10 programming assignments and 5 reports that the learner has to write.

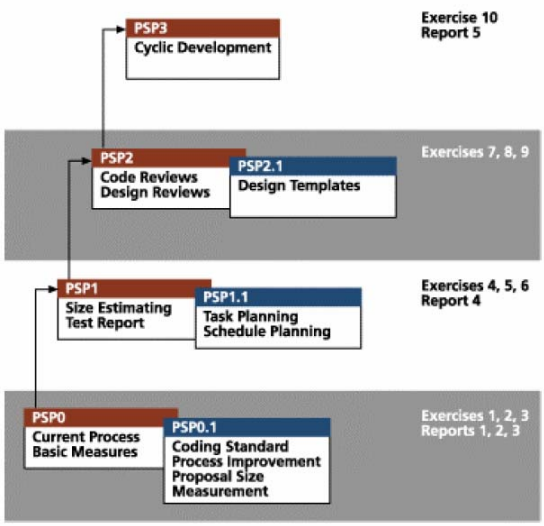

Fig.4. The PSP Evolution [4], [1]

The PSP class has been taught in many variations. One is a 2-3 week long class where the learners go through all assignments and reports one after another, always completing the assignment before moving on to the next one. Another version is where the learners go through Part I: Planning (assignments 1 through 6) during a week or 7 work days, followed by a few weeks of break, then another session for Part II: Quality (assignments 7 through 10 and the final report).

A common format is 2x5 days where the first work week is dedicated to Planning, the second week to Quality. During this intense week it is next to impossible for most folks to complete their assignments, thus the assignments pile up, and people end up doing them during weekends or after work, when they are not at their best.

B. Problems Encountered with the Recommended Method

Teaching the PSP class to over 100 engineers in companies that considered themselves on the leading edge, demonstrated to me that the knowledge that people are expected to learn by the end of the class is useful, but the introduction doesn't work as expected. I came to realize that the problems I experienced were not just "whining" from engineers who don't want to change the way they have been working. They genuinely didn't understand how to make the connection from the programs that they write in the class to the work they do every day.

Here are some of the problems that I have encountered:

The class is too disruptive to regular business because of its format. Everybody who attempted to convince an organization to go ahead with the class has heard this complaint before. The more a software organization needs the knowledge embodied in the PSP framework, the more the organization is not capable of taking a break to learn. If they would be capable to stop to learn, they wouldn't be in the predicament that they are today.

Class tools and work tools are different. Often engineers taking the class are not using their regular tools to perform the classroom exercises. For the classroom exercises they use laptops that may not even be theirs (only on loan). They may not have the same environment setup as their regular work. All this adds up to a lot of unfamiliar things. When the class is over, they go back to doing all the familiar things that they have been doing before the class.

Class programs and work programs are different. Engineers writing, for example, user interface code for a web application don't see how something that they learn about writing programs for numerical recipes can help them write better user interface code. This problem is similar to the next one.

The new knowledge is learned in a context that is different from the context of the everyday work. The engineers learning the new skills, for example planning and estimation, reported after the class that now they know how to apply these skills on these numerical type programs that were used in the class, but their regular work is different, and this same knowledge cannot be applied.

The technical solution of the applications developed in the classroom is different from the technical solution used in everyday work. Most applications even in this age of object-oriented programming using Java or C# are still monolithic applications with module dependencies that are tightly coupling together the system's parts. For engineers to see how they could use their new skills on their everyday problems, the new skills need to be learned on these applications. Once they learned the new skills, they see that writing monolithic applications is a problem.

Lack of tool support. This is an odd problem. Most large (and some small) organizations that have embraced the PSP have built their own tools. There are a few that are also

selling the PSP support tool. Because the market is still limited, the prices are high. Folks in small organizations still have to use the spreadsheet that the SEI folks created in the late nineties. This is one problem that the alternate introduction method doesn't address.

IV. An Alternate PSP Introduction Method

In 2001 working with two other instructors we started to prepare a new set of problems that would match more closely the problems that the majority of engineers taking the class were working on every day. The effort failed. We created only a couple of assignments. This pointed out that we could not create assignments fast enough to match the variety of problems that the engineering teams are working on.

The next experiment was to work with eager volunteers who took the class on their own time. Even these eager folks had difficulty seeing how they can use the knowledge gained in their everyday work. Without doing a proper statistical analysis of the data, I concluded that the results from this group are harder to dismiss, since these folks wanted to learn the new knowledge.

A. Customize the Learning to the Learner

In the book, *Flawless Consulting* [6], Peter Block picked a provocative header for one of the subchapters: Choosing Learning over Teaching. His view is that many consulting engagements make the consultant central to learning. This comes at a high cost: the learner never fully engages in the subject.

Therefore, the alternate method must be base on a great deal of one-on-one interaction between the coach and the learner. This doesn't have to be in person, email and telephone work well.

B. Build One New Habit, Learn One New Skill at a Time

The idea of building new habits one at a time is not new. We can find a reference to this idea in Benjamin Franklin's Autobiography written in the late 1700s: "My Intention being to acquire the *Habitude* of all these Virtues, I judg'd it would be well not to distract my Attention by attempting the whole at once, but to fix it on one of them at a time, and when I should be Master of that, then to proceed to another, and so on till I should have gone thro' the thirteen." [7]

Peter Drucker expresses a similar idea in *The Effective Executive* as he talks about a successful executive that achieved great results during his tenure: "He did this by single-minded concentration on one task at a time. This is the 'secret' of those people who 'do so many things' and apparently so many difficult things. They do only one at a time. As a result, they need much less time in the end than the rest of us." [8]

The need for a gradual introduction of these methods recurs in [1], [9]. As the authors of [9] talk about skipping maturity levels, they say: "Because each maturity level in CMM forms a necessary foundation on which to build the next level, trying to skip levels is almost always counterproductive." Given that the PSP and TSP are built on the same principles, we can assert that our focus should be on figuring out how to effectively learn these concepts.

The suggested alternative introduction method is similar the method recommended by the SEI, with few exceptions. The concepts are built up one on the other as they are taught in the PSP class. The difference is that instead of introducing and practicing these concepts in a classroom learning format, I suggest introducing these concepts in 60-90 minute sessions, one at a time, and distributed a few weeks apart. Leaving a few weeks between sessions leaves time for the engineers to practice the material on their regular work and not on assignments.

It is up to the coach to work with the learner to find suitable size components or parts of work that the learner can practice on to learn the new skills.

An outline of the proposed introduction method follows:

1. Identify tasks required to accomplish the project objectives and log time worked on each identified task (PSP 1.1)
2. Estimate task duration and schedule hours available for project work for each project week based on the time log from previous weeks (PSP 1.1)
3. Add peer review and unit test tasks to the personal task list
4. Conduct a weekly personal review of the personal data to identify lessons learned at the personal level during the previous week and propose lessons to be learned the following week (PSP 0.1)
5. Identify work phases and categorize tasks into phases (PSP 0)
6. Analyze each personal work phase for entry and exit criteria and input and output artifacts
7. Log defects found in your own work and determine the injection phase for each (PSP 0)
8. Build review checklists based on the defect log and replace most peer reviews with personal reviews (PSP 2)
9. Identify system structure and apply personal iteration to each software component addition or modification (PSP 3)
10. Create a quality model of the system based on the work phases and the defect injection and removal rates
11. Analyze each task for the knowledge it requires and decide on actions to improve the knowledge used to complete the task (PSP 2.1)
12. Establish a size measure for the types of artifacts (work products) that you create, analyze the changes in the artifact size (PSP 1)

C. Getting Started

Get started by moving forward. Decide to do something about the status quo. You need not be an executive to do this. As Peter Senge expresses it in *The Fifth Discipline* [10] it is sufficient to have three people who want change to get started. Find your partners.

Start with a *Lunch & Learn* on the first point from above. Leave ample time for Q&A. The material from point 1 above is basic project management so it should be easier to get it going. Once the concept is explained to engineers, work with them to practice these skills.

It is good to have an SEI-certified PSP Instructor on hand, because the instructor is expected to know the material inside-out, therefore this person can be a great resource. It can be a double-edge sword situation also, since the

instructor may not want to change things much, since changing things will jeopardize his SEI certification.

Assuming that you found two willing partners, you will need to get a copy of [1] and review [4], [5]. You need to learn what is in these publications because it is difficult to tweak something that you don't understand. You can skip some of the details, if you can rely on somebody who knows this material. The goal is not to reinvent the PSP, but rather to tweak the introduction and enable you to get it going in your organization. You will think about improving what you do, once you have the PSP implemented. The closed loop corrective action is built into the PSP framework.

Understand the objectives of each of the steps outlined above. Follow the advice from [11] to outline your work in the format: *outcome desired, action required*. Practice this also for a few weeks until it becomes a habit.

Some activities may seem tedious or unnecessary. The message from [12] that reads as *"A successful manager of time is willing to do that which the unsuccessful manager of time is not willing to do"* can be fitted for your situation as well: "A successful manager of personal process is willing to do that which an unsuccessful manager of personal process is not willing to do."

CONCLUSION

The PSP provides a framework for software engineers to do outstanding work. Learning the practices embodied in the framework is difficult, because many of these practices require new skills. New skills are difficult to learn. It is important that we learn them one at a time, with ample practice time, until they become a habit. Individuals who understand their performance can form high-performance teams and the TSP enables this.

ACKNOWLEDGMENT

Since 1992 the works of Watts Humphrey have deepened my understanding of the software development process. I owe him my deepest gratitude.

REFERENCES

[1] Humphrey, Watts, S. *A Discipline for Software Engineering, The Complete PSP Book*. Boston, MA: Addison-Wesley, 1995.

[2] Humphrey, Watts, S. *Introduction to the Team Software Process (TSPi)*. Reading, MA: Addison-Wesley, 2000.

[3] TSPi Support Tool that accompanies the TSPi book http://www.awprofessional.com/content/images/020147719X/support/020147719X_TSPi.zip (URL current as of 15 May 2005).

[4] Humphrey, Watts, S. *Pathways to Process Maturity*. http://www.sei.cmu.edu/news-at-sei/features/1999/jun/Background.jun99.pdf (URL current as of 15 May 2005) 1999.

[5] SEI. *Compilation Data for Projects Using TSP/PSP*. http://www.sei.cmu.edu/tsp/results/compilation.html, (URL current as of 15 May 2005).

[6] Block, Peter. *Flawless Consulting, A Guide to Getting Your Expertise Used, Second Edition.* San Francisco, CA: Jossey-Bass/Pfeiffer, 2000.

[7] Franklin, Benjamin. *The Autobiography and Other Writings*. New York, NY: Penguin Books, 1986.

[8] Drucker, Peter. *The Effective Executive*. New York, NY: HarperBusiness Essentials, 1967, 1985, 1996, 2002.

[9] Paulk, Mark C., et. al. *The Capability Maturity Model: Guidelines for Improving the Software Process*. Reading, MA: Addison-Wesley, 1994.

[10] Senge, Peter, M. *The Fifth Discipline, The Art and Practice of a Learning Organization*. New York, NY: Currency Doubleday, 1990.

[11] Allen, David. *Getting Things Done, The Art of Stress-Free Productivity*. New York, NY: Penguin Books, 2001.

[12] *Franklin Day Planner System Guidebook*. Salt Lake City, UT: Franklin International Institute, 1989.

Revised: 15 May 2005