Incorporating PSP into a Traditional Software Engineering Course: An Experience Report

Jonathan I. Maletic
Div of Computer Science
Dept of Math Sciences
The University of Memphis
Memphis TN 38152-3240
jmaletic@memphis.edu

Anita Howald
Woodson-Tenent
Laboratories
Memphis, TN 38104

ahowald@wtlabs.com

Andrian Marcus
Div of Computer Science
Dept of Math Sciences
The University of Memphis
Memphis TN 38152-3240
amarcus@memphis.edu

Abstract

This paper presents an approach to incorporate PSP into a traditional Software Engineering course that is typically contained within a Computer Science curriculum. Advantages and disadvantages of similar approaches are discussed. The approach has been implemented twice in an undergraduate course at the University of Memphis. This successful experience is described and gives support that the proposed approach is beneficial to both students and educators.

1. Introduction

The Personal Software Process (PSP) [11] is becoming a popular topic in undergraduate computer science and software engineering curriculums. It is being integrated into the CS1 course [5, 7, 10, 17, 25] and taught as a stand-alone course [3, 15, 17]. PSP forms a basis for evaluating and improving a person's software development abilities. A number of studies have been published describing the results of running such courses. Many of these reports describe the pros and cons of when this material should be introduced into a program. Some of this work suggests that the rigors of this process make its introduction at the CS1 and CS2 level too early for most students to benefit [7, 17, 25]. As is common in many computer science courses CS1 and CS2 are overloaded with topics and material. Covering one more topic in such a course is hard on the student and the instructor.

Another approach is to cover PSP in an upper division course. To fully cover the Personal Software Process requires a large commitment of time and effort. The lectures designed to accompany Humphrey's text [11] consist of 15 lectures each of which is 90 to 120 minutes. This equates to approximately an entire academic semester. Given that most Computer Science (CS) programs do not have more then one or two undergraduate Software Engineering (SE) courses, this format is often not a good option for a department with limited means. Also, Humphrey's text and lectures are primarily aimed at the professional rather then the undergraduate student. This tends to make a hard fit in an undergraduate curriculum. With the limits imposed on degree requirements by the respective college and University, there is often little extra time in a four-year curriculum for the student to take an extra course that deals solely with PSP.

Both of these approaches have some inherent problems but teaching some type of personal process improvement is generally agreed (at least by the authors) to be a very worthwhile concept. Lecturing on PSP and having students actually take measurements is a very powerful mechanism to teach estimation and the importance and usefulness of metrics, among other things. Therefore, we feel that incorporating it into the curriculum in some manner is important and worth the investment.

Our approach is to integrate the concepts of PSP into a traditional undergraduate software engineering course. By traditional we not only mean along the lines of textbooks such as Pressman [22], Sommerville [24], or Peleeger [21] but also a course that focuses on object oriented design using a text such as Bruegge and Dutoit [2], or Horstmann [9]. The later model is what is specifically dealt with in this report. Portions of the PSP approach are discussed in lectures and students use parts of PSP for projects and homework. Based on the experiences of teaching PSP in this manner, the approach seems to be quite successful in instilling the student with the basic concepts and importance of such a process.

Before we go into detail about how the course is designed and the results of conducting the course, some background material will be presented. The next section gives a short overview of PSP, if the reader is familiar with PSP they can readily skip section 2. Section 3 is a survey of the related studies and experience reports of teaching PSP. It also gives some motivation for our approach. Section 4 describes the course and section 5 presents some of our experiences. PSP was also integrated into a graduate level software engineering course and some preliminary results, of that experience, are presented. General conclusions and recommendations for teaching PSP are lastly given.

2. The Personal Software Process

PSP (Personal Software Process) was developed at the Software Engineering Institute (SEI) at Carnegie Mellon University by Watts Humphrey to give engineers the process understanding they need to improve organizational performance by making better estimations, improving product quality, measuring personal process improvement, and determining the impact of the process [11, 12]. PSP is a grass roots version of CMM (Capability Maturity Model) to be used by individual software engineers. Several companies who have applied the PSP principles showed higher quality in the software products, a reduction in development time and an improvement in the communication between higher management and software teams [4, 16]. The principles of PSP are to help software engineers to:

- Know their own performance: to measure their work, to recognize what works best, and to learn how to repeat it and improve upon it
- Understand variation: what is repeatable and what they can learn from the occasional extremes
- Incorporate these lessons in a growing body of documented personal practices [23].

Software developers gain experience in empirical work that could lead to individual improvement. “In turn this personal improvement will translate into team performance improvement, and finally into organizational improvement. Hence, PSP is only one of the many steps towards higher organizational performance” [23].

Learning PSP, as described in [11], requires students to work on ten small programming projects at a rate of one per week. The students keep precise measurements such as logical lines of code (new/reused/modified), defects found, phase of injection and removal of defects, time spent on fixing each defect, time spent on each activity and phase of the project, and estimated and actual values of project size and time. The training program

introduces one new concept every week to improve their software development process [11, 12, 23]:

- Measuring and Tracking the project
- Software Project Planning
- Statistical methods for Estimating Size and Time
- Schedule and Resource Planning
- Code Reviews & Defect Prevention Strategies
- Structured Design Methods
- Cyclic Personal Process

PSP is based on the concept that each programmer is different and the process should work for that individual. PSP teaches the principles of process definition and measurement and shows programmers how to tailor a PSP process to best fit their unique needs and preferences. PSP is intended to help students practice software methods that are most effective for them. Programmers would tend to develop high quality large programs as long as they are able to develop high quality small programs.

3. Other Experiences Using PSP

Several published experience reports exist that deal with the inclusion of PSP in the CS/SE curriculum. One approach is to incorporate PSP as a distinct course. A second approach is to include PSP into the existing SE course(s) in the CS curriculum. Here again, there are two different trends: one to include PSP in early programming courses (CS1 or CS2), and the second one (reported less often) is to include PSP in advanced undergraduate SE courses and/or graduate SE courses. Our choice falls into the last category.

In the undergraduate curriculum, the Software Engineering Process is rarely taught as a whole in the first few CS courses. The result of which is that students are never forced to think through the actual process, if they even know what the actual process is, and they tend to develop very poor personal programming practices. This problem is most acute in substantial group-based practical work [10]. The primary aim of introducing the PSP is to expose students to a proven process and reliable tools they can use to develop their own process through learning from their own mistakes. A secondary aim is to provide low-cost, easy to use tools for recording PSP statistics that will give teaching staff a convenient way of monitoring student performance and problem areas [1, 7, 18, 25].

In most cases where PSP was introduced in early programming courses, the conclusions were that first and second-year students are not ready to appreciate the benefits of PSP [3, 5, 7, 15, 17, 25]. Most students complained about the amount of time they needed to learn and apply PSP versus the amount of time they spent on learning about software development. Many of these students lacked the notion of the software development process and any programming experience, thus were unable to see how PSP will help them improve their skills and the quality of their work. In addition, a serious concern exists about the quality of the PSP data generated by these students. This data seems to decrease in quality with the increase of the overhead work needed to handle the PSP manual forms [15]. In fact, this issue is of importance not only in these cases but also in all experiences in using PSP. One partial solution to this problem is the automation of the PSP logs.

One common feature of all the approaches is the simplification of the PSP model in order to fit it into an existing CS/SE course such that no more than 2-4 classes are spent on learning it. A number of forms (between 2 and 5) are used in each case by the students to monitor their time, effort, number of defects in the code, and the size of the code. Previous experiences were concerned about the amount of work needed for the PSP manual forms and

the issue of automated tools for completing the forms is discussed. Preference is given to tools that are (or could be) integrated into the development environment [1, 7, 15, 18]. All of these researchers agree that PSP concepts need to be integrated, in some manner, into the CS/SE curriculum. The benefits outweigh the inherent difficulties.

All the experiences had one or more of the following goals:

- Help the students to develop a better understanding of the phases of software development process and to properly follow the steps when developing a system.
- Give students first-hand knowledge on product and process metrics. Use this information to better estimate the necessary effort in building a software system.
- Continuous quality improvement.

Different approaches considered different goals of primary importance. Hilburn et al. [6-8, 25] consider the issue of software quality as most important and their experiences and methods focus on continuous quality improvement. Others focused more on using PSP to help the students better learn the software development process and estimations.

In our approach, we chose to include a simplified version of PSP into an undergraduate course in Software Engineering [19]. The next section describes this course and how parts of PSP are inserted into the traditional topics.

4. Traditional Software Engineering Course + PSP

Anyone who has taught an undergraduate software engineering course encounters the information overload problem. That is, what subset of the wealth of software engineering topics do you cover? One semester is obviously not enough time to cover all the important issues. Vertical integration of software engineering topics into the curriculum is the only practical way to properly educate students in these topics. Until more SE degree programs are created, we are left with the current model of one or two software engineering courses in a CS program.

The undergraduate software engineering course that is being run at the University of Memphis consists of an overview of life cycle models then it focuses on object-oriented design and advanced object-oriented programming methods. Issues concerning testing software systems are also covered. A set of design/programming projects are assigned to support the concepts covered in lecture. The course deviates from the traditional model by introducing PSP to cover process models, process improvement, estimation, and metrics. The resulting course covers the advanced topics of OO design and programming within the framework of process improvement.

We have experimented with using both of the textbooks on PSP. In using the textbook, *Introduction to the Personal Software Process* [13], we found that it did not fit perfectly into an upper division course. The other textbook, *A Discipline for Software Engineering* [11], is often over the heads of inexperienced undergraduates. While, either of these books are a good additional reference for the course, we found ourselves developing lectures on the material that better fit the level of the student and the restricted amount of time allocated to discussion of PSP.

Table 1 gives an overview of the topics covered in the course. The traditional topics listed in this table are not of great issue here. In fact, many departments/instructors may find that these particular traditional topics do not reflect their needs or curriculums. What is of interest here is that PSP topics are integrated into the course and support the other material covered therein.

Week	Lecture Topic
1	Overview of Software Engineering and Life Cycle Models
2	Object Oriented Design, UML diagrams
3	**Software Process Improvement, CMM, and PSP Overview**
4	**PSP Baseline, Measuring Software Size**
5	**Object Oriented Design, UML diagrams**
6	Object Oriented Analysis and Design, UML diagrams
7	Programming Inheritance and Multiple Inheritance
8	**Planning and Estimation**
9	**Estimation and Metrics**
10	Libraries, frameworks, and reuse
11	Exception Handling
12	Walkthroughs, Inspections, and Reviews
13	Software Testing
14	Testing OO Systems

Table 1. An example of integrating PSP with Traditional SE topics. The PSP related topics are in bold.

Approximately, five weeks of the fourteen-week term are devoted to PSP topics. For the most part these lectures are generated from material in [11]. Around week three and four an overview of the personal software process is given, the baseline process is described, some planning is covered, and how software size is measured and used for process improvement is discussed. This gives the students a platform to take measurements of their own work and understand the goals of process improvement.

Then just after midterm, issues of planning, estimation methods, and metrics are covered. The students used some of the PSP forms and took measures. With this experience, they tend to have a good practical handle of why and how to produce size estimations. Then in the twelfth week, to kick off a series of lectures on software testing, methods for design and code reviews are described.

For projects and homework, students used a subset of the forms presented in [11]. The students did time and defect recording for all design and programming assignments. A project plan summary is also handed in for each assignment. Initially, a PSP0 summary is used, and then as the student gathers more experience a later version can be utilized. We used a summary form similar to PSP1. Special attention was given to the quality of the PSP data. Students were aware that their grade depended on the accuracy of their data. Any inconsistency within the data was reason for grade penalty. This approach made some students dislike the whole process but improved the overall quality of the PSP data from one assignment to the next.

5. Results and Experiences

The undergraduate course described above was conducted twice by the author [19] and a graduate software engineering course that incorporates PSP in a similar manner is currently (Fall 2000) being run for a second time [20]. We focus on the results obtained in the undergraduate course as it best motivates the need for such vertical integration.

Students in the class had varying programming background and experience (see table 2). A small number of the students had one or two years of industry experience. This largely influenced the way they perceived and applied PSP in the class. As a general trend, the students with more experience were the ones who understood first the benefits of applying PSP. The less experienced students had trouble understanding the long-term benefits of

using PSP and considered the effort unnecessary. The later category of students had more trouble learning and understanding the software development process in general and the object-oriented methodology. The more experienced students were interested in perfecting their skills and PSP seemed to help achieve this goal.

Student's programming experience	% of students
1 - 2 year	20%
3 – 4 years academic	45%
5 – 10 years academic + industry	20%
10+ years + industry	15%

Table 2: Typical student's programming experience in the course. These are approximate averages over two terms.

The students answered a number of questionnaires (both formal and informal) both during and at the end of the term, in order to evaluate PSP. The results were combined (see table 3) to express the general trends among students. Again, the more experienced students evaluate the whole process differently than the less experienced ones.

The results led to several conclusions. Most of the students found that learning the parts of PSP was easy, applying it a bit more difficult due to the time requirement to fill out the manual logs, and that PSP is useful overall. Most students failed to see upfront the benefits of PSP but reconsidered their position at the end of the course. The vast majority of students agreed that using the project plan summary helped them better understand the software development process, and follow its steps more rigorously. Most of all, the students agreed that good record keeping considerably helped them improve their estimation skills for effort, time, size, and quality of the software product.

The use of PSP did …	Majority response	
	More experienced students (4+ years)	Less experienced students
…help me better understand the SD process.	Agree	Agree
…help me better follow SD process steps.	Agree	Agree
…improve the quality of my software.	Neutral/Disagree	Disagree
…better measure the quality of my software.	Agree	Agree
…help me write better software documents.	Neutral/Disagree	Disagree
…improve the estimation of my efforts.	Agree	Agree
…improve my time management.	Agree	Agree
…increased the amount of work too much.	Neutral/Disagree	Agree
…not help me in any way whatsoever.	Disagree	Disagree
I learned PSP easily.	Agree	Agree
I did not see benefits of PSP upfront.	Neutral/Agree	Agree
I consider now PSP overall useful.	Agree	Neutral/Agree
My time data is accurate.	Agree	Agree/Neutral
My defect data is accurate.	Neutral	Neutral/Disagree
My size data is accurate.	Agree	Agree
I am going to use PSP in future.	Neutral/Agree	Neutral/Disagree

Table 3: Student survey

By making a project plan, students are able to determine how to do the work and to determine how much time their work would require. Since many students usually start programming before actually giving an in-depth thought for the solution, students using PSP

were required to design their programs and validate the correctness of their design. This process gives the students the ability to think about their design before actually implementing their code.

Despite the stated goal of the course, software quality improvement, most of the students failed to acknowledge that PSP directly supports software quality improvement. This fact is explained by the lack of historical data on previous projects for the students. In addition, students had the tendency to record defects less accurately. It is obvious that PSP needs to be used more than once by students in order to get into their habits and to provide maximum benefits. Overall, after conclusion of the course, students have a better concept of planning, measurements, and data analysis. The students can give better estimations for future projects (those similar to the project within the semester that PSP was taught). Obviously, estimation is an important skill for any software engineer working in the industry.

A side effect of PSP in the classroom is additional information for evaluating student progress. It is often difficult to evaluate class assignments and projects and the timesheets and defect logs from PSP represent an additional dimension that can be used in student evaluation. The PSP logs maintained by the students can also reflected the actual progress of students during the term (and during an individual assignment). While these forms are prone to low quality and falsification, they do prove useful in some cases. Also, falsification often leads to inconsistencies in the recorded data, which are fairly easy to spot.

6. Preliminary Results of the Graduate Course

The experiences from the undergraduate course led to a graduate course that also incorporates PSP concepts. Currently, we are also applying a similar approach in a graduate software engineering course. Unfortunately, few of the students in the graduate course took part in the undergraduate course where PSP was already incorporated. If they had taken the undergraduate course than their baseline process would be more mature and the course could focus on team oriented issues. However, as it was, getting individuals to understand the issues of PSP had to be the main focus.

Students in the graduate course work in groups of 3-5 persons. The backgrounds of the team members are purposely varied for each group. That is, each team has only one or two industry-experience software developers with a mixture of less-experienced graduate students. In addition, the instructor attempts to split the individuals who have worked in previous projects from other courses into different teams. The purpose is to study the complexity and difficulties within team structure. Precise records are kept for documenting each team member's contribution to the project.

The project was divided into three phases: Requirements and Analysis, Design, and Implementation. Personal and team records are kept for each individual and team's progress and duties. All faults are recorded and the amount of effort for each phase is reported. The first two phases have three parts, an initial version of the document, an independent review by another team, and a final version of the document. Records are kept for each part. The implementation consists of a beta version, final version, and independent acceptance test by another team. Special attention is given to the difference between students that already were exposed to PSP and those who are encountering it for the first time.

Generally, when a group of students starts a project, they get little or no guidance on how to proceed. If they are lucky, one or two of the experienced team members will have worked on well run teams and have some ideas on how to proceed. The Team Software Process (TSP) [14] provides team projects with explicit guidance on how to accomplish their objectives. Some elements of TSP were introduced to the students in order to help them

better manage their teams. The teams had to produce the following elements that are part of TSP:

- Written team goals
- Defined team roles
- A process development plan
- An overall development plan and schedule
- Detailed next-phase plans for each team member
- A project status report

The course was offered in this format in the Fall '99 term and was quite successful with respect to student feedback and evaluation. Students were, for the most part, very positive about the concepts of PSP and how it helped them to improve their ability to make better estimates. It is foreseen that this term should equally be as successful.

7. Conclusions and Recommendations

Our experience tends to reinforce the idea that learning only a subset of PSP is enough for the student to understand the usefulness of such habits for process improvement and software development. The more experienced student appears to have more affinity towards the principles embodied by PSP. Also, based on other studies, introduction of PSP concepts at the CS1/CS2 level seems to be premature. Students do not have the cognitive discourse necessary for programming and problem solving in these introductory courses and the addition of PSP concepts may just be too much material in too little time. In fact, the goal of these courses is to give the student the necessary discourse.

It seems clear that a vertical integration of the principles of process improvement (e.g., PSP, TSP, CMM, etc.) into a larger part of the curriculum is the soundest approach. What needs to be further investigated is the when, where, and how much. When should this material be first introduced in a SE program and how will the concepts be spread over which courses? Starting simple and moving in an incremental fashion may be one option. For example, just having students take time/effort recordings in CS1/CS2.

Designing a degree program to include these topics is the best-case scenario but the fact of the matter is that there are few Software Engineering degree programs at the undergraduate or graduate level. Also, many programs (undergraduate and graduate) have a number of good software engineering courses in place. The more traditional approaches to teaching software engineering cover a large amount of important information. The reason most often given for not teaching PSP is that there is so many other issues that need to be covered that there is just not enough time to talk about PSP in detail. In the best case, a student would have the time and desire to take all of the courses being offered on this topic and receive a more complete academic background on this subject. However, if a student is only going to take one course, or only one course is offered on software engineering can they be given a taste of PSP with the traditional topics?

Lastly, it is our feeling that a textbook on OO design/programming, which uses a process improvement methodology such as PSP or some variation, would be an interesting and useful mechanism to teach both topics.

8. References

[1] Brown, A. L., Oudshoorn, M. J., and Maciunas, K. J., "The Personal Software Process in Undergraduate Software Engineering Education", in Proceedings of International Symposium on Software Engineering Education in Universities, Rovaniemi, Finland, March 1997, pp. 52-59.

[2] Bruegge, B. and Dutoit, A., *Object-Oriented Software Engineering Conquering Complex and Changing Systems*, Prentice Hall, 2000.
[3] Disney, A. M. and Johnson, P. M., "Investigating Data Quality Problems in the PSP", in Proceedings of ACM SIGSOFT 6th International Symposium on Foundations of Software Engineering, Orlando, FL USA, November 1-5 1998, pp. 143 - 152.
[4] Ferguson, P., Humphrey, W. S., Khagenoori, S., Macke, S., and Matvya, A., "Results of Applying the Personal Software Process", *IEEE Computer*, vol. 30, no. 5, May 1997, pp. 24-31.
[5] Grove, R., "Using the Personal Software Process to Motivate Programming Practices", in Proceedings of ITICSE '98, Dublin, Ireland, 1998, pp. 98-101.
[6] Hilburn, T. B., Hirmanpour, I., and Kornecki, A., "The Integration of Software Engineering into a Computer Science Curriculum", *Lecture Notes in Computer Science* no. 895, 1995, pp. 87-98.
[7] Hilburn, T. B. and Towhidnejad, M., "Doing quality work: the role of software process definition in the computer science curriculum", in Proceedings of Twenty-eighth SIGCSE Technical Symposium on Computer Science Education, San Jose, CA, Feb 27-Mar 1 1997, pp. 277-281.
[8] Hilburn, T. B. and Towhidnejad, M., "Software Quality: A Curriculum Postscript?", *SIGCSE Bulletin* March 2000.
[9] Horstmann, C. S., *Mastering Object-Oriented Design in C++*, Wiley, 1995.
[10] Hou, L. and Tomayko, J., "Applying the Personal Software Process in CS1: an Experiment", in Proceedings of 29th SIGCSE Technical Symposium on Computer Science Education, Atlanta, Feb 26-Mar 1 1998, pp. 322-325.
[11] Humphrey, W. S., *A Discipline for Software Engineering*, Addison Wesley, 1995.
[12] Humphrey, W. S., "Using a Defined and Measured Personal Software Process", *IEEE Software*, vol. 13, no. 3, May 1996, pp. 77-88.
[13] Humphrey, W. S., *Introduction to the Personal Software Process*, Addison Wesley, 1997.
[14] Humphrey, W. S., Lovelace, M., and Hoppes, R., *Introduction to the Team Software Process*, Addison-Wesley, 1999.
[15] Johnson, P. M. and Disney, A. M., "The Personal Software Process: A Cautionary Case Study", *IEEE Software*, vol. 15, no. 6, November/December 1998, pp. 85-88.
[16] Kelly, D. P. and Culleton, B., "Process Improvement for Small Organizations", *IEEE Computer*, vol. 32, no. 10, October 1999, pp. 41-47.
[17] Lisack, K. S., "The Personal Software Process in the Classroom: Student Reactions (an Experience Report)", in Proceedings of Thirteenth Conference on Software Engineering Education & Training, 2000.
[18] Maciunas, K. J., Oudshoorn, M. J., and Brown, A. L., "Process Improvement of Software Engineering Education", in Proceedings of First Australasian Conference on Computer Science Education, Sydney, July 1996, pp. 66-73.
[19] Maletic, J., "COMP4/6081 Introduction to Software Development Spring '99", Web Page, Date Accessed: September, http://www.cs.memphis.edu/~maleticj/COMP4081/, 1999.
[20] Maletic, J. I., "COMP7083 Software Development Methods Web Page", Web Page, Date Accessed: May, 2000, http://www.msci.memphis.edu/~maleticj/COMP7083/, 2000.
[21] Pfleeger, S. L., *Software Engineering Theory and Practice*, Prentice Hall, 1998.
[22] Pressman, R., *Software Engineering A Practitioner's Approach*, McGraw Hill, 2001.
[23] Sherdil, K. and Madhavji, N. H., "Human-Oriented Improvement in the Software Process", in Proceedings of The 5th European Workshop on Software Process Technology, Nancy, France, October 1996, pp. 276-282.
[24] Sommerville, I., *Software Engineering*, Addison-Wesley, 1996.
[25] Towhidnejad, M. and Hilburn, T. B., "Integrating the Personal Software Process (PSP) across the Undergraduate Curriculum", in Proceedings of ASEE/IEEE Frontiers in Education Conference, November 1997.