Journal of Software Engineering and Applications, **2012, 5, 639-644**
http://dx.doi.org/10.4236/jsea.2012.58074 Published Online August 2012 (http://www.SciRP.org/journal/jsea)

Implementing a Personal Software Process (PSP[SM]) Course: A Case Study

Sakgasit Ramingwong, Lachana Ramingwong

Department of Computer Engineering, Faculty of Engineering, Chiang Mai University, Chiang Mai, Thailand.
Email: sakgasit@eng.cmu.ac.th, lachana@gmail.com

Received April 5th, 2012; revised May 2nd, 2012; accepted May 13th, 2012

ABSTRACT

In order to remain competitive in software industry, software engineers need to continuously improve their proficiency. Personal Software Process (PSP[SM]) provides a strong concept which helps software engineers inspecting and improving themselves. Yet, when being applied on an undergraduate computer engineering course, several complex mathematical calculations from PSP official exercises could encumber the performance of students who do not possess adequate mathematics background. This paper reports a result of implementing PSP course for undergraduate computer engineering students in Chiang Mai University, Thailand.

Keywords: Software Engineering; Personal Software Process; Process Improvement; Case Study

1. Introduction

Personal Software Process (PSP[SM]) is a software development process which focuses on improving quality of software development in an individual level. It was developed by Watt S. Humphrey and Software Engineering Institute, Carnegie Mellon University in 1989 and has been adopted by a large number of software professionals worldwide [1]. PSP utilizes actual historical data of a software engineer in order to provide precise effort estimation as well as self-propose guidelines which aim to prevent potential errors and maximize the engineers' proficiency. Efficiency of implementing PSP is highly based on each engineer's discipline and consistency.

An official PSP training course consists of two parts: PSP fundamental and PSP advanced courses [2,3]. Each course involves five full days of training. In total, both courses involve approximately 80 hours of training. The morning sessions include theories and lectures while the afternoon session focus on exercise and workshops. Nine assignments, which comprise eight programming and one written report, are assigned for the trainees.

Improvement of software development process has recently become an emerging trend in Thailand. Several leading universities, such as Naresuan University, Kasetsart University and Chiang Mai University, currently include PSP in their undergraduate and graduate curriculum [4,5]. Generally, each semester of both Thai undergraduate and graduate study consists of approximately 15 weeks. Chiang Mai University made a first attempt to implement the official PSP academic material in an undergraduate course in 2010. In order to keep the course as similar to the official training as possible, the academic materials were minimally adjusted. Nevertheless, several difficult issues surfaced during the course. One of the strongest feedbacks from the students was some exercises were too difficult. As a result, several students were demotivated, causing them unable to keep up with the subsequent exercises. This left rooms for improvement for forthcoming course implementation.

The second section of this paper describes overview of PSP. Then, a case study of attempting to implement the official PSP material in a Thai undergraduate course is described in the third section. The fourth section highlights challenges encountered from the course as well as possible solutions. Finally, the fifth section concludes the paper.

2. PSP Overview

PSP is a structured guideline for personal software development process. It aims to assist software engineers to study themselves and encouraging them to make changes as appropriate. PSP measures size, time, defects and development processes and collect them as historical data. The developers are required to manually enter these data into a provided student workbook. Then, these data are subsequently used for forecasting of effort needs for new projects. The accuracy and efficiency of the estimation are heavily depended on consistency of the software en-

gineer.

Software Engineering Institute (SEI) officially provides two PSP training courses. Firstly, PSP fundamentals training includes principles of PSP, introducing of PSP processes, templates, and data collection. Four programming assignments are given to the trainees. At the end of the course, the trainee should be familiarized with PSP process and templates. Then, PSP advanced involves introduction of formal designs, further data collection and higher level of data interpretation. Three programming projects are assigned. Additionally, one written self-assessment report based on PSP data from all projects is given at the end of the course. Details of PSP assignments on both official courses are illustrated in **Table 1**.

The accuracy of both time and size estimation of the trainees is likely to increase [1]. This is due to the growing amount of their historical data, as long as the engineers are consistent with their performance.

3. A Case Study on Implementing Official PSP Course in Thailand

Software Park Thailand arranged a PSP Academic Initiatives program in 2010 [6]. This program aimed to introduce international standards, *i.e.* PSP and Capability Maturity Model (CMMI), to academic section. More than twenty lecturers from leading Thai universities joined the courses. At the end of the program, more than half of them are officially accredited as certified PSP Developers by SEI.

During late 2010, Chiang Mai University, as one of the certified developer from PSP Academic Initiatives, made a first attempt to implement the PSP courses for undergraduate computer engineering students. The students obtained a free academic version of PSP materials which is slightly different from the official training materials [7]. Since the academic version had different preset PSP templates (for example, PSP0.1 for *Program* 2 and PSP1 for *Program* 3) and one extra exercise, the students were instructed to change the preset templates to match with the settings in the official student workbook. The last assignment, *Program* 8, which was not included in the official training, was also removed. However, an extra assignment similar to a prerequisite project of Software Park Thailand's PSP Academic Initiative was added to the assignment list as *Program* 0. This program involved a simple source line counting which could be later reused in *Program* 2. *Program* 0 did not require any data collection or estimation since its objective was to simply test the students' programming skills. As a result, the computer engineering students were given a total of nine assignments, *i.e.* eight programming and one written report.

Thirteen senior undergraduate computer engineering students enrolled in the course. All of them completed at least three course involving programming using various computer languages and are moderately to highly skilled in programming. The students used C++, PHP, Java and Visual Basic in this class. The distribution of language they used is displayed in **Figure 1**.

Table 1. Details assignments on PSP official courses.

PSP Fundamentals				PSP Advanced			
Day	*Templates*	*New Artifacts*	*Assignment*	*Day*	*Templates*	*New Artifacts*	*Assignment*
1	PSP0	• Time estimation • Time logging • Defect logging	*Program* 1. Finding mean and standard deviation from a set of data.	1	PSP2.1	• Design templates	*Program* 5. Integrate a t-distribution function using Simpson's rule. Program 1 could be reused.
2	PSP1	• Size estimation • Test report • Process improvement proposal • Coding standard • Counting standard	*Program* 2. Counting source lines of code. The counted must be able to identify parts and number of items in each part.	2	PSP2.1	N/A	*Program* 6. Reverse calculation of t-distribution integration. Program 5 could be reused.
3	PSP2	• Reviewing • Review checklists	*Program* 3. Calculating for correlation between two data sets.	3	PSP2.1	N/A	*Program* 7. Calculating for 70% prediction intervals. Program 4 and 6 could be reused.
4	PSP2.1	N/A	*Program* 4. Calculating for relative size table of a set of data. Program 1 could be reused.	4	N/A	• Applying concepts of PSP data collection to other context	*Performance Analysis Report.* Self-assessment and process improvement proposals.
5	No assignments			5	No assignments		

The Thai education system, both undergraduate and graduate, is semester based. There are two semesters in an educational year. Each semester involves approximately 15 weeks of study, excluding the examination weeks. Each subject is lectured for three hours per week, resulting in a total of 45 hours per semester. In order to cope with this academic system, the contents in PSP materials in this case study were moderately tailored. The assignments were given at week number 2, 3, 4, 6, 7, 9, 11, 12 and 14. It can be seen that there were five weeks (excluding the first introductory week) without assignments. On the weeks with assignments, all students needed to submit the projects within five days after instructed. Late penalties were applied to the students who missed the deadline. The assignments were worth 80% of the course score. In the end, all students appeared to successfully clear all assignments. **Figures 2** and **3** signify average program size (only codes which were added or modified are counted) and average development time of the class, respectively.

Interestingly, although implementing PSP, especially based on the official material, is likely to improve the accuracy of effort estimation, statistics from this class show otherwise. **Figure 4** illustrates average error on program size estimation of the class as well as its linear regression (the students were not required to estimate their *Program* 1's size thus there were no data at that point). In addition, students' average error on development time estimation and its linear regression are shown in **Figure 5**. The errors of estimation are calculated as in (1) [1].

$$\%\text{Error} = \frac{\text{Actual Data} - \text{Estimation}}{\text{Estimation}} \times 100 \quad (1)$$

Obviously, the overall estimating ability of the class was not improving. In contrast, the accuracy of the class forecasting on both program size and development time seemed to go down.

It might be possible to argue that the estimation accuracy of the program size was fluctuated. More data might be needed in order to accurately interpret the result. On the other hand, from *Program* 4, the precision of time estimation error had gone down significantly.

Figure 6 illustrates average productivity (added or modified source lines of code per hour) of the class. It can be seen that the productivity slightly continually increased. This is an adequate result since productivity could harshly drop after the official designing activities were added to the development process.

Average defect density (defects per 1000 lines of code) of the class is shown in **Figure 7**. This is probably the most outstanding result of this class since the defect density

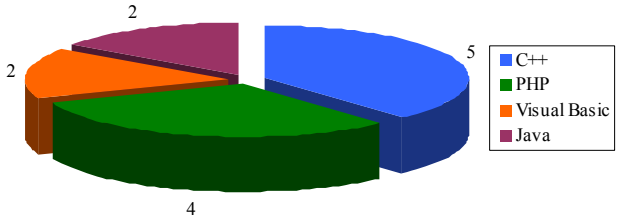

Figure 1. Distribution of programming language used in PSP course.

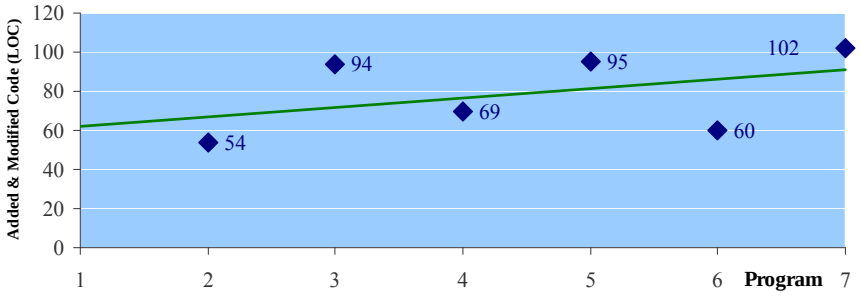

Figure 2. Average added and modified program size (lines of code).

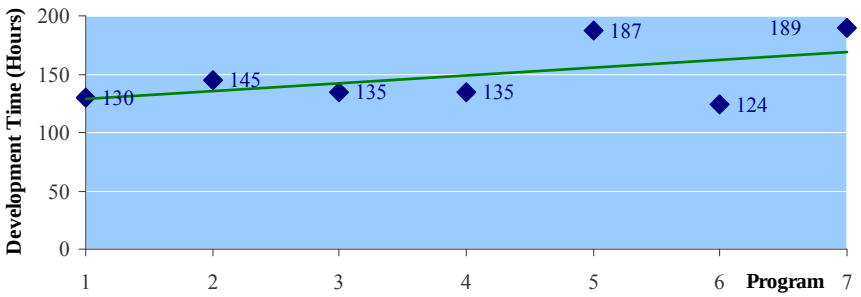

Figure 3. Average development time (hours).

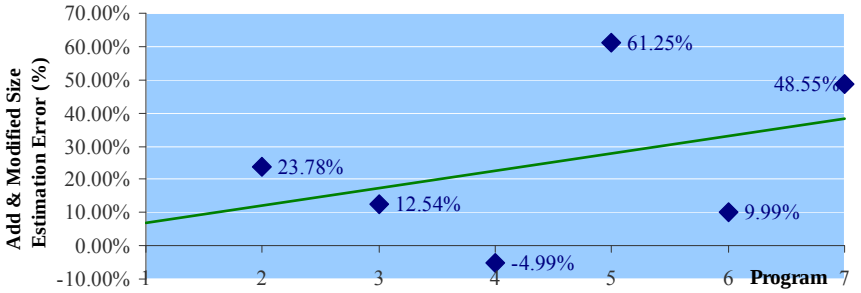

Figure 4. Average error on added and modified size estimation (%).

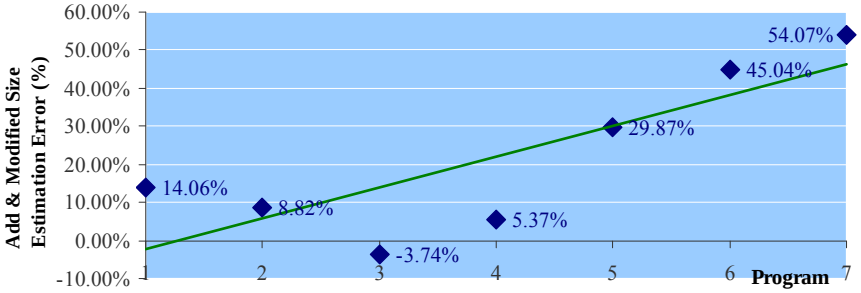

Figure 5. Average error on development time estimation (%).

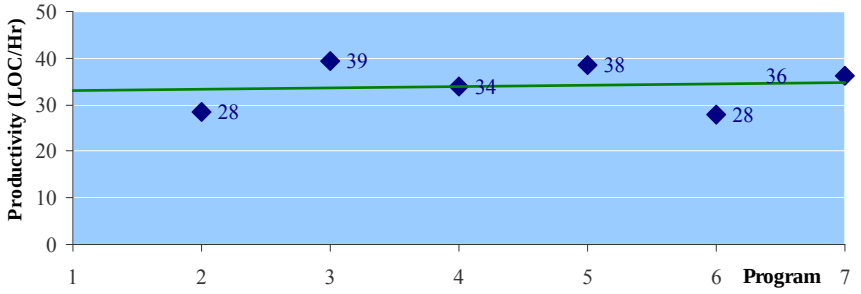

Figure 6. Average productivity (lines of code/Hour).

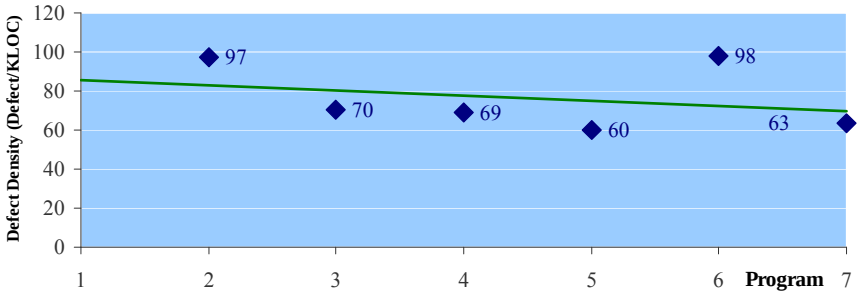

Figure 7. Average defect density (defects/1000 Lines of code).

steadily dropped throughout the course. The dropping figure suggests that the quality of development process increased. This might be a benefit from design and review processes as well as personal process improvement proposals.

4. Challenges and Possible Solutions

A number of challenges surfaced during this implementation of PSP course. Some of them were raised by the students while others were obtained by observation. Among these challenges, two issues were considered as the most serious matters.

4.1. Complexity of Exercises

One of a hidden objectives of the official PSP exercises is all trainees will be able to develop their own PSP tools

after finishing the courses. Almost all of the exercises are related to statistical tools which are used for data analysis and prediction. This involves several complex and repetitive use of equations. The complexity of the exercises continuously rises towards the end of the course. A small miscalculation in an early project can cause major errors in later projects. (2) and (3) are examples of complex calculation in the *Program* 6 which resulted in the second highest development time estimation error [7].

$$p = \frac{W}{3}\left[F(0) + \sum_{i=1,3,5\cdots}^{num\_seg-1} 4F(iW) + \sum_{i=2,4,6\ldots}^{num\_seg-2} 2F(iW) + F(x) \right] \quad (2)$$

$$F(x) = \frac{\Gamma\left(\frac{dof+1}{2}\right)}{(dof*\pi)^{1/2}\,\Gamma\left(\frac{dof}{2}\right)}\left(1+\frac{x^2}{dof}\right)^{-(dof+1)/2} \quad (3)$$

About half of the students complained about the sophisticated mathematical calculation. A few of them appeared to lose motivation. One student even admitted that, after a number of errors found in testing, he gave up his attempt and copied parts of the code from his colleague. This could result in unusable data and ultimately lead to failure of class objectives.

In order to solve this problem, alternative exercises might be considered. Since PSP mainly focuses on processes, a set of continual projects which needs a considerable development time and are partly reusable might be sufficient. Yet, the hidden objective of developing the students' own PSP tools might be voided if this solution is implemented.

4.2. Continuation of the Course

Due to the difference in course duration and occurrence, the exercises in this case study were tailored. Unfortunately, this seems to cause interruption between projects, especially on the weeks without assignments. Some students informed that they were unable to remember parts of their projects.

On the official training program, the trainees are required to develop programs everyday. Comparing a half-day and a one-week gap, it is rather obvious that the official course is more suitable to develop a series of connected programs. Yet, the impact of this particular problem might be lessened if the students are encouraged to focus more on design and documentation. In fact, optimistically, having a long gap between assignments is more similar to actual software development, especially when the programmers need to fix a deployed system or release a new version.

4.3. Other

There were also other problems that were encountered during the PSP implementation in this case. For example, many of the students had a habit of compiling as soon as finishing a line of code. If the students are strict with PSP concept, they might need to switch to the student workbook and record this activity very often. This might cause to frequent interruption to their work. In this case, this challenge was mitigated by setting up a protocol, *i.e.* if no defect was found during the compiling, the student needed not to record this process.

It was also found that the students did not spend enough time to review their designs and code. This resulted in ineffective defect prevention which is one of the main objectives of PSP. Also, a few of them were unable to submit the project within deadlines. This might be caused by their poor time management skills. Both problems need the students to increase their focus on the tasks as well as plan their development ahead.

English capability is another challenge which hindered the performance of the course. Some students were unable to follow the instructions properly, especially on the late assignments. Translating parts of the materials into student's native language might be an solution to this problem.

5. Conclusion

A PSP course was implemented in Chiang Mai University, Thailand. A group of undergraduate computer engineering attended and finally finished the class. The results of the class were rather interesting. While the estimation of source code size and development time should theoretically be increasingly more accurate, the statistics from this case study suggested otherwise. Several factors, especially complex mathematical calculations in the exercises and continuation of the course, were identified as the main issues behind this. This suggests that some tailoring effort might be needed in order to efficiently implementing PSP in particular scenarios.

REFERENCES

[1] W. S. Humphrey, "PSP(SM): A Self-Improvement Process for Software Engineers," Addison-Wesley Professional, Upper Saddle River, 2005.

[2] Software Engineering Institute, "Personal Software Process (PSP) Fundamentals," Carnegie Mellon University, Pittsburgh, 2011.

[3] Software Engineering Institute, "Personal Software Process (PSP) Advanced," Carnegie Mellon University, Pittsburgh, 2011.

[4] S. Jitprapaikulsarn, "Teaching Classes," Department of Computer Engineering, Naresuan University, Phitsanulok, 2011.

[5] Department of Computer Engineering, "219342 Software Process and Quality Assurance," Kasetsart University, Bangkok, 2011.

[6] Software Park Thailand, "PSP Academic Initiative," Software Park Thailand, Pak Kret City, 2011.

[7] Software Engineering Institute, "PSP Academic Material," Carnegie Mellon University, Pittsburgh, 2011.

