Adjusting the Instruction of the Personal Software Process to Improve Student Participation

Laurie A. Williams
Department of Computer Science
University of Utah
Salt Lake City, UT 84112

"The PSP made me make a conscious effort to try and correct any errors made as soon as possible, and I feel that I am a better programmer for it."

-- Anonymous PSP freshman student
University of Utah

Abstract *- No customer is fully satisfied unless they receive a product that does what they want, and they receive it when they want it, defect-free and at an agreed upon price. To address all four of these requirements, Watts Humphrey, at the Software Engineering Institute(SEI), developed the Personal Software Process(PSP), which applies proven quality principles to the work of individual software engineers.[1] PSP was initially taught to practicing software engineers in industry and to graduate students. Earlier this year, Humphrey published a new text[2], suitable for the beginning software engineering student. This text outlines a process for managing and producing high quality software, reducing the mathematical and statistical rigor of the original PSP, but maintaining a solid base of disciplined practices and an overriding quality philosophy. This paper describes how the Personal Software Process has been incorporated into the undergraduate Computer Science courses at the University of Utah. It highlights how instruction has been adapted to improve student participation, retention, and utilization of the material.*

Introduction

Attention must be focused on educating a new generation of software engineers with processes and principles that will address the Software Crisis and restore integrity to the software industry. Work done at SEI on the Capability Maturity Model and the Personal Software Process(PSP) has shown that the "quality of software work is governed by the practices of the software engineers. If their practices are poor, their products will be poor. . . "[3] Without a focus on all of the disciplined aspects of producing a high quality product, students establish poor work habits which carry through to their professional careers.

The University of Utah has sought to educate the "whole software engineer" by incorporating software engineering principles throughout the curriculum. PSP concepts have been incorporated into freshman, sophomore and senior level Computer Science classes. Humphrey's Introduction to the Personal Software Process[2], designed for use in the undergraduate curriculum, instructs students on planning and managing their time, removing defects early in the development process and instills a quality philosophy. However, human nature has caused the students to initially reject some of the discipline and rigor required by PSP. This paper addresses some adaptations to our initial instruction of PSP, which has improved the student's acceptance and active participation in learning these very important concepts.

The Introduction to the Personal Software Process

Humphrey introduces skills and practices to beginning software engineers in his latest book, The Introduction to the Personal Software Process[2]. Students are typically taught how to translate "customer requirements" (a.k.a. their assignments) into a given programming language. PSP rounds out their perspective by highlighting the need for the software they produce to be delivered on-time, and defect-free in order for it to be a quality product. Without this added perspective, students traditionally focus on successfully translating their assignment into programming code that compiles and passes basic test cases (often provided by their professor).

Specifically, the PSP concepts that were taught to our undergraduates were time management/planning and defect removal. The students recorded the time they spent on the various development phases of their programming assignments. Using provided tools (discussed later), these

0-7803-4086-8 ©1997 IEEE

numbers were combined with the number of lines of code they wrote to produce a productivity factor, "Lines of Code/Hour", which was then used for future planning and estimation. Additionally, they performed code reviews and recorded each of the defects they removed from their program, from code review phase through testing. This data was compiled via a tool into summary reports, which highlights problem areas to the student. Once the students begin to realize their own personal weaknesses by studying these reports, they can use this information to prevent injecting these types of defects in the future. Humphrey's book also explains that defect removal is an economic issue where the cost of defect removal increases about tenfold with each later phase of the development process. There wasn't a student in the class who forgot that 250 engineers spent a full year to remove 30,000 defects from Windows NT – an average of 16 hours per defect![2]

Generally, the students were receptive to learning these PSP concepts. In an anonymous survey, 81% of the students said they felt that the material they had learned about time management/planning would be beneficial to their academic and professional careers. Likewise, 52% said they felt the defect removal material was beneficial. The remainder of this paper will outline some specific techniques on how these results were achieved and how they could be improved.

Data Entry

The Introduction to the Personal Software Process[2] comes with Microsoft Excel spreadsheets to track and manipulate the PSP data. However, our freshman use UNIX workstations. Therefore, spreadsheets were developed using a UNIX package, xspread, which is not as user-friendly or flexible as Excel. Many of the students had never used UNIX before, found these procedures difficult, and profoundly disliked working with the xspread spreadsheets. Many equated PSP with xspread and grew to dislike both. This lead us to an important lesson: for added acceptance and accuracy of these disciplined software engineering steps, it is imperative to choose a tool that is essentially invisible, allowing students to focus on the programming tasks.

Therefore, a web-based tool was developed for use in the Winter 1997 Quarter. Complaints about difficulty in following data entry procedures virtually disappeared, allowing the students to focus on the intended lessons. In the Fall 1997 Quarter, 73% of students anonymously reported that their time recording log data accurately reflected the actual time they spent on their program. This number jumped to 88% the following quarter with the new tool.

The new tool, however, did not improve the students' accuracy in recording their defects. PSP procedures require the students to record each defect they remove from their program after code review. In the Winter 1997 Quarter, students recorded an average of five defects/program. Clearly, the students were not recording all their defects. Many of their defects are purely syntactic at this point in their development. It took only a few seconds to correct a missing semicolon or to match a bracket. It takes equally long, if not longer, to record each defect. Therefore, in Spring 1997, a "Quantity" field was added to the defect recording screen. Then, students could more easily record, for example, they were missing three semicolons without making three separate entries. The average number of defects entered per student jumped to 11 (although this is still quite low). In the PSP process taught to professionals and graduate students, recording each defect individually is more appropriate because the defects tend to be more complex and time consuming.

Time Management vs. Defect Removal

In the Fall 1996 Quarter, the CS1 students were taught the first half of Humphrey's book which deals with time management/planning. However, particularly early in the quarter the students were overwhelmed with learning to compile programs and to use the UNIX workstations. Time management was not particularly relevant to their other educational tasks, and in their view, hindered their ability to focus on these tasks. As a result, in the Winter 1997 Quarter the CS1 class began their PSP instruction with the second half of the book which focuses on defect removal. They learned to perform code reviews and to record and classify their defects. The students found this material much more relevant. These students then learned time management in CS2 in the following quarter. This switch is probably only necessary if PSP is introduced to students in their very first programming class.

Incremental Development

In A Discipline for Software Engineering [1], Humphrey outlines PSP3 which deals with incremental development. Essentially, the strategy is to subdivide a program into smaller pieces and apply PSP processes, including design, code, code review, compile and test, to each of these smaller programs. Each increment is built on top of previous increments which have been fully developed and tested and are of high quality.

Humphrey's undergraduate PSP book, however, does not cover PSP3. It is beneficial for all students, including freshman, to learn the correct way to carry out incremental development. I have observed that some students practice

incremental development incompletely (i.e. they don't perform testing on each module after it is successfully compiled). Others try to write the entire program at one time, get confused by too many syntax/logic errors, and end up handing in a program that won't even compile.

Therefore, beginning in Spring 1997, the students were instructed on PSP3 and the proper method of doing incremental development. This also allowed them to see that PSP does apply to either development process. (Note: Students enter the time they spend in each phase on the time recording log. Incremental development increases the time to record this data because the students cycle through the phases quite rapidly. This data entry problem must still be addressed.)

Code Review

To reinforce the value of performing code reviews before compile, one lab period was dedicated to code reviews. The students were given some broken code. They performed a code review, fixed the defects they identified, and continued fixing until they got the program to compile. The program consisted of header, implementation and test files. Two defects, in particular, made an impression on the students. The first, a semicolon missing at the end of a class declaration in the header file, gave an error message pointing to the test file. Many students never fixed that defect. Another was a classical = instead of a == in a conditional expression. This particular segment of the code was purposely not tested by the test program. Many did not correct this defect since it was syntactically correct and not identified in testing. Both of these defects should have been easily found by a proper code review.

Other Results

When tested on PSP concepts, the students were easily able to display their knowledge of the material. Hopefully, this is evidence that they are successfully being indoctrinated with a quality philosophy and disciplined principles. However, results generally do not show that they are "walking what they talk". Productivity was flat throughout the quarter in the CS2 class despite the instruction in earlier defect removal. Additionally, there was no marked improvement in their estimating variance. It must be considered, though, that the students were learning completely new programming concepts with each program, so the associated learning curve might mask improvements.

On several occasions, the students were shown average PSP data for the entire class (i.e. average variance from estimate, productivity, etc.). They seemed to be very interested in being shown this type of data, which also might tend to cause them to be more accurate in their own recordings. I would highly recommend this practice to maintain the students' interest.

Summary

The Personal Software Process(PSP) is a disciplined framework for producing high quality software. Undergraduate students at the University of Utah have begun to learn PSP, which instructs the students on time management/planning and defect removal. PSP helps round out the students' perspective that all the factors involved in producing a quality product are essential. Student acceptance of the PSP principles is going well. The paper outlines how instruction has been modified to improve student acceptance, participation, and retention of PSP concepts. Two concerns remain. First, since many students use an incremental development process, our time recording process needs to be adapted for accurate recording of short cycles through the development phases. Also, additional focus must be placed to encourage greater utilization of the material in the students' actual program development practices.

Resources developed for the instruction of PSP (survey, lecture notes, tool) at the University of Utah can be accessed via the World Wide Web at http://www.cs.utah.edu/~lwilliam.

References:

[1] Humphrey, W. A Discipline for Software Engineering, Addison-Wesley, 1995.
[2] Humphrey, W. Introduction to the Personal Software Process, Addison Wesley, 1997.
[3] Humphrey. W. Making Software Manageable, Crosstalk: The Journal of Defense Software Engineering, December 1996, pp 3-6.