Klaus-Peter Löhr · Horst Lichter (Hrsg.) (2005): Software Engineering im Unterricht der Hochschulen, SEUH 9, Aachen, dpunkt.verlag, Heidelberg

Using Personal Software Process exercises to teach process measurement

A. Inkeri Verkamo, Asko Saura
Department of Computer Science, P.O. Box 68, FIN-00014 University of Helsinki
verkamo@cs.helsinki.fi, as@iki.fi

Abstract

Software process measurement is an essential skill to be taught to future software engineers. It is not adopted through classroom teaching, unless the students are required to try out the suggested methods, in order to experience both the advantages and the pitfalls of process measurement. The Personal Software Process (PSP) is a well known tool for this purpose. In our university we have used PSP as a part of our graduate course for teaching process measurement. Our experience and the feedback from our students suggest some changes both in the exercises and in the process itself.

1 Introduction

It is generally admitted that the basis of successful software process improvement is measurement. Any changes in the software process must be preceded by measurements of the present process, to give a basis for evaluating the effects of the change. On the other hand, measurements of the software process are not necessarily reliable or comparable unless the process itself is well defined and rigorously followed.

Students specializing in software engineering need practice in measurement. Even though many of them have some experience in real world software development, they have seldom observed process measurement in the industry, and it is even rarer that such measurements would be carried out in a systematic way. To point out both the usefulness of measurement and the problems in its rigorous implementation, the curriculum of future software engineers should include some practical exercise in process measurement.

One possible tool for teaching process measurement is the Personal Software Process (PSP) [Hum95] advocated by CMU-SEI. By doing the set of programming exercises and the corresponding process measurements the students are expected to detect and address the most important problems in their personal process, at the same time learning a systematic way of process improvement. PSP is taught in courses offered by CMU-SEI but the course textbook [Hum95] describes the method in adequate detail that it can also be applied independently.

In the University of Helsinki, we use PSP as a part of our graduate course for teaching process measurement. Our experience and the feedback from our students suggest some changes both in the exercises and in the process itself. In this report we describe the experience we have gained in using PSP, based on two master's theses that analyzed the measurements and the self-evaluations of the students [Ros03, Sau04].

2 Personal Software Process exercises

The material provided in [Hum95] is organized as ten programming exercises that are performed using the steps of PSP. In each exercise, the student collects measurements of his/her own process, such as lines of code, defects detected, and time spent in each phase. The measurements are analyzed and used for predicting corresponding values in the subsequent exercises. The number of measurements and predictions and the analysis increases with the levels of PSP in seven steps ranging from PSP0 to PSP3. In this way the student is expected to gain more understanding of his/her own process, to detect those phases of the process that are most time consuming or failure prone, and hence to be able to improve his/her personal process where it is most needed. During the course the student also produces five reports that define standards to guide his/her work and analyze the process. The exercises, process levels, and reports in PSP are shown in Table 1.

Table 1: Exercises, process levels, and reports in PSP [Hum95]

Exercise	Process	Report
1. Standard deviation	PSP0	
2. LOC counter	PSP0.1	R1 LOC counting standard R2 Coding standard
3. LOC and object counter	PSP0.1	R3 Defect analysis report
4. Linear regression	PSP1	
5. Numerical integration	PSP1.1	
6. Prediction interval	PSP1.1	R4 Midterm report
7. Correlation	PSP2	
8. Sorting	PSP2.1	
9. Chi-squared test	PSP2.1	
10. Multiple regression	PSP3	R5 Final report

The programs developed in the exercises are small statistical routines that can be used for analyzing the measurement data. The students may use any programming language that they are familiar with. They are advised to develop the software using their usual development process, extended with the measurements required in each step of PSP, so that the measurements and predictions apply to their personal way of developing software. However, since the measured phases are those that are typical to the traditional waterfall model, combining the measurements with a strongly iterative process like XP might require some redefining of phases and even adoption of more suitable measures (e.g., number of iterations). – For a more detailed description of PSP, see [Hum95].

Attempts to introduce PSP as an off-the-shelf process have been mixed successes. Industry practitioners report that they see value in the ideas presented in PSP – planning based on collected data, error prevention, etc. – but when they have the chance, they either do not start using PSP in the first place or quickly cease to use major parts of the process [Fer97, Mor00]. If the data collection and analysis overhead and the context switches from development to process analysis hamper adoption it would be useful to integrate data collection with the development tools [Joh02]. It has even been suggested that existing team processes do not produce the kind of development tasks that are easily structured as PSP projects or measured by PSP measures [Mor00].

3 PSP exercises in the University of Helsinki

In the University of Helsinki, students majoring in computer science can select software engineering as their field of specialization. As part of their specialization studies they can measure their own software process along the lines of PSP. This is offered as an alternative way of accomplishing the course *Software Processes and Quality* [Sof04], where one of the main goals is to introduce the concepts of process quality and measurement. The course gives a short introduction to PSP and TSP (Team Software Process) [Hum99] as examples of process models that rely heavily on measurement. The duration of the course is only eight weeks, and accordingly a slightly modified set of PSP exercises is used, consisting of seven programming exercises and four reports, leaving out the original exercises 7, 8, and 9 and report R4 (see Table 1).

During the years 1999–2003, a total of 164 students took the course, and 75 of them accomplished it with the PSP exercises. In the first year of the course, only a few students finished their PSP exercises, but we have gradually increased the amount of tutoring, and recently about half of the students take the PSP exercises, instead of accomplishing the course in the more traditional way of classroom exercises and exam. It can be assumed that the PSP students are typically practically rather than theoretically oriented and experienced in programming.

4 Experience

In our study, we were particularly interested in observing the effect that PSP might have on the quality of the students' software process. As a basis for our research hypotheses, let us first give a brief overview of some previously reported results.

Various sources report a declining defect density on their PSP courses [Hum96, Hay97, Abr02]. This means that students learn to produce fewer defects in the first place, which is a major PSP objective. During the first few exercises, as the students proceed from PSP0 to PSP1, the defect density declines sharply, by over 25% [Hay97]. Later, when moving from PSP1 to PSP2, the decrease is only slight. In addition to lowering the number of defects, PSP aims to address their impact by finding defects early on. Initially, over 90% of all defects are found and removed during the late phases: compile and test [Hay97]. After code and design reviews are introduced in PSP2, up to half of all defects are found in the early process phases [Hay97].

Several studies suggest improvement either in the estimation accuracy of program size or effort [Hum96, Hay97, Kam00, Pre01], but contradicting observations have also been reported [Abr02]. The PSP course mandates frequent process changes (seven out of ten new programs are written with an altered process), which means that the previously collected data quickly loses its value as a basis for predicting future performance.

PSP adds a considerable load of new tasks to each programming assignment. These include keeping records, conducting reviews, and making predictions and postmortem analyses. While these new tasks initially lower productivity, they are expected to pay off in continuously increasing productivity once the programmer has learned to make use of the tools [Hum95, p. 19]. Students have reported frustration on the amount of bookkeeping [Pre01, Abr02]. During the PSP course, students are just learning to use the process, which means we should expect their productivity to decrease while they are adding new process elements to their work. Surprisingly, many sources report a basically unchanging average productivity over PSP courses [Hay97, Pre01, Abr02].

We formulated the following hypotheses of the expected results of our PSP course:

H1 *Over the PSP course, defect density decreases.*

H2 *The most important defect removal phase changes during the course: In the beginning, most defects are removed in compile. Later on, testing and reviews will account for removing the most defects.*

H3 *Over the course, both size and effort prediction accuracy will increase.*

H4 *Despite the process changes, productivity will stay roughly the same.*

We analyzed the coursework reports of 36 students from our three first PSP courses (1999–2001). The data collected by students was of poor quality, as has also been noted elsewhere [Joh98]. Before analyzing the data, we manually corrected many obvious errors in the reports.

Klaus-Peter Löhr · Horst Lichter (Hrsg.) (2005): Software Engineering im Unterricht der Hochschulen, SEUH 9, Aachen, dpunkt.verlag, Heidelberg

As expected in hypothesis H1, defect density decreased over the course. We calculated the density in defects per KLoC (Figure 1). Median defect density decreased constantly. Strong decrease in defect density occurred at exercises 4 and 10. Exercise 4 was the first exercise to use PSP1 and exercise 10 was the first and only exercise to use PSP3. However, as PSP1 only adds the size estimation tool PROBE [Hum95] to the process, it is unlikely that the decrease of defect density in exercise 4 was caused by this process change. We suspect that the very sharp drop on exercise 10 may result from it being a very big exercise and the last one, so that the students were just tired of the whole thing. Another explanation is that they used the reviews adopted in exercise 6 to avoid some defect types.

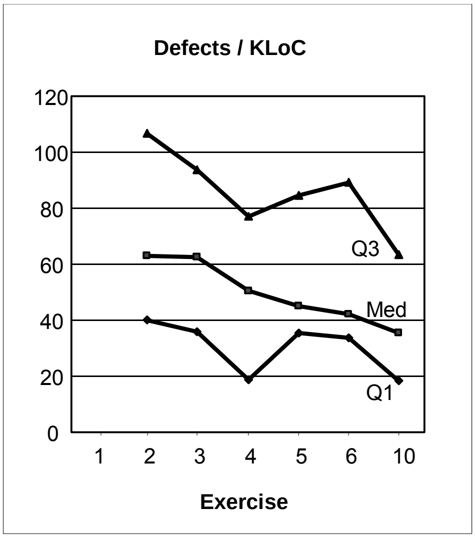

Figure 1: Defect density by exercise, in defects / KloC.

Surprisingly, hypothesis H2 proved to be false. Contrary to the previous reports, the compile phase accounted for about half of all removed defects during the entire course (Figure 2). Introducing reviews in exercise 6 sharply decreased the amount of compile defects, but the compile phase still remained on top. At the same time, defects found in testing were noticeably less. Perhaps our students would require more coaching on review methods.

Klaus-Peter Löhr · Horst Lichter (Hrsg.) (2005): Software Engineering im Unterricht der Hochschulen, SEUH 9, Aachen, dpunkt.verlag, Heidelberg

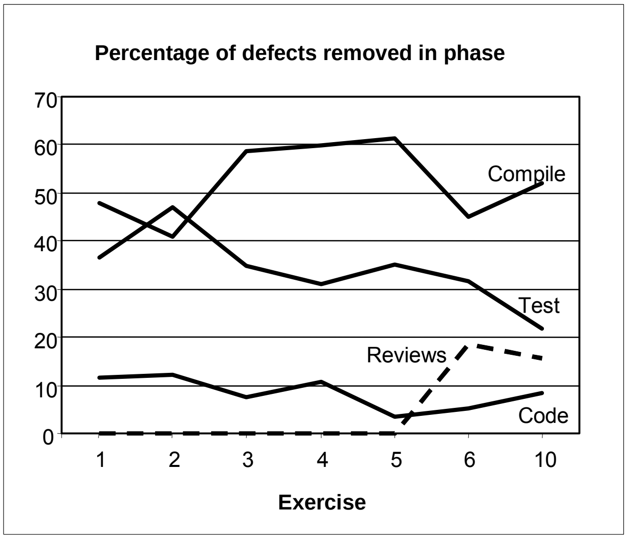

Figure 2: Percentage of defects removed by phase.

Both effort and size estimation accuracy increased clearly during the course (Figure 3), especially after the size estimation tool was introduced in exercise 4. After exercise 4, only exercise 6 shows major underestimation of effort, possibly because it introduces the time-consuming reviews to the process. Many students heavily underestimated the size of exercise 10 which was considerably larger than all previous exercises.

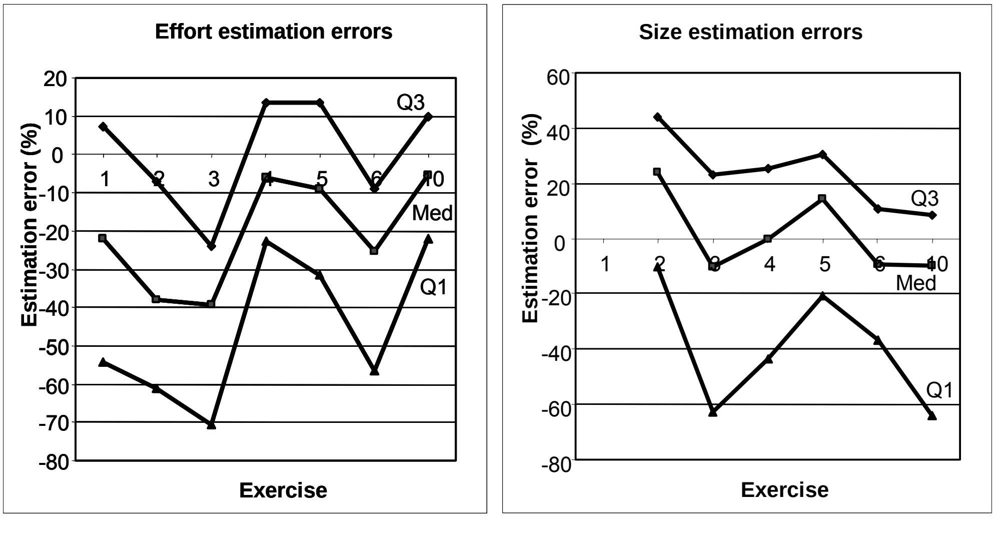

Figure 3: Estimation errors by exercise.

Hypothesis H4 expected productivity to stay roughly constant for the duration of the course, but this proved false (Figure 4). There is a noticeable drop in average productivity when PSP1 is introduced (exercises 4 and 5). This may be due to the time-consuming size prediction and paperwork. The average productivity stays low, except for the last exercise, where, as we previously noted, students may not spend too much time searching for the last few defects. Moreover, the additional PSP1 process elements seem to slow down the most productive students more than the others (Q3 in Figure 4).

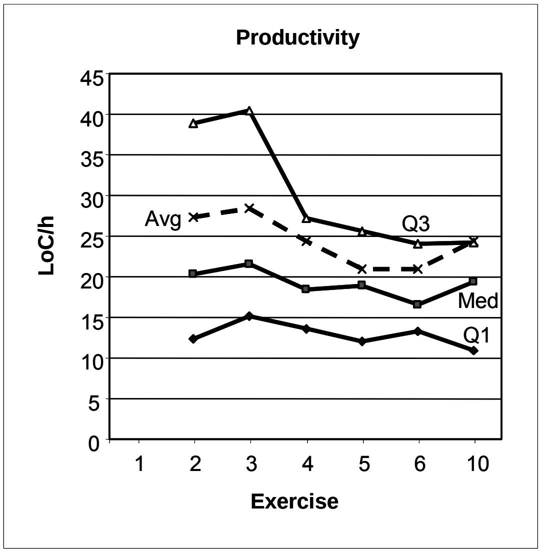

Figure 4: Productivity by exercise, in LoC per hour.

In their final reports, the students evaluated the PSP exercises and their own learning during the course. They considered the series of exercises a good introduction to a measurement based process, but to be really valuable, more exercise is needed: many students were disappointed with their own prediction skills and their slow (or nonexistent) improvement. Nevertheless they felt that by guiding them to plan their entire process, instead of just starting to code from some part of the task, the method could potentially help them in learning to make better estimates. Reviewing one's own work was considered difficult and even unmotivating, but still several of the students mentioned reviews as one of the tools that they might consider using in the future.

5 Future plans

Based on our experience we consider PSP a useful small scale tool for teaching process measurement. The basic measurements (time, size, and number of defects) are easily comprehensible to the students. The series of exercises is long enough to allow some learning, and also to show some basic difficulties in starting a measurement program.

The main drawback of the PSP exercises is the continuously changing process. In our course, seven exercises are performed using six different processes (in the original course, ten exercises using seven processes), which means that in almost every exercise the student is required to learn some new measures. In consequence, the measurement data collected during the exercises represent the student's process in a learning phase rather than in actual use. When predictions of future performance are based on this data the result may be haphazard. As we observed, some students grasp the new concepts quickly and have stable results, while others may have one or several exercises that are too much influenced by transient difficulties to be useful at all. The small number of measurements emphasizes this problem, since there is little room for discarding even obvious outliers from the small collection of historical data.

To be useful for collecting historical data the process should be the same from one measurement to the other. Hence we plan to change our set of exercises to use only two processes. In the first 4 exercises the students use their own basic process with added measurements for size, time, and defects. The measurements on these exercises form the historical data that is analyzed to define a basis for process improvement. The analysis should reveal the most time consuming and/or most error prone phases of the process, and the student can then choose one of several strategies to improve his/her process. The last 3 exercises are then performed using the modified process, and again the collected measurement data can be used to analyze the effect of the change.

A second source of dissatisfaction was the content of the programming exercises. The software produced in the exercises consists of statistical software, which is very different from the kind of programs that the students are used to working on. Moreover, statistical software packages exist in abundance, so that writing yet another piece of software for this purpose is not motivating. The advantage of using (and testing) your own software in doing the statistical calculations needed for measurement analysis is outweighed by the difficulty of understanding the concepts under a strained schedule. We plan to change the topic of the exercises into something more familiar to the students. As before, the exercises should form a continuum that allows and encourages reuse of previous work, at the same time producing software that the students might have a chance of using even later in their studies or work.

Finally, a strong point of criticism that we feel necessary to address is the lack of measurement and analysis tools. The original collection of paper sheets where students write down their measurements by hand has already been replaced by web pages, both in our course and elsewhere [Joh98, Mor00, Bor02]. Likewise the students are provided Excel sheets with calculations of the various quality measures. A more controversial topic is to what extent the analysis should be automated. Certainly when this kind of analysis is used routinely in the industry, the more it can be automated the better [Mor00]. On the other hand, at the learning phase it is crucial to understand where the numbers come from: we have numerous examples of students who will accept any value computed by the Excel sheet, no matter how unreasonable it may be.

Nevertheless we hope to improve the tool assistance offered for PSP, by providing simple tools for collecting the measurements.

6 Conclusion

We feel that in spite of certain weaknesses, PSP forms a useful basis for teaching the art and practice of process measurement to future software engineers. While the students may find detailed and rigorous bookkeeping and analysis tedious, they are also rewarded with an understanding of how effective even simple means of prediction are in software development. With the improvements suggested above we expect the PSP exercises to be even more useful in improving the students' awareness of the methods of process improvement.

References

[Abr02] P. Abrahamsson, K. Kautz: Personal Software Process: classroom experiences from Finland. In: J. Kontio, R. Conradi (ed.), Software Quality – ECSQ 2002, number 2349 in Lecture Notes in Computer Science, pages 175-185. Springer. June 2002

[Bor02] J. Börstler, D. Carrington, G. W. Hislop, S. Lisack, K. Olson, L. Williams: Teaching PSP: challenges and lessons learned. IEEE Software, 19(5):42-48. 2002

[Fer97] P. Ferguson, W. S. Humphrey, S. Khajenoori, S. Macke, A. Matvya: Results of applying the Personal Software Process. Computer, 30(5):24-31. 1997

[Hay97] W. Hayes, J. W. Over: The Personal Software Process (PSP): an empirical study of the impact of PSP on individual engineers. Technical Report CMU/SEI-97-TR-001. Carnegie Mellon University, Software Engineering Institute. 1997

[Hum95] W. S. Humphrey: A discipline for software engineering. Addison-Wesley. 1995

[Hum96] W. S. Humphrey: Using a defined and measured Personal Software Process. IEEE Software, 13(3):77-88. 1996

[Hum99] W. S. Humphrey: Introduction to the Team Software Process. Addison-Wesley. 1999

[Joh98] P. M. Johnson, A. M. Disney: The Personal Software Process: a cautionary case study. IEEE Software, 15(6):85-88. 1998

[Joh02] P. M. Johnson, H. Kou, J. Agustin, C. Chan, C. Moore, J. Miglani, S. Zhen, W. E. J. Doane: Beyond the Personal Software Process: Metrics collection and analysis for the differently disciplined. Proc. 2003 International Conference on Software Engineering. Portland, Oregon. 2003

[Kam00] J. Kamatar, W. Hayes: An experience report on the Personal Software Process. IEEE Software, 17(6):85-89. 2000

[Mor00] M. Morisio: Applying the PSP in industry. IEEE Software, 17(6):90-95. 2000

[Pre01] L. Prechelt, B. Unger: An experiment measuring the effects of Personal Software Process (PSP) training. IEEE Transactions on Software Engineering, 27(5):465-472. 2001

[Ros03] R. Rossi: Adoption of the Personal Software Process (in Finnish). Technical Report C-2003-68. Department of Computer Science, University of Helsinki. 2003

[Sau04] A. Saura: Software fault density and workload estimation in the PSP (in Finnish). Technical Report C-2004-28. Department of Computer Science, University of Helsinki. 2004

[Sof04] Department of Computer Science, University of Helsinki: Software Processes and Quality, course home page. *http://www.cs.helsinki.fi/u/verkamo/swpr/swprs2004_eng.html*