Teaching and Using PSP in a Software Engineering course: An Experience Report

K Venkatasubramanian, S Bastin Tony Roy, Muralikanth V Dasari

Computer Science and Information Systems group
Birla Institute of Technology and Science
Pilani, Rajasthan, India - 333031

{kvenkat,tony,dasari}@bits-pilani.ac.in

Abstract

In this paper, we describe our experiences with teaching software process improvement using some elements of the PSP as part of a traditional software engineering course. The goals were to help students develop good software development habits early, and to encourage them to see software development as a systematic discipline rather than a trial-and-error activity. We find that PSP is a viable and useful approach and has quantifiable, positive impact. Problems in teaching PSP are in keeping students motivated and keeping them focused on general ideas instead of details. Problems in using a personal software process are maintaining enough self-discipline and finding proper tool support.

Keywords: software process improvement, process measurement, quality metrics

1 Introduction

The Personal Software Process (PSP) framework is an approach suggested by Watts Humphrey in 1995[1]. It describes a methodology that leads an individual software engineer towards disciplined, well-defined work with continuous self-improvement. The PSP ideas are independent of programming language, application domain, and team organization. PSP shows engineers how to manage the quality of their products and how to make commitments they can meet. It also provides them with the data to justify their plans. PSP has been shown to substantially improve the estimating and planning ability of engineers while significantly reducing the defects in their products.

1.1 Context of the PSP Exercises

The PSP was offered as part of a traditional one-semester software engineering course for a class of 36 graduate students at BITS, Pilani. This course is the first exposure to software engineering for most students. We have introduced the PSP exercises in this course, for teaching software process improvement.

This paper appeared at the Software Engineering Education and Training Annual Conference 2001 *(SEETAC2001)*, Chennai, India.

The students had good programming experience in languages like C, C++ and Java. However, they did not have any exposure to a defined and measured software process. The focus of the current work is on gaining experience with teaching and using the PSP where the students were required to work through the PSP exercises on a defined schedule and in a structured course environment.

During the semester, over a period of twelve weeks, about ten lectures were given to explain the methods and mathematical models to be used in doing the PSP exercises. Students recorded data during the development of these programs, and submitted their reports along with their programs. To provide an incentive for doing the PSP, the exercises constituted 25% of the total weightage of the course.

In addition to the PSP, the students were also required to complete a term project covering various aspects of a typical software development life cycle. This term project was a team effort.

In the following sections, we describe our experiences with our first attempt at teaching and using PSP in a traditional software engineering course for graduate students.

2 Quantitative results

This section presents some of the results obtained in our first experience in teaching PSP. These results confirm those published by others and add information about the perception of the students about the PSP course.

2.1 Student performance

The overall performance of the class is shown in the figures below. Figures 1 and 2 show the minimum, maximum and average lines of code and total development time respectively for each program for the entire class.

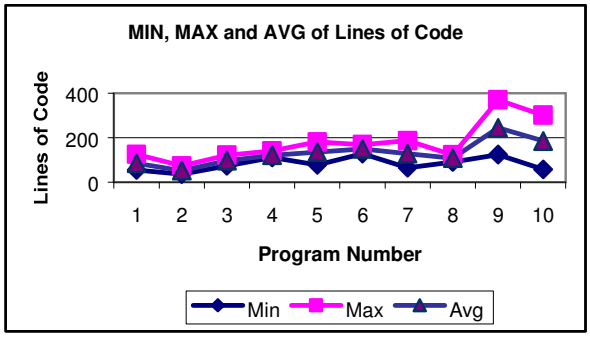

Figure 1: Minimum, maximum and average Lines of Code (LOC)

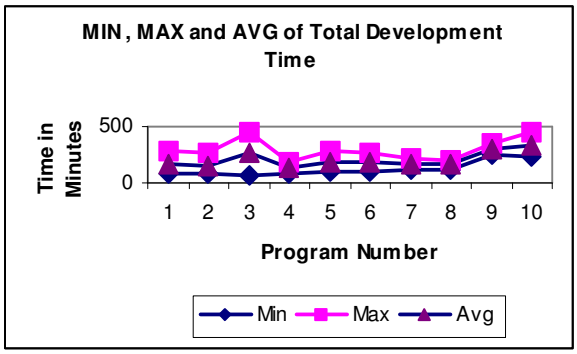

Figure 2: Minimum, maximum and average of total development time.

Figure 3 shows the defect densities over the 10 exercises of the course. We see that the total number of defects found during development per 1000 lines of code (KLOC) decreases significantly over time. Figure 4 shows that the productivity (LOC per hour of development time) is not adversely affected by the PSP during the course, despite the large amount of bookkeeping effort involved.

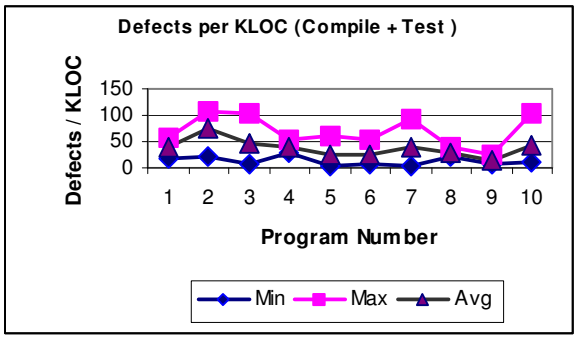

Figure 3: Defects per KLOC (Compile + Test)

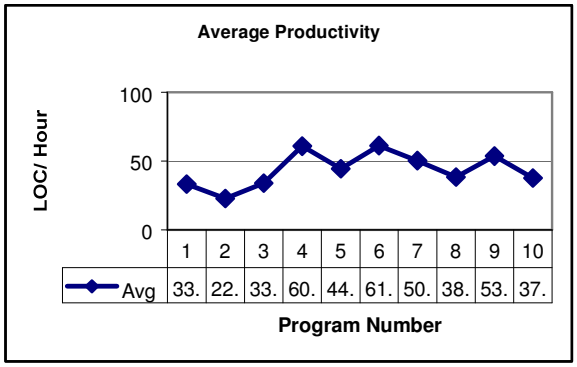

Figure 4: Average productivity

It was observed that students do not only learn to produce software with less defects, they also learn to estimate more precisely how long it will take them to deliver the product. The deviation in size estimation accuracy and time estimation accuracy over the ten exercises are shown in Figures 5 and 6 respectively.

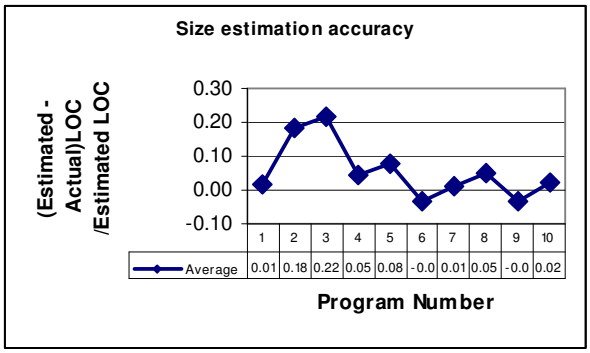

Figure 5: Size estimation accuracy

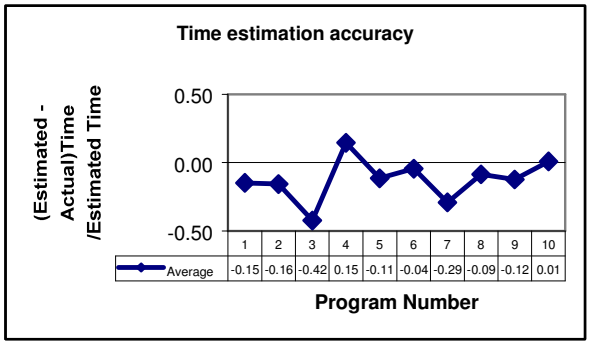

Figure 6: Time estimation accuracy

As we can see, average estimation errors are reduced significantly over the PSP course. Further, the defect removal rate of the students improved significantly over the 10 programs, as shown in Figure 7.

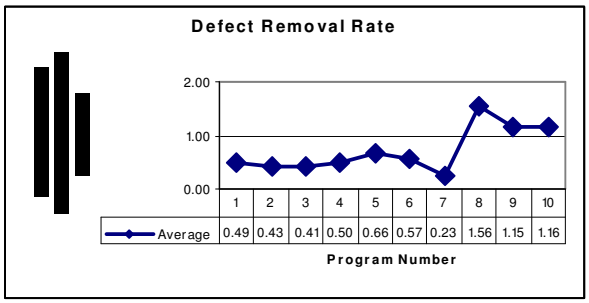

Figure 7: Defect removal rate

2.2 Course evaluation by the students

During and after taking the course, we had regular interactions with the students and obtained their feedback. More than half of the class found the PSP experience to be quite helpful in understanding software process improvement. They felt they really learned about the way they tend to work, and could see significant improvement in their overall productivity.

Although, most of them did not like filling out all the forms and collecting all the data, they grudgingly admitted that it was worthwhile to do that, especially the time logs and the defect logs.

There were several others who were hardly motivated and felt they would never really want to use PSP unless they were in an environment where it is expected of them.

3 Learning and teaching PSP

We learned important lessons in two areas: keeping the students motivated and keeping them focused on the important things.

3.1 Motivation

Most of the students were only moderately motivated to do the PSP exercises. Subjectively, the amount of bookkeeping effort required for PSP planning appears unreasonable for two reasons. First, the fraction of bookkeeping effort in the PSP course is indeed large, because the exercises are rather small. Second, students with little team project experience do not recognize why good planning is important at all.

3.2 Focus

The second significant problem in teaching PSP is that students tend to concentrate too much on the fine details of the individual methods suggested. For instance they concentrate so much on the questions like which values of the regression parameter for time estimation are acceptable ones, that they do not understand why regression is used at all and which alternative methods are used when and why.

As some of the details are indeed complicated, we find it very important to keep the students' focus on the general ideas of PSP and on the general ideas of how to implement them instead of on the details of the specific implementation suggested in the course. This requires the teacher to emphasize the rationale of each method over its actual content, and to emphasize that all methods taught in the course are only suggestions and must be optimized based on personal data after the course. Students that do not see the big picture will probably not be able to make improvements on their personal software process after the course.

4 Using a personal software process

Although a personal software process is very useful in principle, its use is hampered by a number of severe problems. We discuss each of the most important ones in a separate section.

4.1 Lack of discipline

The single most important lesson we learned on using PSP is this: Properly using and improving a personal software process requires a lot of discipline; more than most students appear to be able to come up with.

Often, introducing appropriate PSP support tools will help reduce the problem. For some students, this might still not be sufficient. The key to successful PSP use for them might be to drop most of the standard PSP elements and use only what appears most useful for them. For instance, most of the students did not use planning, because in an academic setting this is rarely practical and often superfluous.

4.2 Tool support

As mentioned above, the bookkeeping required for measurements, gathering historical data, planning, and process improvement data analysis is a tedious work. Manual bookkeeping costs time, detracts from the main task, and provokes errors. Therefore, for sustained use of a personal software process, support tools are required.

5 Conclusions

Our experiences with teaching and using PSP in a traditional software engineering course can be summarized as follows:

- PSP is a good methodology for teaching the discipline required for software engineering. Using a personal software process, the students can appreciate the importance of a defined and measured software process. They can greatly improve their personal productivity and quality of their work. Use of appropriate tools will enhance the effectiveness of the PSP significantly.

- However, for most students it is not easy to actually get PSP to work for them, mostly because of problems with self-discipline and motivation.

- When teaching PSP, it is very important to keep the students' focus on the general ideas and to educate them to judge for themselves what is useful for them and what is not. Thus they can be encouraged to adapt the PSP to suit their requirements.

6 References

[1] Watts S. Humphrey. A Discipline for Software Engineering. SEI Series in Software Engineering. Addison Wesley, 1995.

[2] Watts S. Humphrey.Using A Defined and Measured Personal Software Process. IEEE Software, 13(3):77-88, May 1996.

[3] Watts S. Humphrey. Introduction to the Personal Software Process. SEI Series in Software Engineering. Addison Wesley, 1997.

[4] J. Kamatar and W. Hayes. An Experience Report on the Personal Software Process. IEEE Software, November/December 2000, pp 85-89