Software Engineering Institute

TSP Symposium 2013 Proceedings

Sergio Cardona, Universidad del Quindío, Colombia
João Pascoal Faria, University of Porto
Fernanda Grazioli, Universidad de la República
Pedro Henriques, Strongstep – Innovation Center in Software Quality
James McHale
Silvana Moreno, Universidad de la República
William Nichols
Leticia Pérez, Universidad de la República
Mushtaq Raza, University of Porto
Rafael Rincón, Universidad EAFIT, Colombia
Diego Vallespir, Universidad de la República

January 2014

SPECIAL REPORT
CMU/SEI-2013-SR-022

Software Solutions Division

http://www.sei.cmu.edu

Carnegie Mellon University

This material is based upon work funded and supported by Cost recovery under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of Cost recovery or the United States Department of Defense.

This report was prepared for the
SEI Administrative Agent
AFLCMC/PZM
20 Schilling Circle, Bldg 1305, 3rd floor
Hanscom AFB, MA 01731-2125

DM-0000833

Table of Contents

List of Figures

List of Tables

Abstract

The 2013 TSP Symposium was organized by the Software Engineering Institute and took place September 16–19 in Dallas, Texas. The goal of the TSP Symposium is to bring together practitioners and academics who share a common passion to change the world of software engineering for the better through disciplined practice. The conference theme was "When Software Really Matters," which explored the idea that when product quality is critical, high-quality practices are the best way to achieve it. In keeping with that theme, the community contributed a variety of technical papers describing their experiences and research using the Personal Software Process℠ (PSP℠) and Team Software Process℠ (TSP℠). This report contains the four papers selected by the TSP Symposium Technical Program Committee. The topics include demonstrating the impact of the PSP on software quality and effort by eliminating the programming learning effect, analyzing student performance during the introduction of the PSP using an empirical cross-course comparison, incorporating PSP practices into introductory programming courses, and analyzing factors affecting productivity performance in PSP training.

1 Introduction

James McHale

The 2013 TSP Symposium was organized by the Software Engineering Institute (SEI) and took place September 16–19 in Dallas, Texas. The goal of the TSP Symposium is to bring together practitioners and academics who share a common passion to change the world of software engineering for the better through disciplined practice. The conference theme was "When Software Really Matters," which explored the idea that when product quality is critical, high-quality practices are the best way to achieve it. In keeping with that theme, the community contributed a variety of technical papers describing their experiences and research using the Personal Software Process[SM] (PSP[SM]) and Team Software Process[SM] (TSP[SM]).

The technical program committee consisted of Barry Dwolatzky, University of Witwatersrand; Elias Fallon, Cadence Design Systems; João Pascoal Faria, University of Porto; Jared Freeman, Naval Oceanographic Office; Bradley Hodgins, Naval Air Systems Command; Mark Kasunic, Software Engineering Institute; James McHale, Software Engineering Institute; Yuri Ontibon, SEONTI; David Ratnaraj, Advanced Information Systems; Rafael Salazar, Tecnológico de Monterrey; Diego Vallespir, Universidad de la República (Uruguay); and Alan Willett, Oxseeker.

This year's report contains four papers that focus on PSP in an academic environment with somewhat broader implications not only for TSP but also for new process introduction. Among other things, the papers selected this year show that PSP provides a consistent empirical platform that lends itself to both effective instruction and valid experimentation.

Demonstrating the Impact of the PSP on Software Quality and Effort: Eliminating the Programming Learning Effect (Diego Vallespir, Fernanda Grazioli, Leticia Pérez, and Silvana Moreno) investigates whether it is the individual practices of PSP or the similar nature of the standard programming assignments that leads to better quality and estimating. Both are hallmarks of PSP.

An Analysis of Student Performance during the Introduction of the PSP: An Empirical Cross-Course Comparison (Fernanda Grazioli, William Nichols, and Diego Vallespir) looks at the effects of the different available course sequences of PSP on various dimensions of student performance.

Incorporating Some PSP Practices into Introductory Programming Courses: A Case Study in Universidad del Quindío (Sergio Cardona, Rafael Rincón, and Diego Vallespir) documents an interesting approach to determine if various aspects of PSP can be integrated effectively with existing introductory programming classes, potentially eliminating the need for a separate course to train PSP techniques.

Factors Affecting Productivity Performance in PSP Training (Mushtaq Raza, João Pascoal Faria, Pedro Henriques, and William Nichols) examines data from approximately 3,000 students for personal and process factors that account for variations in student productivity.

2 Demonstrating the Impact of the PSP on Software Quality and Effort: Eliminating the Programming Learning Effect

Diego Vallespir, Universidad de la República
Fernanda Grazioli, Universidad de la República
Leticia Pérez, Universidad de la República
Silvana Moreno, Universidad de la República

Abstract

Data collected in the Personal Software Process (PSP) courses indicate that the PSP improves the quality of the products developed and reduces the development effort. One way this has been determined is through statistical analysis of the evolution of the results (for example, defect density in unit test) obtained by the students in each program of the PSP training course. However, since the programs are in the same application domain, the improvement could be due to programming repetition (i.e., the learning effect). To explore the reasons for the improvements, we asked the following research question: Are the improvements observed in the PSP courses due to the introduction of the phases and techniques of the PSP or to programming repetition? To investigate this, we designed and performed a controlled experiment with 12 software engineering undergraduate students at the Universidad de la República. The students performed the exercises from the PSP for Engineers I/II course without applying the PSP techniques. The overall results indicate that the practices introduced by the PSP, and not programming repetition, contributed to the performance improvements.

2.1 Introduction

Data collected in the Personal Software Process (PSP) courses indicate that the PSP improves the quality of the products developed and reduces the development effort [Hayes 1997, Rombach 2008]. The students (typically software engineers) perform several programming exercises in which techniques and phases of the PSP are added as the exercises advance. One way it has been determined that the PSP improves individual performance is through statistical analysis of the evolution of the results (for example, defect density in unit test) obtained by the students in each program of the PSP training course. For example, if the programs developed by the students during the course are of a better quality as the course progresses, then it can be statistically inferred that the PSP is responsible for the quality improvement.

However, since the programs of the course are in the same application domain, the improvement could be due to programming repetition (i.e., the learning effect). Recently, a study that compared the data obtained from different versions of the PSP courses (in which the phases and techniques of the PSP are introduced at different moments as the exercises advance) concluded that the changes in quality most plausibly regard mastering PSP techniques rather than programming repetition [Grazioli 2012].

Our work aims contribute in this same direction but uses a different approach. To explore the reasons for the improvements, we asked the following research question: Are the performance improvements observed in the PSP courses due to the introduction of the phases and techniques of the PSP or to programming repetition? To investigate this, we designed and performed a controlled experiment with 12 software engineering undergraduate students at the Universidad de la

República. The students performed the exercises from the PSP for Engineers I/II course without applying the PSP techniques.

The results of our experiment show that there is no improvement in the performance of the software engineer concerning product quality and testing effort. This indicates that the practices introduced by the PSP, and not programming repetition, contribute to performance improvements.

2.2 Experiment Setup

This section presents the goals, metrics, hypotheses, subjects, experimental material, and experimental design.

2.2.1 Goals, Metrics, and Hypotheses

The goal of our experiment is to know whether the improvement of software engineers' performance when they develop the programs used in the PSP course is due to programming repetition in the same application domain. The aspects of performance that we considered are quality of the product and the effort required in unit testing (UT).

To determine the quality of the products, we used two measures: defect density in unit test and total defect density of the program (dependent variables of the experiment). These are normally used in experiments that involve the PSP. The defect density was measured as the number of defects per every thousand lines of code (KLOC). The effort used in unit testing was also measured in two ways: time in unit testing per KLOC and average time in unit testing per defect found.

A statistical hypothesis is an assumption about a population parameter. This assumption may or may not be true. Hypothesis testing refers to the formal procedures used in experimentation to accept or reject statistical hypotheses.

There are two types of statistical hypotheses. The null hypothesis, denoted by H0, is usually the hypothesis that sample observations result purely from chance. The alternative hypothesis, denoted by H1, is the hypothesis that sample observations are influenced by some nonrandom cause. The aim of the hypothesis test is to determine whether it is possible to reject the null hypothesis H0 [Juristo 2001].

The experiment raised the null hypotheses and their respective alternative hypotheses for each of the four mentioned metrics. The hypotheses aimed to compare a developed program to another one developed previously to determine whether software engineers improved their performance in any of the aspects mentioned.

We compared programs by pairs to find whether the changes in each dependent variable for performance were statistically significant:

H0 def ut: Median (Defect Density in UT *i*) = Median (Defect Density in UT *j*)

H1 def ut: Median (Defect Density in UT *i*) <> Median (Defect Density in UT *j*)

where *i*, *j* are the numbers of the programs (1 to 8) and $i < j$

The same types of null and alternative hypotheses were raised for the other three dependent variables.

2.2.2 Subjects

The subjects of the experiment were Computer Science undergraduate students of the Universidad de la República of Uruguay, all of them advanced students in their fourth or fifth year. They had completed the course Programming Workshop, in which they learned the Java language, and they had completed at least three more programming courses and a course on object-oriented languages. We consider therefore that the group that participated in the experiment was homogeneous due to the students' similar advancement in their careers.

The students participated in the experiment in order to obtain credits for their careers, and that was their motivation. It was mandatory for them to attend the theory classes (lectures) where the software development process used (PSP0 and PSP0.1) was presented. It was also mandatory for them to follow the scripts provided and to collect the data using the tool for that purpose. The students did not know that they were taking part in an experiment; they thought that they were taking a course with an important component of laboratory practices. They did know, however, that the data they collected would be used in research work, and they gave their written consent for it.

Finally, participation in the course by the students was voluntary. This course was not mandatory for their Computer Science degrees; therefore, enrolling in it was optional.

2.2.3 Experimental Material

The experimental material was made up of the process scripts of PSP0 and PSP0.1, the requirements of the Programs 1 to 8 used in the PSP course, and the tool for data collection. All this material was exactly the same as that used in the PSP for Engineers I/II courses (in the eight-program version). The tool for data collection was the one distributed by the SEI (the PSP support tool developed in Microsoft Access).

2.2.4 Experimental Design

The design of this experiment was a repeated measures design. Twelve students developed eight software programs following an established process. The eight programs were the same for the 12 participants and were developed in the same order. These programs, as previously mentioned, are the ones used in the PSP for Engineers I/II course.

The students used the PSP0 for the first program and the PSP0.1 for the remaining seven programs. These two levels of the PSP aim only to collect data of the process (time, defects, etc.) and do not introduce the practices of the PSP (reviews, design, PROBE, etc.). This design of the experiment made it possible to know whether the students improved their performance due to programming repetition.

We refined our goal using the Goal Question Metric approach [Basili 1994]:

Analyze and compare the data collected at eight program assignments

for the purpose of evaluating individual performance improvements

with respect to defect density in unit testing, total defect density, time spent in unit testing per KLOC, and average time spent in unit testing per defect found

from the viewpoint of a researcher in the context of the PSP0.1 level training of 12 undergraduate students

2.3 Results and Discussion

Table 1 presents median and interquartile ranges of the four variables under study for Programs 1 to 8.

Table 1: Median and Interquartile Ranges for the Four Variables Under Study

Defect Density in Unit Testing (# defects found in UT / KLOC)								
	Pr 1	**Pr 2**	**Pr 3**	**Pr 4**	**Pr 5**	**Pr 6**	**Pr 7**	**Pr 8**
Median	24.55	56.98	18.13	18.48	36.38	18.40	13.78	8.59
IQR[a]	13.65	21.20	31.84	18.14	30.19	17.11	25.11	12.20
Total Defect Density per KLOC (# defects found / KLOC)								
Median	111.11	136.59	72.51	74.04	137.00	61.33	63.80	40.06
IQR	49.19	151.87	89.24	51.52	124.61	51.31	83.18	63.14
Time Spent in Unit Testing per KLOC (minutes in UT / KLOC)								
Median	331.28	1297.97	301.52	241.94	638.80	652.71	540.85	338.76
IQR	335.59	1044.97	345.24	301.34	1136.47	1297.96	523.87	490.12
Average Time Spent in Unit Testing per Defect (minutes in UT / # defects found in UT)								
Median	11.33	16.61	15.00	11.75	20.50	37.00	29.00	39.00
IQR	7.75	17.46	10.00	15.75	12.17	40.75	37.00	28.25

a. IQR, interquartile range.

There were 12 students in our experiment (few samples), and the data of each one in the eight exercises of the PSP was considered (repeated measures). In a context of few samples and repeated measures, the most suitable statistical hypotheses test is the Wilcoxon signed-ranks test [Wilcoxon 1945]. This test is used to compare two sets of scores that come from the same subjects and when normality cannot be assumed. It is the nonparametric test equivalent to the dependent *t* test. We used the two-tailed Wilcoxon test because we did not know a priori if the dependent variables would increase or reduce their values.

Table 2 presents the results of applying the Wilcoxon test to each pair of programs for the hypothesis of defect density in unit test (DDUT). The table presents the comparison between pairs of programs. Each cell contains the *p* value (two-tailed) of the Wilcoxon test. The cells in green and red indicate that the null hypothesis has been rejected ($p \leq 0.05$). The green ones also indicate that there was an improvement in defect density in UT as the students advanced in the exercises; the red ones indicate the opposite. The gray cells indicate that it was not possible to reject the null hypothesis.

It can be observed that it is statistically significant that the defect density in UT for Program 2 is higher than in the rest of the programs. There is one motive that can explain this behavior. Program 2 of the PSP course is the only one that is not a mathematical program. Exercise 2 consists of developing a program to count lines of code for a program. Although this can be a cause for a higher defect density, we cannot assure so.

Table 2: Wilcoxon Test for DDUT

Program	2	3	4	5	6	7	8
1	$p = 0.028$	$p = 0.722$	$p = 0.158$	$p = 0.347$	$p = 0.136$	$p = 0.388$	$p = 0.006$
2		$p = 0.006$	$p = 0.003$	$p = 0.019$	$p = 0.002$	$p = 0.010$	$p = 0.002$
3			$p = 0.754$	$p = 0.084$	$p = 0.937$	$p = 0.754$	$p = 0.272$
4				$p = 0.117$	$p = 0.929$	p = 1.000	$p = 0.136$
5					$p = 0.015$	$p = 0.084$	$p = 0.006$
6						$p = 0.929$	$p = 0.084$
7							$p = 0.209$

In Program 5, the defect density in UT is statistically higher than those found in Programs 6 and 8. But the hypothesis cannot be rejected between Programs 5 and Programs 3, 4, and 7.

These results show there is not a continuous improvement as regards defect density in UT. Removing Program 2 from the analysis, no difference can be detected between Program 3 and the following, or between Program 4 and the following, or between Program 6 and Programs 7 and 8. The differences found between Programs 5 and 6, and between Programs 5 and 8, may be due to the characteristics of Program 5. However, other experiments are necessary to prove it. This is different from the improvements found when the regular course was used [Hayes 1997, Rombach 2008].

Table 3 presents the results of applying the Wilcoxon test to each pair of programs for the hypothesis of total defect density (TDD) per KLOC. The colors are used in the same way as in Table 2.

Table 3: Wilcoxon Test for TDD per KLOC

Program	2	3	4	5	6	7	8
1	$p = 0.239$	$p = 0.239$	$p = 0.010$	$p = 1.000$	$p = 0.004$	$p = 0.041$	$p = 0.008$
2		$p = 0.034$	$p = 0.010$	$p = 0.158$	$p = 0.003$	$p = 0.006$	$p = 0.005$
3			$p = 0.695$	$p = 0.182$	$p = 0.041$	$p = 0.530$	$p = 0.034$
4				$p = 0.050$	$p = 0.108$	$p = 0.480$	$p = 0.050$
5					$p = 0.004$	$p = 0.084$	$p = 0.012$
6						$p = 0.754$	$p = 0.347$
7							$p = 0.158$

Programs 6 and 8 show an improvement in the total density of defects injected compared to previous programs. However, this does not happen with Program 7, which only shows an improvement compared to Programs 1 and 2. Although we can observe that statistically there is not a continuous improvement, we do observe that Programs 1, 2, and 5 show higher numbers of injected defects than the rest of the programs. In Programs 6 and 8, the subjects have less injection of defects. This improvement may be due to the fact that the subjects recorded their own injected defects from Program 1. This practice, not carried out normally, raises awareness of the type of defects that the person usually injects, apparently provoking a smaller number of injected defects.

Table 4 presents the results of applying the Wilcoxon test to each pair of programs for the hypothesis of time spent in unit testing (TSUT) per KLOC. The red color indicates statistical evidence of an increase in the time spent, green indicates a decrease, and gray indicates that the null hypothesis could not be rejected.

Table 4: Wilcoxon Test for TSUT per KLOC

Program	2	3	4	5	6	7	8
1	$p = 0.005$	$p = 0.937$	$p = 0.388$	$p = 0.023$	$p = 0.019$	$p = 0.308$	$p = 0.754$
2		$p = 0.023$	$p = 0.003$	$p = 0.209$	$p = 0.433$	$p = 0.034$	$p = 0.003$
3			$p = 0.530$	$p = 0.117$	$p = 0.136$	$p = 0.480$	$p = 0.638$
4				$p = 0.012$	$p = 0.015$	$p = 0.209$	$p = 0.480$
5					$p = 0.209$	$p = 0.308$	$p = 0.041$
6						$p = 0.117$	$p = 0.028$
7							$p = 0.530$

In this case, there is not a steady improvement in the performance either. The improvement considered is to reduce the necessary time in UT per KLOC. The results show that it is worse in Program 5 (compared to 4) and in Program 6 (also compared to 4). Program 8 shows an improvement concerning Programs 2 to 5 and 6. However, there is no statistical evidence of an improvement concerning Programs 3 and 4. This shows that programming repetition (using these programs) does not result in an improvement in the time spent in UT per KLOC.

Table 5 presents the results of applying the Wilcoxon test to each pair of programs for the hypothesis of average time spent in unit testing (TSUT) per defect found in UT. The colors are used in the same way as in Table 2.

Table 5: Wilcoxon Test for Average TSUT per Defect

Program	2	3	4	5	6	7	8
1	$p = 0.050$	$p = 0.155$	$p = 0.575$	$p = 0.059$	$p = 0.021$	$p = 0.047$	$p = 0.010$
2		$p = 0.859$	$p = 0.389$	$p = 0.929$	$p = 0.038$	$p = 0.093$	$p = 0.010$
3			$p = 0.214$	$p = 0.386$	$p = 0.051$	$p = 0.386$	$p = 0.041$
4				$p = 0.594$	$p = 0.051$	$p = 0.093$	$p = 0.009$
5					$p = 0.008$	$p = 0.047$	$p = 0.004$
6						$p = 0.575$	$p = 0.878$
7							$p = 0.790$

The results indicate that in the last three programs the UT average time per defect found in general increases. In particular, Program 8 presents statistical evidence that the average time spent in UT per defect found is more than in Programs 1 to 5. Therefore, the results show that in the last programs the efficiency of UT (defects found per unit of time) decreases. There are several possible reasons for this: fewer defects that reach the UT phase, more tests carried out that lead to a greater effort in UT, and less effectiveness in the tests (percentage of defects found in the total number of defects that get to UT).

We have already shown in the first analysis that the defects that get to UT do not decrease per KLOC statistically for certain comparisons between programs, in particular many of the ones that are presented in red. On the other hand, the effort per KLOC in UT even decreases for some pairs of programs that appear in red. The last possible reason (effectiveness of UT) cannot be discussed within the frame of our experiment. Therefore, we cannot clearly establish the reason for the loss of efficiency in UT in the context of this experiment.

To sum up, since the experiment does not change the level of PSP used (PSP0.1 from Program 2 to 8), the results of this experiment indicate that the programming repetition in the same application domain and the collection of data of the processes

- do not continuously improve defect density in UT
- seem to improve in the last three programs the total defect injection (This can be due more to the data collection about the defects injected than to the learning effect of the application domain.)
- do not continuously improve the time spent in UT per KLOC
- seem to deteriorate the efficiency of UT

2.4 Conclusions and Future Work

The presented results contribute to eliminating an important threat to the validity of different experiments performed with the PSP. These results agree with a previous result that indicates that the practices introduced by the PSP and not programming repetition contribute to the improvement of individual performance [Grazioli 2012]. Moreover, as both studies show the same kind of results by following different approaches, the confidence in the conclusions increases. Furthermore, we found that there is a different behavior in Program 2 and in Program 5 regarding software quality. This behavior, which we showed is independent from the PSP practices, has to be analyzed more deeply by performing new controlled experiments.

In addition, this experiment shows that without adequate practices the quality of software and the performance of the process cannot be improved simply through the programming learning effect. Someone once said, "Insanity is when you keep doing the same things while expecting different results."[1] In other words, it is impossible to improve without implementing changes. In fact, the changes suggested by the PSP are the ones that generate the improvements in the performance of the software engineer.

Our future work will compare the data we have obtained with the results that are normally found in the PSP courses. We also intend to replicate this experiment, analyze other data, and design a more complex experiment that will enable us to isolate and study the different practices of the PSP and the synergy produced between them.

[1] This quotation or variants of it are attributed to different persons, among them, Benjamin Franklin, Rudyard Kipling, Albert Einstein, Rita Mae Brown, and a Chinese proverb. We could not find out who is the original author of that phrase.

2.5 Author Biographies

Fernanda Grazioli
Research Assistant
School of Engineering, Universidad de la República
Fernanda Grazioli is a research assistant at the Engineering School at the Universidad de la República (UdelaR). She is also a member of the Software Engineering Research Group (GrIS) at the Instituto de Computación (INCO). Grazioli holds an Engineering degree in Computer Science from UdelaR and a Master of Science in Computer Science from the same university.

Silvana Moreno
Teaching and Research Assistant
School of Engineering, Universidad de la República
Silvana Moreno is a teaching and research assistant at the Engineering School at the Universidad de la República (UdelaR). She is a member of the Software Engineering Research Group (GrIS) at the Instituto de Computación (INCO). Moreno holds an Engineering degree in Computer Science from UdelaR and a Master of Science in Computer Science from the same university.

Leticia Pérez
Assistant Professor
School of Engineering, Universidad de la República
Leticia Pérez is an assistant professor in the Engineering School at the Universidad de la República (UdelaR). She is also a member of the Software Engineering Research Group (GrIS) at the Instituto de Computación (INCO). Pérez holds an Engineering degree in Computer Science and a Master of Science in Computer Science, both of them obtained at the UdelaR.

Diego Vallespir
Assistant Professor
School of Engineering, Universidad de la República
Diego Vallespir is an assistant professor in the Engineering School at the Universidad de la República (UdelaR), director of the Informatics Professional Postgraduate Center at UdelaR, director of the Software Engineering Research Group (GrIS) at UdelaR, and a member of the Organization Committee of the Software and Systems Process Improvement Network in Uruguay (SPIN Uruguay). Vallespir holds an Engineering degree in Computer Science, a Master of Science in Computer Science, and a Doctor of Philosophy in Computer Science, all of them obtained at UdelaR. He has several articles published in international conference proceedings. His main research topics are empirical software engineering, software process, and software testing.

2.6 References

[Basili 1994]
Basili, Victor, Caldiera, Gianluigi, & Rombach, Dieter. "The Goal Question Metric Approach," 528–532. *Encyclopedia of Software Engineering, Vol. 1*. Edited by John J. Marciniak. John Wiley & Sons, 1994.

[Grazioli 2012]
Grazioli, Fernanda & Nichols, William. "A Cross Course Analysis of Product Quality Improvement with PSP," 76–89. *TSP Symposium 2012 Proceedings* (CMU/SEI-2012-SR-015). Software

Engineering Institute, Carnegie Mellon University, 2012. http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=34091

[Hayes 1997]

Hayes, Will & Over, James. *The Personal Software Process: An Empirical Study of the Impact of PSP on Individual Engineers* (CMU/SEI-97-TR-001). Software Engineering Institute, Carnegie Mellon University, 1997. http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=12801

[Juristo 2001]

Juristo, Natalia & Moreno, Ana M. *Basics of Software Engineering Experimentation*. Kluwer Academic Publishers, 2001.

[Rombach 2008]

Rombach, Dieter, Munch, Jurgen, Ocampo, Alexis, Humphrey, Watts S., & Burton, Dan. "Teaching Disciplined Software Development." *Journal of Systems and Software 81*, 5 (May 2008): 747–763.

[Wilcoxon 1945]

Wilcoxon, Frank. "Individual Comparisons by Ranking Methods." *Biometrics Bulletin 1*, 6 (December 1945): 80–83.

3 An Analysis of Student Performance During the Introduction of the PSP: An Empirical Cross-Course Comparison

Fernanda Grazioli, Universidad de la República
William Nichols
Diego Vallespir, Universidad de la República

3.1 Introduction

Almost every new product or system that we use in our daily lives has a software component for its operation. Meanwhile, both the size and complexity of the software increase day by day. In this context, software engineering has need for improved software quality and better cost and schedule management, as well as reduced software development cycle time [Sommerville 2010].

The Team Software Process (TSP) is a software development process for teams that satisfies these needs and that uses the Personal Software Process (PSP) for each team member [Humphrey 2005a, 2006]. The PSP is a defined and measured software process designed to be used by an individual software engineer to address the needs of software businesses by improving the technical practices and individual abilities of software engineers, and by providing a quantitative basis for managing the development process [Humphrey 2005b].

Given that the TSP is a successfully used process and it is qualified as the best software development process for medium- and large-scale projects [Jones 2010], it is important to know whether the processes and the techniques of the PSP lead to development of high-quality products. Therefore, the general goal of this study is to know if the different techniques and phases of the PSP (therefore, the PSP itself) produce positive changes in the aforementioned aspects of software development.

The PSP is taught through a course. Several versions of the course use the same exercises, but introduce process phases and techniques in modified sequences. For an earlier version of the course, several published studies demonstrated improvement in developer performance[2] with process insertion [Hayes 1997; Paulk 2006, 2010; Rombach 2008; Kemerer 2009], but the retrospective analysis left some threats to the validity of these claims. One threat to the validity of the claims of these studies is the confounding of the effect of introducing process phases and techniques insertions with the gaining of domain experience as related programs are developed.

Given this known problem (validity threat to prior experiments in PSP), the main goal of this study is to use the PSP data from the latest two course formats to determine whether the different techniques introduced improve several aspects of developers' performance, or if such improvement is only a consequence of gaining experience in the problem domain. A secondary goal is to document observations and results of the two recent course versions, which do not have yet published works.

Based on the work of Hayes and Rombach [Hayes 1997, Rombach 2008], and continuing our previous study of defect density in unit testing [Grazioli 2012], we decided to evaluate the effects of

[2] The term *performance* covers several aspects, such as improving the quality of the produced product, producing better estimations, and increasing the code production rate, among others. It should not be confused with productivity.

the last two PSP course versions through three hypotheses, focusing on determining the main reason for the improvements and not just evaluating the effect size of the improvements. Therefore, we defined the particular goals of this study as follows:

- Analyze and compare the data collected at the PSP levels in two different courses for the purpose of evaluating performance improvements of engineers with respect to ***yield / production rate / size estimation accuracy*** from the viewpoint of a researcher in the context of the PSP training of engineers in the PSP for Engineers I/II revised course and the training of engineers in the PSP Fundamentals and Advanced course.
- In case of improvements, determine if these are due to the specific techniques introduced or if such improvements are only a consequence of the experience gained in the problem domain.

On the basis of these goals, we tested the following hypotheses:

- As engineers progress through the PSP training, their yield increases significantly. More specifically, the introduction of design review and code review following PSP Level 1 has a significant impact on the value of engineers' yield.
- As engineers progress through the PSP training, there is no real substantive gain or loss in production rate. That is, the number of lines of code designed, written, and tested per hour does not change with a higher PSP level.
- As engineers progress through the PSP training, their size estimates gradually grow closer to the actual size of the program at the end. More specifically, after the introduction of a formal estimation technique for size in PSP Level 1, there is a notable improvement in the accuracy of engineers' size estimates.

3.2 Data Set

We used data from the eight-program course version, PSP for Engineers I and II (PSPI/II), taught between June 2006 and June 2010, and from the seven-program course version, PSP Fundamentals and Advanced (PSP Fund/Adv), taught between December 2007 and September 2010. These courses were taught by the Software Engineering Institute (SEI) at Carnegie Mellon University or by SEI partners, including a number of different instructors in multiple countries.

We analyzed 347 subjects in total, 169 from the PSP Fund/Adv course and 178 from the PSPI/II course. From this we made several cuts and ran data-cleaning algorithms to include only the students who had completed all programming exercises, in order to remove errors and questionable data. We determined other cuts on the data set by performing an analysis and assessment of the data quality based on the data quality theory.

3.3 Statistical Model

In our context, several participants perform the same task (programming) but follow different processes (PSP levels). This is a repeated measures experiment. We want to notice whether there are changes in the individuals' performances when they change the applied process.

To know whether engineers improve their performance during the course, we studied the changes in engineers' data over seven different programming assignments. Rather than analyzing changes in group averages, this study focuses on the average changes of individual engineers. Some engi-

neers performed better than others from the first assignment, and some improved faster than others during the course. To discover the pattern of improvement in the presence of these natural differences between engineers, we used the statistical method known as the repeated measures analysis of variance (ANOVA for repeated measures) [Tabachnick 1989].

The following terms and independent variables must be clear for understanding the analyses:

- Subject – A student who performs a complete PSP course.
- Course Type – Refers to a PSP course version. It can be PSP Fund/Adv or PSPI/II.
- Program Assignment or Program Number – Refers to an exercise that a student has performed during the PSP course. Values range from 1 to 7. Program Assignment 8 of the PSP I/II course version will not be analyzed as there is no way to compare it with another assignment in the PSP Fund/Adv course version.
- PSP Level – Refers to one of the six process levels used to introduce the PSP in these course versions. It can be PSP0, PSP0.1, PSP1, PSP1.1, PSP2, or PSP2.1. Each program assignment has a corresponding PSP level according to the PSP course version. As we want to analyze the introduction of phases and techniques during the courses, we group PSP0 and PSP0.1, we group PSP1.0 and PSP1.1, and we analyze PSP2.0 and PSP2.1 separately.
- Yield = 100 * Defects removed before compile phase / Defects injected before compile phase
- Production Rate = (Actual A&M LOC / Actual Minutes) * 60
- Size Estimation Accuracy = (Estimated LOC – Actual LOC) / Estimated LOC

As it is necessary to understand the followed approach, Table 6 shows which PSP level is applied on each program assignment, for each course version.

Table 6: PSP Levels for Each Program Assignment

Program Assignment	PSP Fund/Adv	PSP I/II
1	PSP 0	PSP 0
2	PSP 1	PSP 0.1
3	PSP 2	PSP 1
4	PSP 2	PSP 1.1
5	PSP 2.1	PSP 2
6	PSP 2.1	PSP 2.1
7	PSP 2.1	PSP 2.1
8	—	PSP 2.1

To analyze whether performance improvements are due to the programming repetition or to the introduction of phases and techniques, we defined and used an indirect statistical method of analysis. This method consisted of three steps in which we examined the relationships between program number, PSP level, course version, and engineers' performance, applying ANOVA.

In the first step, we examined whether are there differences between the two courses by comparing the variable under study for each program assignment (comparing the same program in different courses). When a program yielded no statistical difference, it was discarded. If there are significant differences when there is no PSP level difference within the courses for that program, then the level cannot be the root cause of the differences in the variable under study. But, when the differences are found when there is a level difference for that assignment, then we should

move forward to the second step in order to find if the PSP level could be the root cause of the changes.

We know that in each course, each program assignment is completed following a specific PSP level. In the second step, we looked at each course separately to see whether the differences between the course programs' assignments occurred when the PSP level changed or if the differences occurred even when the PSP level did not change between two assignments. If there are significant changes between programs assignments with the same PSP level, this could indicate that the effects on the dependent variable are due to the repetition of exercises and not to a specific technique introduction. Otherwise, if the significant changes exist only between programs' assignments with different PSP levels, then we must study (in the third step) the behavior of the engineers' performance through the PSP levels, when grouping the program assignments by PSP level.

In the third and last step, we looked at each course separately again to discover whether the differences between the PSP levels occurred when a specific technique that is expected to improve an aspect of the engineers' performance is in fact introduced. If there are significant changes between PSP levels where the technique is introduced, this will show that the technique introduced is the factor affecting the engineers' performance and not the program repetition.

Figure 1 shows a flowchart that represents in a clear graphic way the flow of the third step analysis procedure that we followed for each dependent variable.

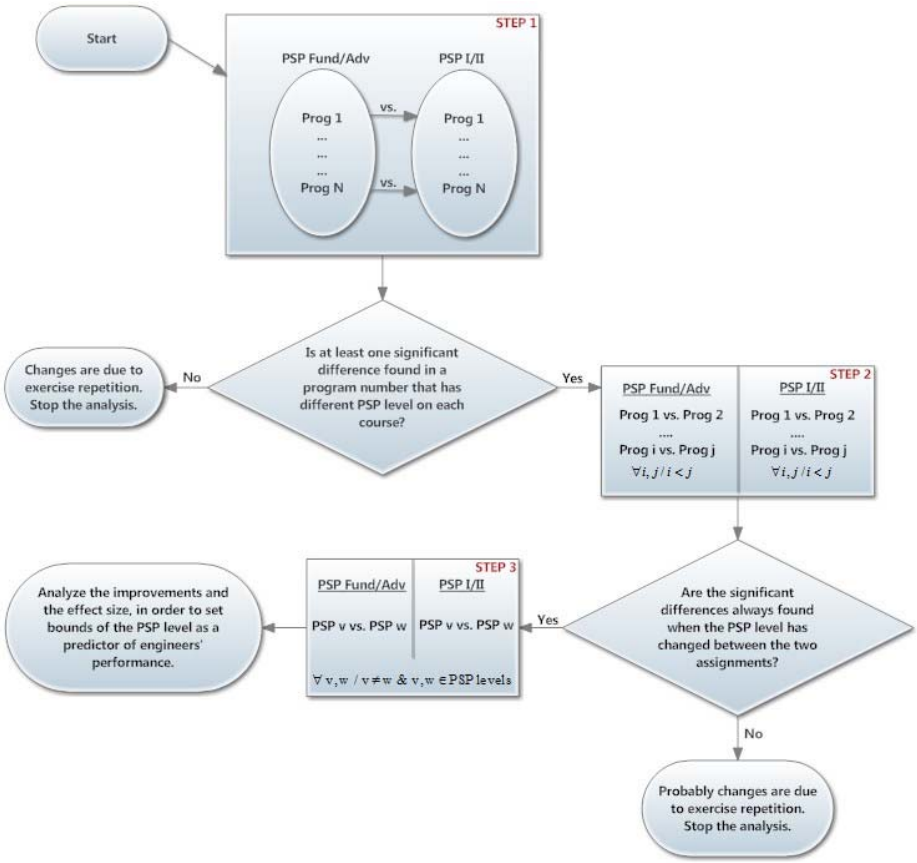

Figure 1: Three-Step Analysis Approach Flowchart

3.4 Results

This section presents a summary of the results obtained for the three hypotheses. We should remember that in following the same approach as a previous study that shared the same main goal, we analyzed performance improvements of engineers with respect to defect density in unit testing and found significant improvement with a mean reduction of a factor of 2.3. That result suggests that improvements in defect density in unit testing are most plausible regarding mastering PSP techniques rather than programming repetition [Grazioli 2012].

3.4.1 Yield

After following the analysis procedure for yield, for each course we found significant difference only between assignments with different PSP levels, and we did not find significant difference in process yield between PSP0 and PSP1. According to the design and code review introduction in PSP Level 2, these improvements were expected. The left plot of Figure 2 shows the estimated marginal means of yield versus program number, for both courses. The graphic shows how the two courses have low yield during assignments with PSP Level 0 or 1, then an important increment on yield after the first PSP2 introduction.

Looking at the two-way ANOVA results of Step 3, in both courses we found significant difference between PSP0 and PSP2, PSP2.1. We also found significant difference between PSP1 and PSP2, PSP2.1. The right plot of Figure 2 shows the 95% confidence intervals of yield for each PSP level, for both courses.

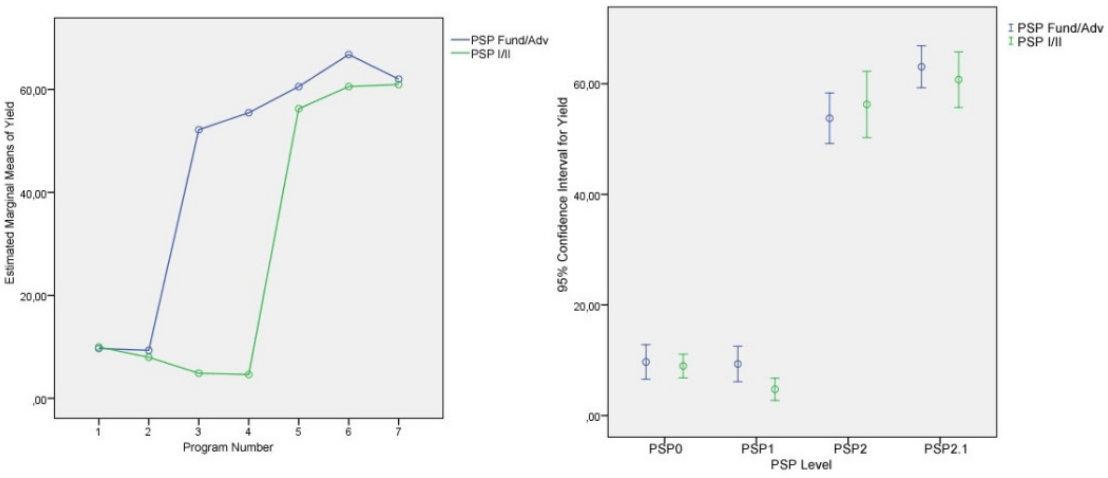

Figure 2: Estimated Marginal Means and 95% Confidence Interval of Yield

Our results show significant improvement in the process yield with a mean increase of a factor of 1.9. Our results also support that design and code review techniques are the main reason for the improvements rather than the learning effect.

3.4.2 Production Rate

After following the analysis procedure for production rate, for each course we found significant difference only between assignments with different PSP levels. There is a deterioration of production rate as engineers move forward in the PSP level. The left plot of Figure 3 shows the estimated

marginal means of production rate versus program number, for both courses. The graphic shows how an engineer's production rate evolves during the complete courses.

Looking at the two-way ANOVA results of Step 3 without course discrimination, we find that there is significant difference between each PSP level compared in pairs. The right plot of Figure 3 shows the 95% confidence intervals of production rate for each PSP level, considering both courses together.

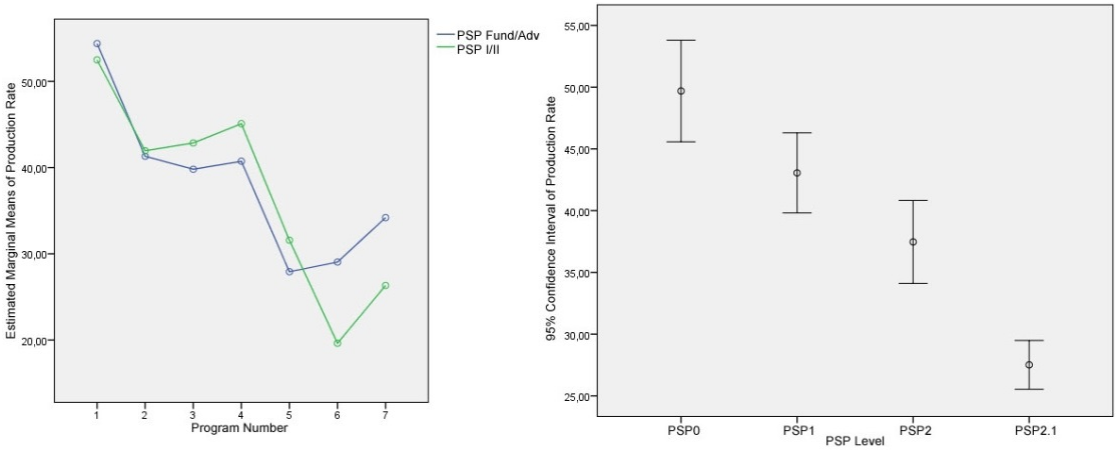

Figure 3: Estimated Marginal Means and 95% Confidence Interval of Production Rate

Regarding production rate, we found a mean reduction of a factor of 0.7. In our study, both courses appear to be effective in demonstrating that the increments in the amount of design documentation and data tracking proposed by the PSP deteriorates the production rate during the PSP course. Our result differs from previous studies of the 10-program course version, some of which find improvements and others find no real gain or loss [Hayes 1997, Rombach 2008, Paulk 2010].

3.4.3 Size Estimation Accuracy

After following the analysis procedure for size estimation accuracy (SEA), for each course we found significant difference only between assignments with different PSP level. According to the PROBE technique introduced, which is based on engineer historical data, these improvements were expected. The ANOVA works as one would expect when the trend is always in the same direction, but not if some are overestimating and others underestimating. So it is necessary to define a new dependent variable that is the absolute value of size estimation accuracy. The left plot of Figure 4 shows the estimated marginal means of abs(SEA) vs. program number, for both courses. The graphic shows how the two courses perform differently, even if we cannot see the specific effect of the introduction of the size estimation technique in PSP Fund/Adv course. Remember that in PSP Fund/Adv we cannot compare PSP1 to something previous, as there is not a previous assignment with a size estimation calculus done by the student. We can see the evolution of the rest of the course, but not specifically the PSP1 introduction. In this graphic of the estimated marginal means, the size estimation accuracy appears to be more consistent by the end of the courses.

Looking at the two-way ANOVA results of Step 3, in the PSP Fund/Adv course we found that there is significant difference between PSP1 and PSP2.1. But as we do not have assignments with PSP0, we cannot study the effects of the introduction of PSP1. Regarding the two-way ANOVA

results for the PSP I/II course, we found that there is significant difference between PSP1 and PSP2, PSP2.1. The middle plot of Figure 4 shows the 95% confidence intervals of absolute value of size estimation accuracy for each PSP level, for both courses.

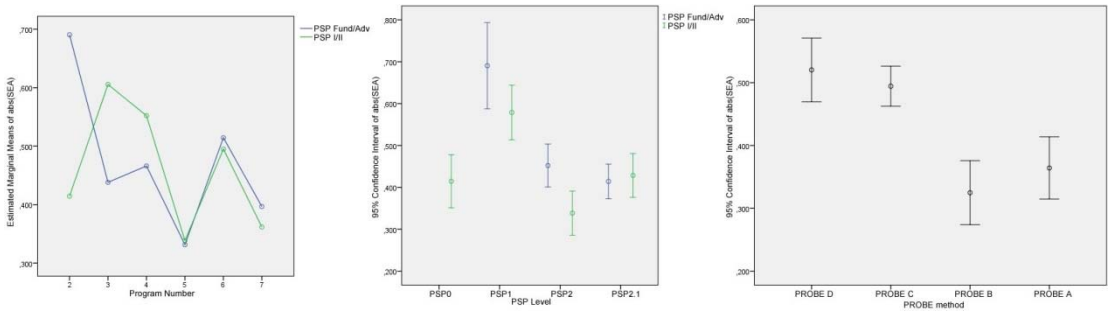

Figure 4: Estimated Marginal Means and 95% Confidence Interval of abs(Size Estimation Accuracy)

With these results, we do not really see directly that the introduction of the estimation technique improves the size estimation accuracy, because PSP2 and PSP2.1 introduce the design and code reviews and design templates, not the estimation techniques.

To get a clearer idea of the relationship between the estimation techniques introduction and the size estimation accuracy, we propose to analyze the data in a different way. We look not at the PSP level but at the specific PROBE method that is applied in each assignment. To do this, we execute again the third step of the indirect analysis method, but this time reorganizing the student data by PROBE method (A, B, C, or D). We found that there is a significant difference between PROBE A and PROBE C, D as well as a significant difference between PROBE B and PROBE C, D. The right plot of Figure 4 shows the 95% confidence intervals of the absolute value of size estimation accuracy for each PROBE method, for both courses together.

With the available data, it is very difficult to separate the possible causes of size estimation improvement: the introduction of the formal estimation technique and the experience in the problem domain. With the presented results, it is clear that data shows and supports the hypothesis that the engineer's size estimates improves. But we cannot determine if the introduction of the size estimation technique is the main reason of that improvement because

- PROBE A and B cannot be applied until there are a minimum of three historic points
- it takes accumulated data for the size estimation technique to become effective
- the estimation process takes multiple repetitions to stabilize
- the estimation technique is not just one technique; in fact, it is a package of three different methods, and the student varies its application during the course
- the PSP level introduction on the last two courses is not the optimal to study this hypothesis

Regarding size estimation accuracy results, we found significant improvement with a mean reduction of a factor of 2.6. For this particular dimension, we were not able to discard the domain learning effect as the root cause of the improvements, as the estimation technique introduced in the PSP courses is based on historical data and needs repetition.

3.5 Threats to Validity and Limitations

To apply the repeated measures ANOVA, some assumptions must be met: subjects must be randomly selected, observations on these subjects are independent, and the dependent variables must be normally distributed and have equality of variances.

The researchers did not select the subjects; the students selected the course, and there is no precondition to do one course or another. So the random selection seems to be satisfied. On the other hand, some other biasing factor remains, because the students who took the PSP Advanced course are more likely to go on to instruction or teaching. This group might respond better to the PSP instruction, and this could be seen as a threat to validity.

As to other potential factors, a completely independent observation of the subject is almost impossible to achieve as classes work together with the same instructor and thus they do not only depend on the sole quality of the instructions. Given the quite large set of data, the large number of different instructors, and numerous different classes, this assumption should, however, not be completely violated.

The analysis of the collected data showed that the requirement for normal distribution of the dependent variables is not fully met. However, the data are mounded without severe outliers. Nevertheless, different transformation techniques were applied to better meet this assumption for each hypothesis to reach a more normal distribution variable. Fortunately, an ANOVA is not very sensitive to moderate deviations from normality; simulation studies, using a variety of non-normal distributions, have shown that the false-positive rate is not affected very much by this violation of the assumption [Glass 1972, Harwell 1992, Lix 1996].

The PSP training aims at providing engineers with techniques to improve their daily work with seven or eight assignments, depending on the course version. The data is collected within a class setup where the attendees can concentrate on the assignment and are not distracted by colleagues, working on multiple projects, and so forth. The investigation thus can only show the improvements achieved during the duration of the class.

A general translation of the achieved improvement effects to generally improved workplace performance must, however, be made very carefully. The results show trends that can be interpreted to mean that the trend might continue and finally lead to the assumed results. It is also not directly possible to conclude that the results are immediately valid for large-scale projects, when the engineers are working in multiple project teams, and the project is executed over a long time span.

3.6 Conclusions

The analyses executed in this work substantiate that trends in personal performance observed during PSP application are significant, and that the observed improvements or deterioration represent real change in individual performance, not in the average performance of the group.

Because of our approach, we are able to suggest that the PSP is the root cause of the improvements rather than the domain learning effect in process yield and in defect density in unit testing. Since PSP level changes so rapidly in the PSP Fundamentals and Advanced course and in the PSP I/II revised course, the program number and the PSP process level are tightly correlated in a way that makes separating the effects difficult. This is one of the reasons why we were not able to re-

ject the learning effect in the other two hypotheses. However, the results of our analysis related to these hypotheses lead us to think that the process phases and the introduced techniques are probably one of the main reasons for the changes, so further research and experimentation are necessary to confirm it.

With our results, we show that the use of PSP produces positive changes regarding the improvement quality of the software product, which is one of the major needs of software development. Given the size and complexity of modern software projects, success requires that all individuals produce high-quality software products with predictable cost and schedule. It is, therefore, essential to base organizational processes on practices that work at an individual level and satisfy these needs. This work suggests that PSP has demonstrated the capability to address these needs.

3.7 Author Biographies

Fernanda Grazioli
Research Assistant
School of Engineering, Universidad de la República
Fernanda Grazioli is a research assistant at the Engineering School at the Universidad de la República (UdelaR). She is also a member of the Software Engineering Research Group (GrIS) at the Instituto de Computación (INCO). Grazioli holds an Engineering degree in Computer Science from UdelaR and a Master of Science in Computer Science from the same University.

William Nichols
Bill Nichols joined the Software Engineering Institute (SEI) in 2006 as a senior member of the technical staff and serves as a PSP Instructor and TSP Coach with the Team Software Process (TSP) Initiative. Prior to joining the SEI, Nichols led a software development team at the Bettis Laboratory near Pittsburgh, Pennsylvania, where he had been developing and maintaining nuclear engineering and scientific software for 14 years. His publications include the interaction patterns on software development teams, design and performance of a physics data acquisition system, analysis and results from a particle physics experiment, and algorithm development for use in neutron diffusion programs. He has a doctorate in physics from Carnegie Mellon University.

Diego Vallespir
Assistant Professor
School of Engineering, Universidad de la República
Diego Vallespir is an assistant professor in the Engineering School at the Universidad de la República (UdelaR), director of the Informatics Professional Postgraduate Center at UdelaR, director of the Software Engineering Research Group (GrIS) at UdelaR, and member of the Organization Committee of the Software and Systems Process Improvement Network in Uruguay (SPIN Uruguay). Vallespir holds an Engineering degree in Computer Science, a Master of Science in Computer Science, and a Doctor of Philosophy in Computer Science, all of them obtained at UdelaR. He has several articles published in international conferences. His main research topics are empirical software engineering, software process, and software testing.

3.8 References

[Glass 1972]
Glass, G. V., Peckham, P. D., & Sanders, J. R. "Consequences of Failure to Meet Assumptions Underlying Effects Analyses of Variance and Covariance." *Review of Educational Research 42*, 3 (Summer 1972): 237–288.

[Grazioli 2012]
Grazioli, F. & W. Nichols, W. "A Cross Course Analysis of Product Quality Improvement with PSP," 76–89. *Proceedings of the TSP Symposium 2012: Delivering Agility with Discipline* (CMU/SEI-2012-SR-015). St. Petersburg, FL, September 2012. Software Engineering Institute, Carnegie Mellon University, 2012. http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=34091

[Harwell 1992]
Harwell, M. R., Rubinstein, E. N., Hayes, W. S., & Olds, C. C. "Summarizing Monte Carlo Results in Methodological Research: The One- and Two-Factor Fixed Effects ANOVA Cases," *Journal of Educational Statistics 17*, 4 (Winter 1992): 315–339.

[Hayes 1997]
Hayes W. & Over, J. W. *The Personal Software Process (PSP): An Empirical Study of the Impact of PSP on Individual Engineers* (CMU/SEI-97-TR-001). Software Engineering Institute, Carnegie Mellon University, 1997. http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=12801

[Humphrey 2005a]
Humphrey, W. S. *TSP: Leading a Development Team*. Addison-Wesley, 2005.

[Humphrey 2005b]
Humphrey, W. S. *PSP: A Self-Improvement Process for Software Engineers.* Addison-Wesley Professional, 2005.

[Humphrey 2006]
Humphrey, W. S. *TSP: Coaching Development Teams*. Addison-Wesley, 2006.

[Jones 2010]
Jones, C. *Software Engineering Best Practices: Lessons from Successful Projects in the Top Companies.* McGraw Hill Professional, 2010.

[Kemerer 2009]
Kemerer, C. & Paulk, M. C. "The Impact of Design and Code Reviews on Software Quality: An Empirical Study Based on PSP Data." *IEEE Transactions on Software Engineering 35*, 4 (July–August 2009): 534–550.

[Lix 1996]
Lix, L. M., Keselman, J. C., & Keselman, H. J. "Consequences of Assumption Violations Revisited: A Quantitative Revew of Alternatives to the One-Way Analysis of Variance F Test." *Review of Educational Research 66*, 4 (Winter 1996): 579–619.

[Paulk 2006]
Paulk, M. C. “Factors Affecting Personal Software Quality.” *Cross-Talk: The Journal of Defense Software Engineering 19*, 3 (March 2006): 9–13.

[Paulk 2010]
Paulk, M. C. “The Impact of Process Discipline on Personal Software Quality and Productivity.” *ASQ Software Quality Professional 12*, 2 (March 2010): 15–19.

[Rombach 2008]
Rombach, H. D., Münch, J., Ocampo, A., Humphrey W. S., & Burton, D. “Teaching Disciplined Software Development.” *Journal of Systems and Software 81*, 5 (May 2008): 747–763.

[Sommerville 2010]
Sommerville, I. *Software Engineering,* 9th ed. Addison-Wesley, 2010.

[Tabachnick 1989]
Tabachnick, B. G. & Fidell, L. S. *Using Multivariate Statistics*. Harper Collins, 1989.

4 Incorporating Some PSP Practices into Introductory Programming Courses: A Case Study in Universidad del Quindío

Sergio Cardona, Universidad del Quindío, Colombia
Rafael Rincón, Universidad EAFIT, Colombia
Diego Vallespir, Universidad de la República, Uruguay

4.1 Introduction

The sustainability of the software industry depends largely on the training of highly skilled professionals and their ability to develop quality software. The incorporation of the appropriate practices for software development improves the capacity and productivity of information technology organizations. Unfortunately, this could mean high investments of time and training for organizations.

We understand that the academy has committed to training professionals with self-management and administration skills in software processes that can be defined, measured, and controlled. The academic curricula must consider the skill development and technical capacity for the construction of quality software. In this regard, some universities have used quality-oriented process models, and students apply the best software development processes for management, cost, time, removal, estimation of size, management of standards, and prevention of flaws [Cardona 2012a].

The Personal Software Process (PSP) is a software development process for an individual [Humphrey 1995]. This process supports the software engineer for the construction of quality products. The PSP is also a training complement that aims for quality culture in software development, and in the curricula from some universities, it is offered as an elective course. In classroom experience reports, when the PSP is used in the first programming course, the complexity of its implementation is identified because the students not only learn to program but also learn the good practices of software development that the PSP proposes [Bermón 2009].

This article presents the results of research that applied a learning strategy to an experimental group, implementing some PSP practices in a programming course in first semester in the second half of 2012. Some PSP practices were introduced with the idea that the students would apply individual techniques for the development of skills in aspects like planning, time estimation, and management of software flaws. The results showed that the students meaningfully adopted practices associated with time and flaw management.

Initially, the related works along with the conceptual support for the development of this research are presented. Then, the methodology defined for its development is also presented. Following that is the learning strategy design and, finally, the results and conclusions.

4.2 Related Works

Since Watts Humphrey presented the PSP in his book *A Discipline for Software Engineering*, diverse investigations about the impact that the use of the PSP generates in undergraduate and graduate courses in universities have been carried out [Towhidnejad 1997, Hayes 1998, Prechel 2001, Abrahamsson 2002, Börstler 2002, Runeson 2003, Rong 2012]. The PSP has been also used for

experimenting in software engineering courses [Venkatasubramanian 2001, Honig 2008]. Likewise, there are reports of learned lessons from the PSP implementation on the academic sector with software industry support [El Eman 1996, Rincón 2010]. Also, there are academic experiences related to TSP [Bayona 2008, Honig 2008, Rombach 2008].

The analysis of the related works resumes what is proposed by Börstler and colleagues [Börstler 2002], and three primary factors, which influence the teaching of PSP, stand out: the work environment, the coverage level, and the support tools. The work environment refers to the target audience, the course level, and the subject content. The coverage level is associated with the PSP practices applied. The support tools are related to the support means for recording every single activity proposed by the PSP. This paper contributes a new analysis factor associated with or without the application of a learning strategy. Table 7 presents the results obtained from the PSP implementation in different universities worldwide.

Table 7: Academic Experiences of the PSP

University	Target Students	Level of Coverage	PSP Support Tools	Learning Strategy
Lund [Runeson 2003]	Undergraduate and graduate	Full PSP[3]	Spreadsheets	N/A
Zagreb [Car 2003]	Undergraduate	PSP-Lite[4]	Local development	N/A
Purdue [Lisack 2000]	Undergraduate	PSP-Lite	Spreadsheets	N/A
Carlos III [Bermón 2009]	Undergraduate	PSP-Lite	Student workbook	N/A
Umea [Börstler 2002]	Undergraduate	PSP-Lite	Local development	N/A
Utah [Börstler 2002]	Undergraduate and graduate	Full PSP	Local development	N/A

Based on Table 7, it can be established that every reported experience has a factor relevant to the context and the training interests of its students. Given the space limitation for the article, a detailed analysis of every academic experience in the implementation of the PSP has not been done.

4.3 Methodology

The development of this first experiment with the practices of PSP in a Computer Programming course is articulated under the proposal developed by Cardona and Rincón, who present a strategy for implementing PSP practices in all the area courses of the computer programming curriculum in the Computer Engineering Program of the University of Quindío [Cardona 2012b]. A proposal of horizontal incorporation that is applied progressively through the different courses in the academic semesters of the curriculum is presented.

The following objectives for the development of this experimental research are defined:

- to analyze the state of the art and the most significant experience results worldwide of the use of the PSP in academia and to identify their impact in the student skill development process in software engineering, so that these can help as a reference for a theoretical support of the research

[3] It refers to the implementation of the entire body of knowledge of PSP.

[4] It refers to a simplified or an adapted version of PSP.

- to design the scenarios, activities, and learning resources that allow, by means of a training strategy, the appropriation and application of individual practices of the PSP
- to conduct a pilot test with the Programming course students, in order to verify and assess that the strategy contributes to the development of individual practices of software development of students

The research was piloted in the classroom. The populations under study were two groups of a first course in Computer Programming of the Computer Engineering undergraduate program at the Universidad Del Quindío. The methodology is shown in Figure 5.

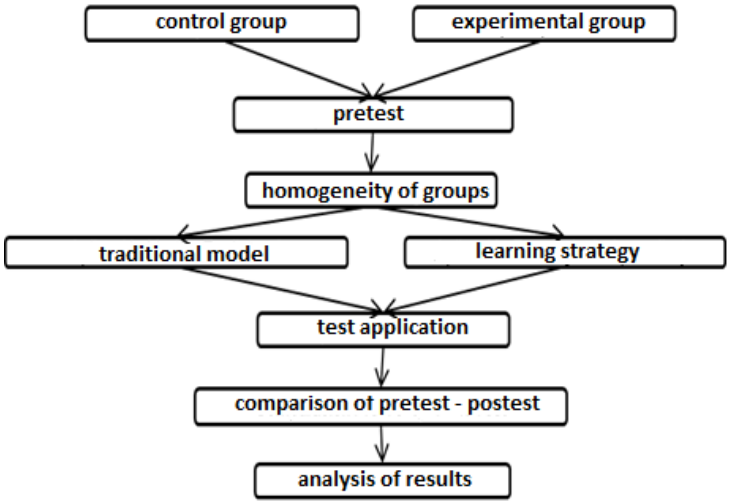

Figure 5: Methodology for the Research

4.3.1 Pre-test

The initial diagnosis applies an instrument with nine questions (Table 8) with options (Never – Sometimes – Always). The questions were designed to elicit the level of adoption of some individual practices for software development in students. The number of students in the control and experimental groups who answered the survey was 31 and 35, respectively.

Table 8: Questions and Categories

Number	Questions Homogeneidad de varianzas (Levene) Distribución normal de residuos Shapiro–Wilk	Category
1	Do you record the time spent during the programming activity?	Time management
2	Do you record the interruption time during the programming activity?	
3	Do you record the flaws that emerge in the making of a programming activity?	Handling and management of flaws
4	Do you understand the encoding flaws generated during programming?	
5	Do you apply some methodology to solve flaws in the codification process?	
6	Do you take into account encoding standards when programming?	Size of product
7	Do you estimate the number of code lines needed to build a program?	
8	Do you plan activities to perform a programming job?	Product planning
9	Do you apply the stages of the development process to build a program?	

With the pre-test exam applied to both groups, the level of homogeneity in each question was analyzed, for both the experimental and control groups. To check the homogeneity of the groups, an analysis of variance (ANOVA) was run, whose response variable is the qualification of the question, and the factor is the group with the control and experimental levels. For each question the assumptions of randomness, the homogeneity of variance, and the normal distribution of residuals were applied. When these assumptions were not met, then the Kruskay and Wallis nonparametric test was applied. Table 9 shows the results of the statistical analysis of each question.

Table 9: Homogeneity Analysis per Question

Question	*p* Value	Variance Homogeneity (Levene)	Shapiro–Wilk Normal Distribution of Residuals	Kruskal–Wallis Nonparametric Test
1	0.1409	0.140908	4.35409^{-15}	0.1848
2	0.5739	0.573945	3.54809^{-14}	0.569942
3	0.2237	0.413037	7.81931^{-10}	0.162059
4	0.2356	0.0750583	4.44089^{-16}	0.204958
5	0.6374	0.808796	1.11022^{-16}	0.712758
6	0.1495	0.149459	1.26715^{-10}	0.1848
7	0.4727	0.0511165	1.057^{-9}	0.451819
8	0.1023	0.617618	3.42777^{-7}	0.132956
9	0.4686	0.151367	1.7582^{-10}	0.48251

Based on these results, both the control and the experimental groups are homogeneous for the nine questions defined in the instrument, and we decided to continue the research methodology.

4.3.2 Learning Strategy

The teacher responsible for the learning strategy was trained in the PSP Fundamental Training course. Also, his master's thesis was aimed at a training proposal to apply PSP and TSP practices along a Computer Engineering curriculum. The teachers responsible for the programming courses had received training in PSP/TSP quality practices promoted by the Ministry of Communications and Technologies of the Colombian government.

The learning strategy was conducted with 23 students from the experimental group during 10 weeks of the academic semester. Parallel to the development of the subject content, the fundamental concepts of PSP0, PSP0.1, and PSP1 levels were incorporated. Six programming exercises were proposed, which were completed directly in the laboratory course, under the teacher's monitoring. For the PSP0.1 and PSP0 levels, the first four exercises were completed, and the remaining two were for the PSP1 level.

The students used the process script, the planning script, and the plan summary of the project for the PSP0, PSP0.1, and PSP1 levels. For the deliverables, the students used the time recording log and the defect registry log for the PSP0, PSP0.1, and PSP1 levels. The coding format standard was used for the PSP0.1 and PSP1 levels. For the course final project, a test report template was required. The recording of each activity was performed on templates designed for that purpose, and the feedback on the results was discussed in the class that followed, highlighting the importance of the proposed activities.

The population under study consisted of freshmen students from the Computer Engineering program. Since many of the students did not have the necessary skills in the use of some tools, a

learning strategy was initially adapted that integrated the concept of the Engineering Notebook that Humphrey proposes [Humphrey 1997]. These notebooks allowed the manual recording of activities, time, and defects by the students. Subsequently, these notebooks were implemented in programmed electronic sheets that bore the recording of each of the activities proposed in the strategy.

The program activities of the course were conducted with the Java language. The development environment used was Eclipse Galileo. The PSP Student Workbook tool was used to collect the data of the process after the eighth week of the course.

A virtual environment on an LMS (Learning Management System) technology platform was designed as a support resource to the learning scenarios defined in the training strategy. There were virtual discussion forums and interactive group activities that allowed the exchange of experiences by the students.

4.3.3 Thematic Structure of the Course

The course structure is defined by thematic units required in a first programming course. The fundamental concepts of the PSP0, PSP0.1, and PSP1 levels were incorporated progressively. Table 10 shows the thematic content and the PSP themes that were given in the course.

Table 10: Thematic Content and PSP Themes

Unit	Thematic Content	PSP Topics
Java Programming Language	Variables, operators, and expressions Primitive data types Objects concepts	• Software quality concepts • Software development process • Current process development
Conditional Programming	Simple decisions (if, if-else) Nested decisions Multiple decisions (switch)	Personal process reference • Introduction to PSP • Introduction to PSP0 • Time planning
Methods	Methods concepts Methods that return value Methods that do not return value Parameter passing	Reference personal process • Time and control management PSP0 • Time and flaws recording • Types of flaws standards
Iterative Programming	Counters and accumulators Cycle conditioned at the end (do-while) and conditioned at the beginning (while, for)	Reference personal process PSP0.1 • Size planning and measuring • Encoding standards
Arrangements	Operations with arrangements Dimensional arrangements Management methods	Reference personal process PSP0.1 • Encoding standards • Process Improvement Proposal (PIP) Personal project management PSP1

The PSP themes were oriented only in the experimental group. For the PSP0-level practices, a teaching guide with the theoretical foundations necessary for learning and implementing the following practices was designed:

- time recording for the completion of the project
- flaw recording and its types
- summary of the project plan
- standards to document and report the types of flaws

In the PSP0.1, a guide was created to aid students in learning to perform the count of code lines (LOC) of their programs, as well in documenting the activities of the development process in order to identify opportunities for improvement in students' work. The elements taken into account for this level were

- definition of a standard for code line counting and an encoding standard during product construction process
- documentation of the Process Improvement Proposal (PIP)

For the PSP1, a guide was also designed that explained the following, using examples: how the template must be filled out for the test report and the estimate for the size of the product.

The traditional methodology was applied to the control group. A teacher responsible for the course was in charge of guiding the five thematic units according to predefined objectives. The methodology focuses on the development of basic programming skills; for this, the students conducted individual and group exercises, and the concept of quality focused on testing their finished products only. The subjects taught in the control group corresponded to those defined in the course micro-curriculum, and the topics related to software quality were not incorporated—unlike in the experimental group, where topics and activities related to PSP were incorporated.

4.3.4 Design of the Learning Strategy

For each thematic unit of the course, the learning scenarios that define the necessary theoretical elements, the work methodology, and the activities undertaken by the students were designed. Table 11 shows the description of the Iterative Programming thematic unit, and similar descriptions were done for the rest of the course units.

Table 11: Thematic Structure of the Course

Unit	Methodology	Activities
Iterative Programming	The teacher presents the fundamental concepts of PSP, the process script, time control, and recording in each phase of the process. He will explain the time log template, which details the actual working time and the interruptions. He will explain to students how to perform the estimation of time for their work, and a series of suggestions to manage time when performing a programming job.	The student will read articles about the fundamental concepts of PSP0 and PSP0.1. In each programming task, the student must use the process script, and the teacher will assign the exercises 1A, 2A, 3A, and 4th, so students develop the proposed programs. Each programming task requires the delivery of the time template. Based on the results delivered by the students, the teacher will conduct a performance analysis of the group works.

For each activity, an evaluation plan was defined based on criteria that take into account the following aspects:

- observation of attitudes and skills that students are developing
- students' response in facing the questions related to the individual development
- monitoring the development of practices that the students do in the lab
- monitoring the tasks that students do during their independent work
- conducting individual assessments

These elements of practice development will have a summative evaluation in a range from 0 to 5.

4.4 Results

To determine whether the intervention with the PSP practices in the experimental group was successful, it was verified in the post-test whether, in each of the questions, the ownership of homogeneity with the control group was retained.

For the analysis of the final results of the learning strategy, only those students who participated in 100% of the proposed activities in the course were taken into account. Also, the dropout factor associated with academic performance and personal difficulties of some students influenced the decrease in the population under study, so that at the end of the course the student group was reduced from 35 to 23.

4.4.1 Post-test

The results obtained in the post-test show that the property of homogeneity of the groups is preserved for Questions 3, 6, 7, 8, and 9, so the learning strategy for the categories of product size and product planning did not have a significant impact within the individual practices of software development.

For Questions 1, 2, 4, and 5, the obtained results show that the homogeneity property of the groups is not preserved; therefore, for the categories of time management and flaw management, the learning strategy was successful. For example, Figure 6 shows that for Question 1, the experimental group applies this PSP practice more than the control group.

Figure 6: Question 1, Time Control for Post-test

4.4.2 Analysis of the Results

We compared the answers of the pre-test and post-test of the students to construct a "result" variable. Thus, if the post-test grade is higher than the pre-test grade, the variable takes the value of 1. If the grade is lower or equal in the post-test, the variable takes the value of 0. If the pretest and post-test graded the answer always with (3), the variable takes the value of 1. Thus, the result variable has only two possible values: 1 and 0; therefore, it is a discrete variable with Bernoulli distribution and $p = 0.5$ because it uses the criterion that at least 50% of students will improve from the pre-test to the post-test. The answers with value of 1 were added, and the variable "number of students who improved with the intervention" was obtained. Due to the sum of variables with Bernoulli distribution, it corresponds to a variable with binomial distribution with $n = 23$ (number of students from the experimental group). The probability $p = 0.5$ indicates that at least half of the students improved with the intervention strategy. Then, a system of hypotheses arose that allowed selecting those questions where students improve their practices. For the experiment, a probability for error of 4.7% was established for characterizing the question in the intervention as successful, which is equivalent to saying that the results have a confidence level of 95.3%.

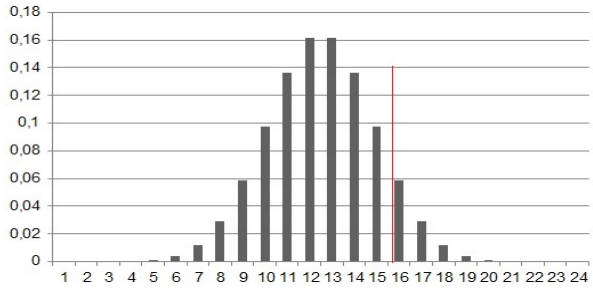

Figure 7: Binomial Distribution for Post-test

Figure 7 shows the binomial distribution for the 23 students from the experimental group with p = 0.5. Those questions where 16 students or more improve with the intervention are the ones that allow characterizing it as successful.

4.4.3 Analysis of the Results from the Experimental Group

The quantitative results of the experimental group in the pre-test and the post-test show a significant improvement in the nine questions applied to students. For example, in Question 2 of the pre-test, related to the interruptions recording practice, 91% of students never apply it, and 9% apply it sometimes. The same question for the post-test shows that only 13% of students never apply it, 74% sometimes do, and 13% always do. Figure 8 shows the frequency of answers for Questions 2 and 3.

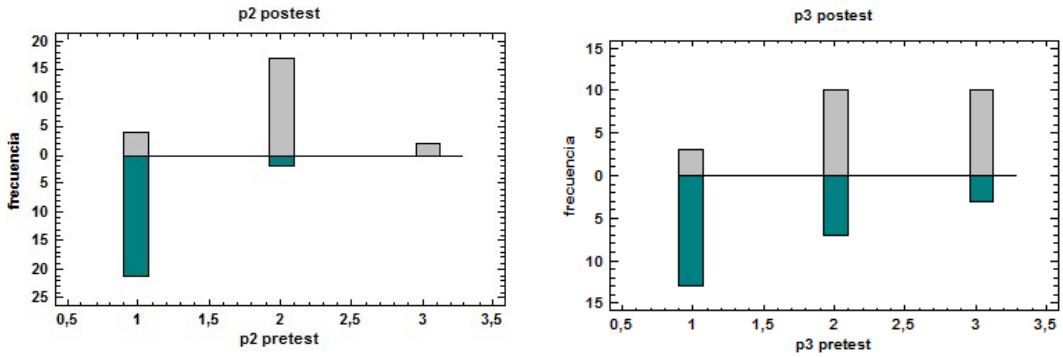

Figure 8: Pre-test and Post-test Results of the Experimental Group

For Question 4 in the post-test, 52% of students in the pre-test answered that they always manage the flaws introduced during their individual work in software development. In Question 5 on the post-test, 74% of students always apply a methodology for the solution of flaws. In both questions, it is evidence of an improvement in the outcomes.

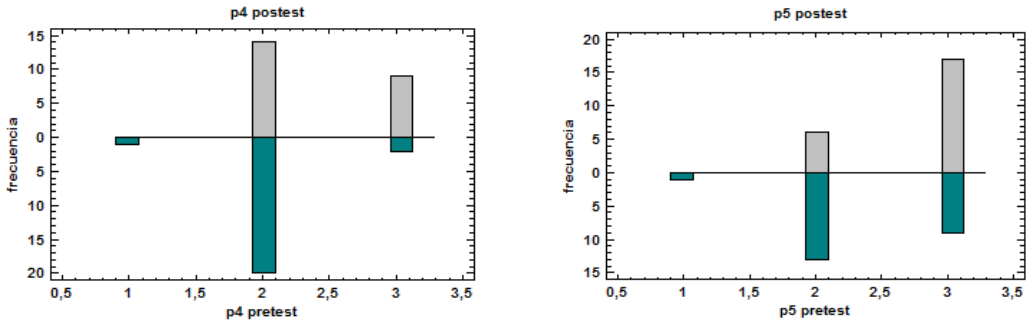

Figure 9: Pre-test and Post-test Results of the Experimental Group

Our results show that the experimental group improved on the post-test compared to the pre-test in every question (see Figure 9).

The intervention in the experimental group shows that each proposed activity imposes on the students an extra effort because they must fill out the formats established, and additionally they submit tasks more formally relative to the control group. One of the most significant difficulties during the intervention with students lies in the processing of formats. Many of them do not complete them fully, so the formats are not filled out correctly.

4.5 Conclusions

The development of this work presented a number of challenges associated with learning the software process, including the ability of students to recognize the value of a discipline applied to the software process (an issue that they have not experienced in early stages yet) and a forced introspection—to learn how software is developed and to understand their individual development habits and the practices needed to improve them. It was also necessary to consider some theories about teaching strategies, which, in our particular context, involved the incorporation of ideas about how to present current practices for students to learn. The most frequent difficulties and mistakes of students were identified, and they were encouraged to reflect high quality in their work.

Considering that PSP involves a rigorous process of gathering information, the students initially perceived it as a filling out forms that involved an additional consumption of time for the development of their work, and they did not understand its added value in the programming learning process. However, since the practices were incorporated gradually during the course, they became a habit that was reinforced by the continuing and ongoing feedback on the individual performance of the students by the teacher. The data collected during the programming process showed that the time log format was very consistent since this activity was incorporated from the beginning of the semester, and its way of measurement is simple. It was difficult recording the defect data during the first six weeks of the semester since the students did not identify the type of defect correctly, and the trend was always to locate defects in the same two or three categories. As for the estimates, they showed an improvement as the semester passed because the students gradually better understood the concepts of baseline and code reuse.

The application of the teaching strategy in the experimental group was successful in five of the nine criteria considered in the instrument applied to students. The conceptual and practical appro-

priation is highlighted in areas such as administration and time management, and the operation and management of defects. As to the estimates of product size, individual work, project planning, and teamwork, no favorable results were obtained in the post-test.

Based on the obtained results, we found that the incorporation of some PSP practices by students of the experimental course have been successful regarding the adoption of the practices associated with time management and recording, and the management and recording of flaws. The development of this work showed a number of challenges because we found that the success of these experiences is associated with the maturity of the students, and to the extent that they recognize the value of an applied discipline to a programming process.

The academic environment also requires political will and commitment from the academic directors since the teachers, who teach the courses related to PSP practices, must spend a great deal of time to give immediate feedback on the work and exercises of the students, conduct permanent support, and teach the topics and concepts related to PSP. This academic strategy becomes complex because teaching and taking courses related to PSP practices require a greater dedication by the teacher and the student than does a regular course.

4.6 Author Biographies

Sergio Cardona
Associate Professor
School of Engineering, Universidad del Quindío, Armenia, Colombia
Sergio Cardona is an associate professor in the Computer Science Department at the Engineering School at Universidad del Quindío (UQ). He is a member of the Research Group SINFOCI at UQ. Cardona holds an Engineering degree in Computer Science from Universidad del Valle, Cali. He has a Master of Science in Computer Science obtained at Universidad EAFIT, Medellín. Currently, he is a PhD student in Information Technologies at Universidad Pontificia Bolivariana, Medellín. He has several articles published in national and international conferences. His main research topics are software quality and empirical software engineering.

Rafael Rincon
Assistant Professor
School of Engineering, Universidad EAFIT, Medellin, Colombia
Rafael Rincon is an assistant professor in the Computer Science Department at the Engineering School at the Universidad EAFIT, Medellin, Colombia; a consultant and researcher on software quality; and an undergraduate and graduate teacher. Rincon holds a degree in Mathematics from the Universidad de Antioquia; a Master of Science in Quality Systems and Productivity obtained at Tecnológico de Monterrey, Mexico; and a Master of Science in Applied Mathematics from the University EAFIT. He has several articles published in national and international conferences. His main research topics are software quality, process improvement, and management and technological innovation.

Diego Vallespir
Assistant Professor
School of Engineering, Universidad de la República
Diego Vallespir is an assistant professor in the Engineering School at the Universidad de la República (UdelaR), director of the Informatics Professional Postgraduate Center at UdelaR, di-

rector of the Software Engineering Research Group (GrIS) at UdelaR, and a member of the Organization Committee of the Software and Systems Process Improvement Network in Uruguay (SPIN Uruguay). Vallespir holds an Engineering degree in Computer Science, a Master of Science in Computer Science, and a Doctor of Philosophy in Computer Science, all of them obtained at UdelaR. He has several articles published in international conferences. His main research topics are empirical software engineering, software process, and software testing.

4.7 References

[Abrahamsson 2002]
Abrahamsson, P., & Kautz, K. "Personal Software Process: Classroom Experiences from Finland." *Lecture Notes in Computer Science 2349* (2002): 175–185.

[Bayona 2008]
Bayona, S., Calvo, J. A., Gonzalo, C., & San Feliu, T. "Teaching Team Software Process in Graduate Courses to Increase Productivity and Improve Software Quality," 440–446. In *32nd Annual IEEE International Computer Software and Applications Conference*. Turku, Finland, July–August, 2008. IEEE, 2008. doi:10.1109/COMPSAC.2008.135

[Bermón 2009]
Bermón, L., Fernandez, A., Sanchez, M., Javier, G., & Seco, A. "Experiencias Docentes en la Aplicación del Proceso Software Personal en Primero de Grado de Ingeniería Informática, " 107–114. In *Fomento e Innovación con Nuevas Tecnologías en la Docencia de la Ingeniería*. IEEE, 2009.

[Börstler 2002]
Börstler, J., Carrington, D., Hislop, G. W., Lisack, S., Olson, K., & Williams, L. "Teaching PSP: Challenges and Lessons Learned." *IEEE Software 19*, 5 (September–October 2002): 42–48.

[Car 2003]
Car, Z. "A Method for Teaching a Software Process Based on the Personal Software Process," 1115–1120. *21st International Association of Science and Technology for Development on Applied Informatic*s. Innsbruck, Austria, February 2003. Acta Press, 2003.

[Cardona 2012a]
Cardona, S. *Diseño de una estrategia de aprendizaje para implementar prácticas de psp y tsp en cursos básicos de programación.* Master's thesis, Universidad EAFIT, 2012.

[Cardona 2012b]
Cardona, S. & Rincón, R. "Ambiente Virtual de Aprendizaje para la Implementación de Prácticas de PSP y TPS en un Curso de Programación de Computadores," 406–416. In *IV Congreso Iberoamericano Soporte del Conocimiento con la Tecnología*. Pontificia Bolivariana University, Bucaramanga, Columbia, October 2012.

[El Eman 1996]
El Eman, K., Shostak, B., & Madhavji, N. "Implementing Concepts from the Personal Software Process in an Industrial Setting," 117–131. *Proceedings of the Fourth International Conference*

on the Software Process. Brighton, U.K., December 1996. International Software Process Association, 1996.

[Hayes 1998]
Hayes, W. "Using a Personal Software Process to Improve Performance," 61–71. *Proceedings of the Fifth International Software Metrics Symposium*. Bethseda, MD, November 1998. IEEE, 1998. doi:10.1109/METRIC.1998.731227

[Honig 2008]
Honig, W. L. "Teaching Successful 'Real-World' Software Engineering to the 'Net' Generation: Process and Quality Win!" 25–32. *21st Conference on Software Engineering Education and Training*. Charleston, SC, April 2008. IEEE, 2008. doi:10.1109/CSEET.2008.38

[Humphrey 1995]
Humphrey, W. *A Discipline For Software Engineering* (p. 789). Addison-Wesley, 1995.

[Humphrey 1997]
Humphrey, W. *Introduction to the Personal Software Processs* (p. 278). Addison-Wesley Logmaan, 1997.

[Lisack 2000]
Lisack, S. K. "The Personal Software Process in the Classroom: Student Reactions (An Experience Report)," 169–175. *Proceedings of the 13th Conference on Software Engineering Education and Training*, Austin, TX, March 2000. IEEE, 2000.

[Prechelt 2001]
Prechelt, L. & Unger, B. "An Experiment Measuring the Effects of Personal Software Process (PSP) Training." *IEEE Transactions on Software Engineering 27*, 5 (May 2001): 465–472.

[Rincón 2010]
Rincón, R. "Análisis y Capitalización de las Experiencias y Lecciones Aprendidas de la Implementación de PSP (Personal Software Process) y TSP (Team Software Process) desde el Sector Académico a las Empresas de Software Mexicanas," 12–15. In *Informe Final Sabático.* Medellín, Colombia, 2010.

[Rombach 2008]
Rombach, D., Münch, J., Ocampo, A., Humphrey, W. S., & Burton, D. "Teaching Disciplined Software Development." *Journal of Systems and Software 81,* 5 (May 2008): 747–763.

[Rong 2012]
Rong, G., Zhang, H., & Xie, M. "Improving PSP Education by Pairing: An Empirical Study," 1245–1254. *International Conference on Software Engineering.* Zurich, Switzerland, June 2012. Curran, 2012.

[Runeson 2003]
Runeson, P. "Using Students as Experiment Subjects – An Analysis on Graduate and Freshmen Student Data," 95–102. *Proceedings of the 7th International Conference on Empirical Assessment in Software Engineering*. Keele University, Staffordshire, U.K., April 2003.

[Towhidnejad 1997]
Towhidnejad, M. & Hilburn, T. "Integrating the Personal Software Process (PSP) Across the Undergraduate Curriculum," 162–168. *27th Annual Conference on Frontiers in Education Conference: Teaching and Learning in an Era of Change.* Pittsburgh, PA, November 1997. Stipes Publishing, 1997.

[Venkatasubramanian 2001]
Venkatasubramanian, K., Roy, S. B. T., & Dasari, M. V. "Teaching and Using PSP in a Software Engineering Course: An Experience Report." *Software Engineering Education and Training Annual Conference* (SEETAC 2001), Chennai, India.

5 Factors Affecting Productivity Performance in PSP Training

Mushtaq Raza, University of Porto
João Pascoal Faria, University of Porto
Pedro Henriques, Strongstep – Innovation Center in Software Quality
William Nichols

Abstract

We analyzed the data from PSP training courses, involving approximately 3000 students, to determine the personal and non-personal factors that affect productivity performance. Regarding non-personal factors, we found, by conducting a detailed per-phase analysis, both process changes and project complexity to be important factors explaining productivity variations throughout the sequence of programs. Regarding personal factors, we found significant variations among individuals that can be partially explained by personal experience and programming language used. We also show that an improved estimation model can be derived by taking into account these factors, leading to significant reductions in estimation errors. Understanding these factors is also useful in analyzing the productivity of individual engineers.

5.1 Introduction

5.1.1 Motivation

The motivation for this work arose in the context of a research project whose goal is to develop models and tools to help PSP students and practitioners analyze their performance, namely, identify performance problems, root causes, and possible improvement actions [Raza 2012, Duarte 2012a]. In previous work, we identified a set of factors affecting, directly or indirectly, time estimation performance, together with performance indicators and recommended values for all the variables involved [Duarte 2012a, 2012b]. To arrive at a similar model for the productivity, we first have to determine which factors affect productivity of PSP developers. The main goal of this paper is precisely to determine such factors, based on the analysis of SEI course data. The knowledge of those factors may be of interest not only for performance analysis (our original motivation), but also for other purposes, like improving estimation methods or even the course design.

From previously published studies, it is known that students' productivity during the PSP training decreases in the first assignments and recovers in the last assignments [Hayes 1997]. An explanation that is usually mentioned is that the initial decrease is caused by the introduction of process changes, and recovery occurs as the new processes or process components are practiced [Rombach 2008]. But to our knowledge, no detailed studies exist providing evidence in favor of that explanation in the context of the PSP. In addition, significant variations of productivity among individuals are often observed [Wen-Hsiang 2011], but to our knowledge, no detailed studies exist that analyze the causes of those variations.

5.1.2 Research Questions and Methods

Considering the motivation previously stated, we aim to answer the following research questions and sub-questions:

RQ1: What non-personal factors affect the evolution of overall productivity[5] and productivity per phase[6] of PSP developers during their PSP training projects?

RQ1.1: Do process changes affect productivity?

RQ1.1.1: Does the productivity decrease initially with the addition of process components?

RQ1.1.2: Does the productivity increase with the repeated usage of process components?

RQ1.2: Do other project characteristics affect productivity?

RQ2: What personal factors (personal characteristics and personal choices) may explain productivity variations among individuals for the same assignments?[7]

RQ2.1: Does personal programming experience affect productivity?

RQ2.2: Does the programming language chosen affect productivity?

RQ3: By taking into account non-personal and personal factors, besides the historical productivity of each individual, is it possible to improve productivity estimates?[8]

To answer these questions, we analyzed SEI's PSP for Engineers I/II training data, including data from 31,140 submissions by 3,114 students for 10 assignments, produced during 295 training classes that occurred between 1994 and 2005.

We started by selecting the relevant tables and columns for the analysis. For each submission, we selected the following data: actual effort, actual size, estimated effort, estimated size, actual effort (time) per phase, student number, and assignment number. For each student, we also selected the following information: programming language used in the course, years of programming experience, volume of code previously developed using the course programming language, and year of the class. Additional information was occasionally inspected.

The next step was to clean the data. We excluded all submissions with 0 minutes for any phase (except for the optional Compile phase or for the DLDR and CR phases before Assignment 7), or with a significant discrepancy (>2 min) between the actual effort and the summation of the actual effort per phase. In the end, we had 26,140 records (submissions) selected.

Before presenting the analysis results, we review the PSP training context in Section 5.2. Subsequently, we analyzed the selected data to answer the research questions and determine the non-personal factors, as described in Section 5.3, and the personal factors, as described in Section 5.4. We conclude the paper in Section 5.5 with a summary of the major findings and recommendations for future work.

---

[5] Productivity is measured in LOC/hour in this study. We also use its inverse in min/LOC.

[6] By analyzing the evolution of the productivity per phase, we expect to obtain a better understanding of the influence of process changes, since they are usually localized in specific phases.

[7] In this study, we analyze only productivity variations among individuals in LOC/hour. In future work, we intend to also analyze variations in terms of time needed to accomplish the same assignments.

[8] In the PROBE estimation method, a productivity estimate (such us the average of previous projects) is implicitly combined with a size estimate to arrive at an effort estimate.

5.2 PSP Training Context: Projects and Process Changes

Table 12 briefly describes the programming projects that are part of the PSP for Engineers I/II courses, the PSP level used in each project, and the authors' judgment of project complexity (in terms of aspects that may lead to a higher development effort per LOC). Figure 10 summarizes the process changes during the training, using feature modeling concepts and notation [Kang 1990]. Such feature modeling will be used to derive performance models in a systematic way.

Table 12: Sequence of Programming Projects and PSP Levels Throughout the PSP Training Course

#	Description	Complexity Level (Authors' Judgment)	PSP Level
1	Mean and standard deviation	Low: simple numerical problem, formulas and test cases given	PSP0
2	Size counting for a program	High: text parsing, no design guidelines, no test cases given	PSP0.1
3	Size counting for a program and its parts	High (same reason as #2)	PSP0.1
4	Linear regression parameters	Low (same reason as #1)	PSP1
5	Simpson's rule integration with normal distribution	Medium: numerical problem with textual algorithm description	PSP1.1
6	Prediction intervals with linear regression and *t* distribution	High: very complex numerical problem	PSP1.1
7	Correlation and significance	Medium: nontrivial numerical problem	PSP2
8	Sort list of pairs	Medium: nontrivial algorithmic problem	PSP2.1
9	Degree to which data fits normal distribution	Medium: nontrivial numerical problem	PSP2.1
10	Multiple regression	Medium: nontrivial numerical problem	PSP2.1

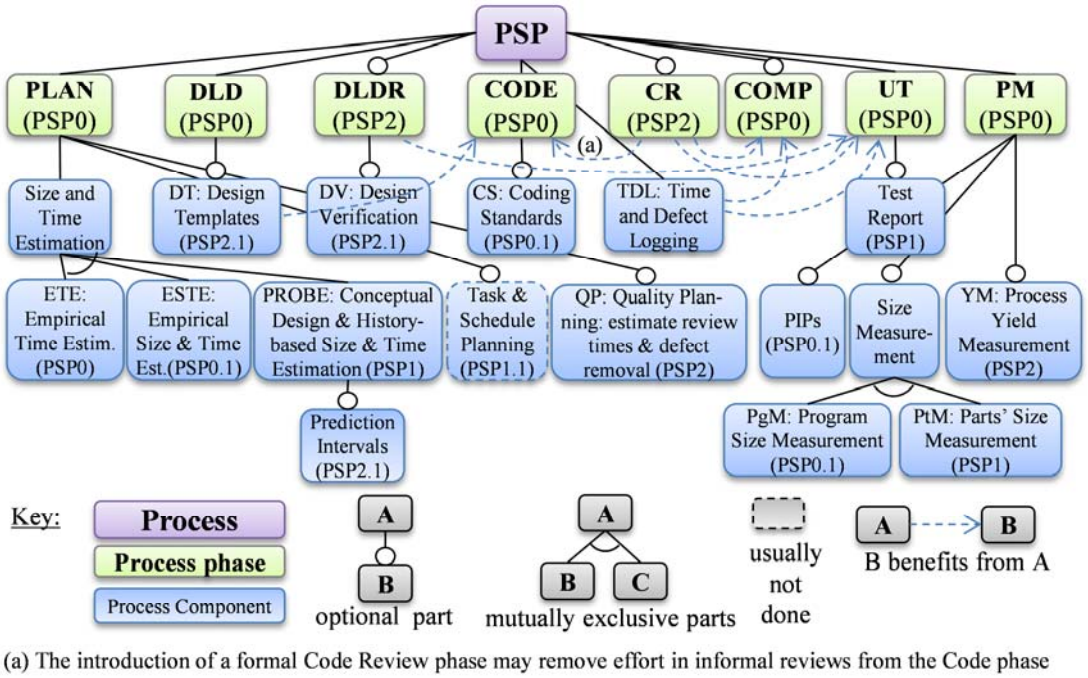

Figure 10: Feature Model of PSP Phases and Components, Showing Changes from PSP0 to PSP2.1

5.3 Analysis of Nonpersonal Factors

5.3.1 Influence of Process Changes and Project Complexity on Productivity

In order to have a first insight about the impact of process changes (RQ1.1) and other project characteristics (RQ1.2) on the evolution of productivity, we computed the chart in Figure 11 from the data set described in the introduction. We computed the productivity per phase, instead of the overall productivity, to obtain a better insight of the influence of process changes, since they usually impact specific phases. To facilitate summations, we measured the inverse of the productivity, that is, the normalized effort in min/LOC. To exclude personal factors, we computed the average for all students.

By comparing the changes in productivity per phase with the process changes marked over the chart (based on the information in Table 12), we conclude that most of the former can be explained by the latter. The most significant of the remaining changes, namely, the slower Code and UT phases in Projects 2 and 3 and in the UT phase in Project 6, could be explained by a higher complexity of those projects (see the authors' judgment of project complexity in Table 12).

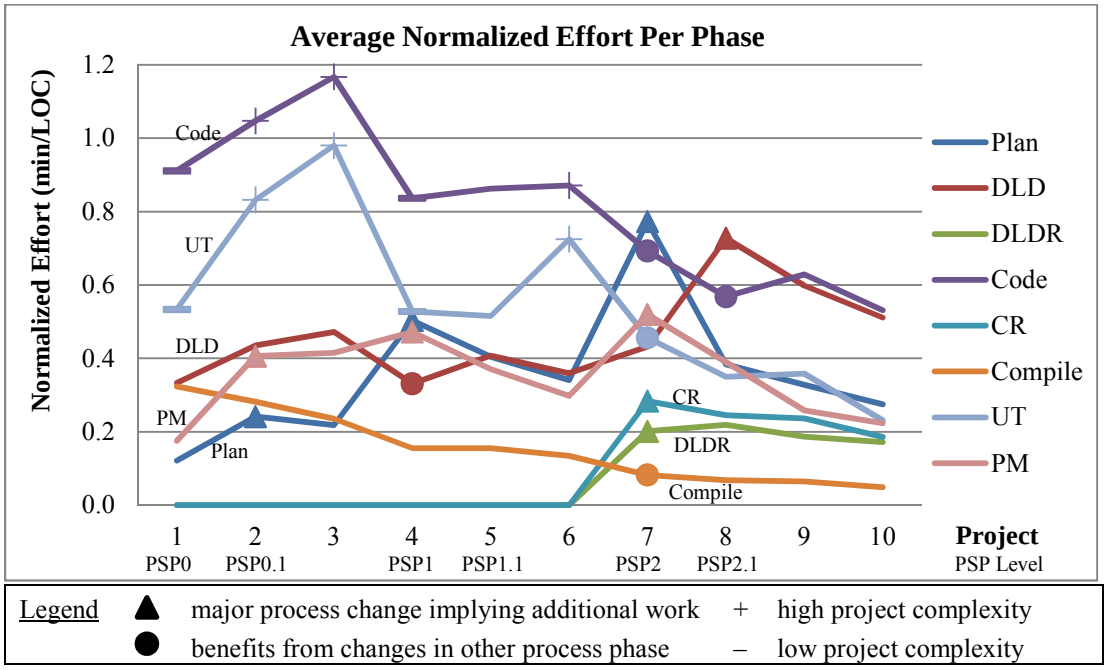

Figure 11: Evolution of the Average Normalized Effort per Phase Throughout the Programs

The chart also allows us to observe the magnitude of productivity changes that occur for each process change. The most noticeable impacts occur with the changes in the PLAN phase in Project 7 (introduction of quality planning) and in the DLD phase in Project 8 (introduction of design templates). In both cases, the time spent in the phase affected exceeds the time spent in the Code phase. The chart also shows that there is an increase of DLD time and a decrease of Code time throughout the training, with similar values by the end of the training. There is also a significant reduction of Compile and Test time and a closer balance between appraisal (reviews) and failure (bug fixing) efforts by the end of the training.

5.3.2 Regression Models for the Average Productivity per Phase

To determine quantitatively the degree to which process changes and variations in project complexity may explain productivity variations, we computed nonlinear multiple regression models for the average normalized effort per phase (in min/LOC) for each project, taking those factors into account. Let us start with the following definitions:

- $\boldsymbol{y_{ki}}$: average for all students of normalized effort (min/LOC) spent in project *i* and phase *k*
- $\boldsymbol{\hat{y}_{ki}}$: regression value for $\boldsymbol{y_{ki}}$, as a function of several coefficients and predictor variables
- $\boldsymbol{r_{ki} \triangleq y_{ki} - \hat{y}_{ki}}$: residual, that is, the difference between actual and regression values
- $\boldsymbol{s_k \triangleq \sqrt{\sum_{i=1}^{n} r_{ki}^2}\,/n}$: residuals standard error (RSE) for the $n = 10$ projects (data points)

The predictor variables for each $\boldsymbol{\hat{y}_{ki}}$ (denoted in uppercase Latin symbols) are determined based on the following information:

- $\boldsymbol{U_{ji}}$: process phase or component *j* (any optional or alternative non-dashed node in Figure 10) is used in project *i*, encoded as 1 = yes, 0 = no (determined from Table 12 and Figure 10)
- $\boldsymbol{T_{ji} \triangleq \sum_{l=1}^{i-1} U_{jl}}$: number of previous projects using process phase or component *j*
- $\boldsymbol{P_k}$: set of child components of process phase *k* (determined from Figure 10)
- $\boldsymbol{A_k}$: set of components from which process phase *k* benefits (determined from Figure 10)
- $\boldsymbol{C_i}$: complexity of project *i*, encoded as 1 = Low, 2 = Medium, 3 = High (from Table 12)

The needed coefficients for each $\boldsymbol{\hat{y}_{ki}}$ (denoted in lowercase Greek symbols) are determined based on the following hypothesis:

- The normalized effort of a mandatory process phase *k*, while optional components are not introduced, is given by a constant value β_k (computed for the lowest project complexity).
- The impact of introducing a process phase or component *j* (any optional or alternative node in Figure 10), in terms of added normalized effort (or removed, in case of alternative replacement), can be described by an exponential learning curve $\boldsymbol{\varphi_j}\left(1 + \frac{\iota_j - \varphi_j}{\varphi_j} 2^{-\frac{T_{ji}}{\lambda_j}}\right)$, with initial value (when $\boldsymbol{T_{ji} = 0}$) ι_j, final value (when $\boldsymbol{T_{ji} \to \infty}$) $\boldsymbol{\varphi_j}$, and half-learning "time" λ_j (times used to reach the mid-value $\boldsymbol{(\iota_j + \varphi_j)/2}$).
- The impact of introducing a process phase or component *j* on another process phase *k* that benefits from *j* can be described by a reduction $\boldsymbol{\delta_{jk}}$ of the normalized effort in phase *k*.
- The impact of the project complexity on the normalized effort in phase *k* can be described by a linear relation with slope μ_k dependent on the phase, that is, a multiplier $\boldsymbol{\left(1 + \mu_k(C_i - C_{min})\right)}$.
- Complexity affects significantly only DLD, CODE, and UT phases (i. e., $\mu_k \approx 0$ for other phases).[9]

Considering the above information, the general form of $\boldsymbol{\hat{y}_{ki}}$ for mandatory phases will be

[9] Our data set doesn't allow us to draw conclusions regarding the impact of project complexity on the CR and DLDR phases, because the projects with a CR and DLDR phase have the same project complexity. Regarding other phases, the data in Figure 11 doesn't suggest any significant impact.

$$\hat{\boldsymbol{y}}_{\boldsymbol{ki}} = \left[\beta_k + \sum_{j\in P_k} \varphi_j \left(1 + \alpha_j 2^{-\frac{T_{ji}}{\lambda_j}}\right) U_{ji} - \sum_{j\in A_k} \delta_{jk} U_{ji}\right] \left(1 + \mu_k (C_i - 1)\right), \text{ with } \alpha_j = \frac{\iota_j - \varphi_j}{\varphi_j},$$

and for optional phases (CR and DLDR) will be

$$\hat{\boldsymbol{y}}_{\boldsymbol{ki}} = U_{ki}\left[\varphi_k \left(1 + \alpha_k 2^{-\frac{T_{ki}}{\lambda_k}}\right) + \sum_{j\in P_k} \varphi_j \left(1 + \alpha_j 2^{-\frac{T_{ji}}{\lambda_j}}\right) U_{ji} - \sum_{j\in A_k} \delta_{jk} U_{ji}\right] \left(1 + \mu_k (C_i - 1)\right).$$

Subsequently, we expanded the summations for each phase, as illustrated for the DLD phase:

$$\hat{y}_{DLD,i} = \left[\beta_{DLD} + \varphi_{DT} \left(1 + \alpha_{DT} 2^{-\frac{T_{DT,i}}{\lambda_{DT}}}\right) U_{DT,i} - \delta_{PROBE,DLD} U_{PROBE,i}\right] \left(1 + \mu_{DLD} (C_i - 1)\right)$$

We computed the coefficients by the least square method (minimizing s_k). Because of the small number of data points (10 projects), we had to simplify some theoretical formulas to assure convergence of the method (see explanations in Figure 13). The results obtained are shown in Figure 12 and Figure 13. From the charts and the values of s_k, we conclude that the factors considered provide a good explanation for the average productivity per phase.

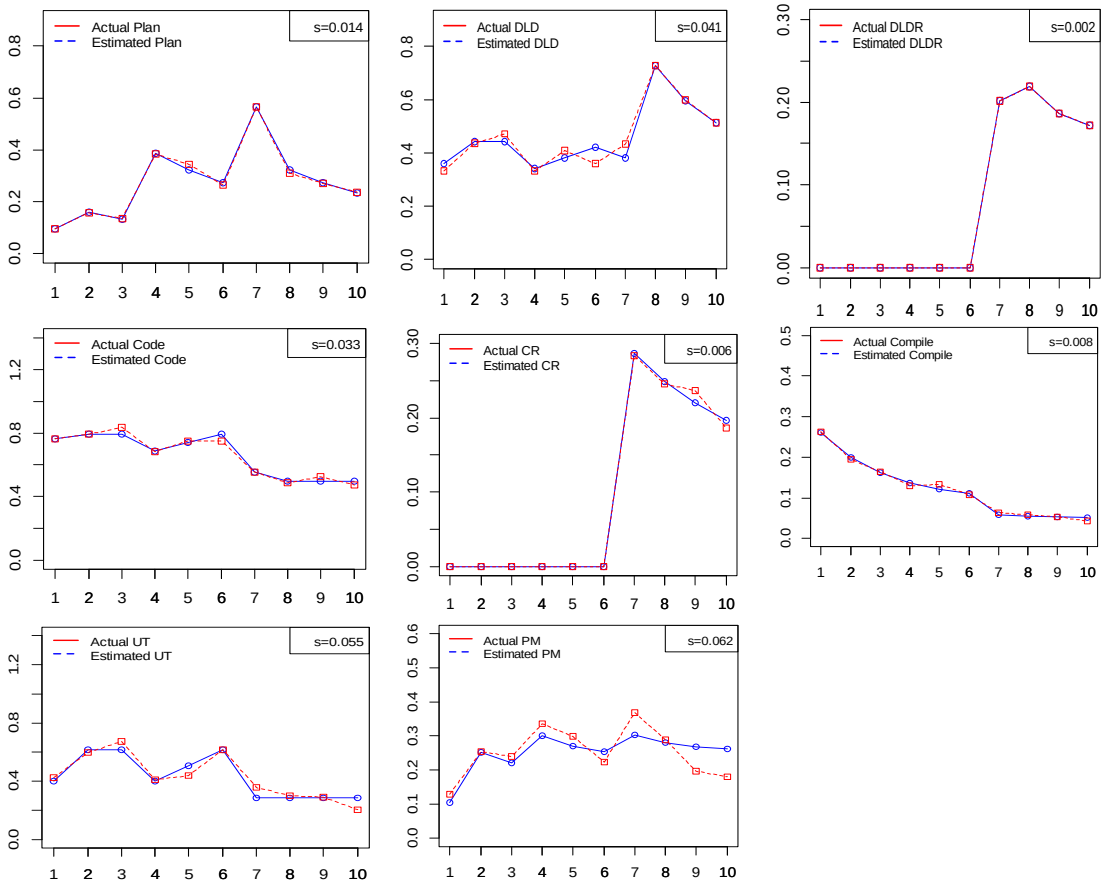

Figure 12: Charts with the Normalized Effort per Phase (min/LOC) Throughout the 10 Projects, Comparing the Actual Values (Average for All Individuals) and Regression Values

$\hat{y}_{PLAN,i}$	$= (0.029U_{ETE,i}+0.061U_{ESTE,i}+ 0.120U_{PROBE,i}+0.067U_{QP,i})\ (1+3.1 \times 2^{-T_{PLAN,i}/2.4})$ *(used a single, unified, learning effect (on the right), with $T_{PLAN,i}$ restarting from 0 on each process change; ignored Task & Schedule Planning)*
$\hat{y}_{DLD,i}$	$= [0.266 +0.021\ (1+9.76\times2^{-T_{DT,i}/1.04})\ U_{DT,i}](1+0.089(C_i-1))$
$\hat{y}_{DLDR,i}$	$= U_{DLDR,i}\ (0.11+0.050U_{DV,i})\ (1+0.81\times2^{-T_{DLDR,i}/0.84})$ *(used a single, unified, learning effect (on the right), with $T_{DLDR,i}$ restarting from 0 on each process change)*
$\hat{y}_{CODE,i}$	$= [0.91 - 0.10U_{CS,i} - 0.20U_{CR,i} - 0.10U_{DT,i}]\ (1+0.12(C_i-1))$ *(didn't consider any learning effect associated with the introduction of the CS process component)*
$\hat{y}_{CR,i}$	$= U_{CR,i}\ [0.120\times(1+0.896\times2^{-T_{CR,i}/3.0})]$ *(fixed the half-learning time = 3 to force convergence)*
$\hat{y}_{Comp,i}$	$= 0.094 + 0.166\times2^{-T_{TDL,i}/1.53} - 0.046U_{CR,i}$ *(considered a learning effect associated with time and defect logging)*
$\hat{y}_{UT,i}$	$= (0.495 – 0.233\ U_{CDR,i})\ (1 + 0.330(C_i - 1))$ *(merged the effects of CR and DLDR, because they're indistinguishable; ignored impact of time & defect logging)*
$\hat{y}_{PM,i}$	$= 0.14+0.15U_{PgM,i}(1+0.9\times2^{-T_{PgM,i}/1.6})+0.027U_{PtM,i}(1+0.9\times2^{-T_{PtM,i}/1.6})+0.021U_{YM,i}(1+2.0\times2^{-T_{YM,i}/1.6})$ *(ignored the impact of PIPs; used the same half-learning time for all components to force convergence)*

Figure 13: Regression Models for the Average Normalized Effort per Phase in a Project i

5.4 Analysis of Personal Factors

In this section we aim to identify, based on the available data, possible personal factors that explain productivity variations among individuals for the same projects (RQ2). First, we'll check whether there are groups of individuals that consistently perform better than others.

5.4.1 Productivity Variations Among Individuals

Figure 14 shows the mean productivity of each group of PSP training students (G1 to G5), for the 10 projects. The groups stratify the students into groups of equal size according to their mean productivity throughout the 10 projects. For example, G1 contains the students with the 20% lowest values of mean productivity during the 10 projects. The chart shows that (1) there are significant differences in productivity among individuals and (2) individuals have a consistent productivity during the 10 projects (i.e., groups keep their relative position throughout the 10 projects).

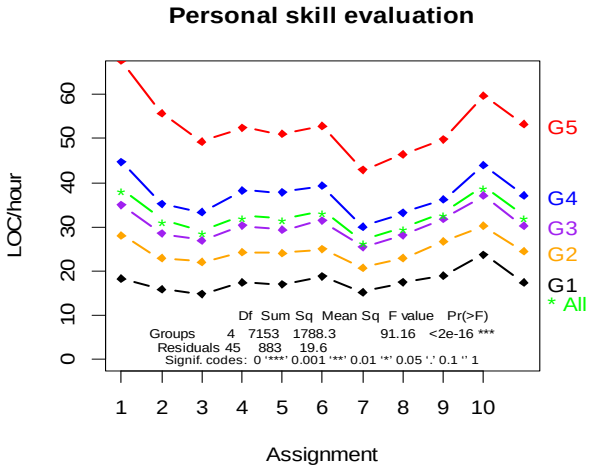

Figure 14: Difference Among Mean Productivity for Different Groups of Individuals in the 10 Programs The last column refers to the average for all assignments.

The extremely small p value ($<2^{-16}$) obtained in the analysis of variance (see Figure 14) confirms the hypothesis that the differences of mean productivity among the groups are statistically significant (significance threshold below 0.1%).

5.4.2 Impact of Technology and Experience on Productivity

To find an explanation for the differences among individuals, we analyzed existing data characterizing the individuals who attended the courses—namely, the experience and programming language used—obtaining the charts shown in Figure 15. The labels in the horizontal axes show the classes considered for each characteristic, and the numbers immediately above indicate the number of individuals in each class. The vertical axis shows the ratios between the average productivity (in minutes/LOC) of the students in each class and the average productivity for all students (2.95 min/LOC). The results obtained show that all three characteristics analyzed influence the productivity during the course, with best values for 6–8 years of programming experience, C# programming language (followed by Java), and 20–100 KLOC previously developed in the programming language used in the course.

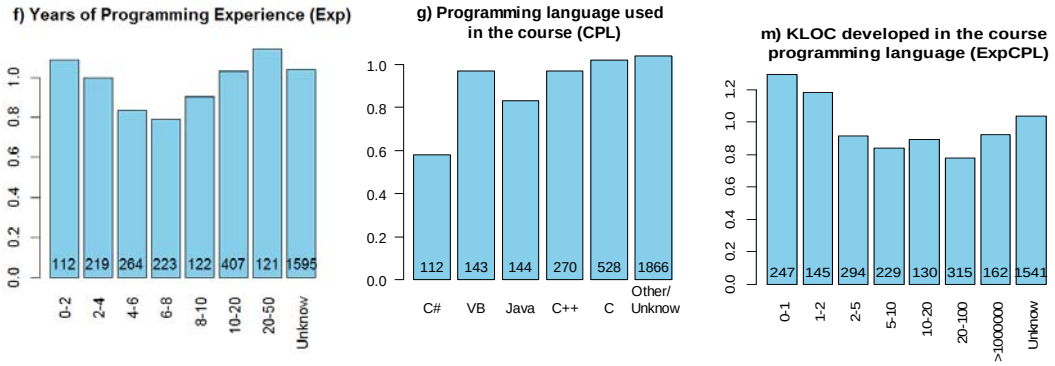

Figure 15: Charts Showing the Impact of Experience and Programming Language in Productivity

5.4.3 Improved Productivity Estimation Model

In this section, we aim to answer the last research question:

> **RQ3:** By taking into account additional non-personal and personal factors, besides the historical productivity of each individual, is it possible to improve productivity estimates?

To that end, we built a productivity model in two phases. First, we obtained a performance model considering only the non-personal factors, by summing up the formulas for the normalized effort per phase obtained in Section 5.3.2., that is,

$\hat{y}_i = \sum_{k=PLAN}^{PM} \hat{y}_{ki}$ – estimated min/LOC in program i, considering only non-personal factors.

Subsequently, we applied correction multipliers to the above model for the three personal factors identified in Section 5.4.2, plus a multiplier related to the historical personal productivity. Since the four factors considered are not completely independent, we had the need to apply only a fraction $\phi\,(0 \le \phi \le 1)$ of each multiplier M, using a modified multiplier M' of the form $M' = [1 + \phi\,(M - 1)]$. For example, if $M = 1.24$ and $\phi = 0.5$, then $M' = 1.12$. The final model obtained for the estimated normalized effort, in min/LOC, of developer j in program i is

$$\hat{z}_{ij} = \hat{y}_i[1 + 0.18(f(Exp_j) - 1)][1 + 0.22(g(CPL_j) - 1)][1 + 0.089(m(ExpCPL_j) - 1)][1 + 0.96(H_{ij} - 1)]$$

where

- $H_{ij} = \begin{cases} avg\ \{z_{hj}/\hat{z}_{hj} |\ h = 1, \ldots, i-1\}, if\ i > 1 \\ 1, if\ i = 1 \end{cases}$: historical productivity factor of developer *j*
- Exp_j: class of years of programming experience of developer *j* (*x* axis, first chart of Fig. 6)
- CPL_j: class of programming language used by developer *j* (*x* axis, second chart of Fig. 6)
- $ExpCPL_j$: class of KLOC of programming experience of developer *j* (third chart Fig. 6)
- f, g, m: multipliers for the class indicated as argument, taken from the *y* axis in Fig. 6

All the numerical coefficients, defining the fraction (or weight) considered of each factor, were calibrated by the least square method, using as data points all the 26,140 submissions. It is worth noting that the weights obtained for the three personal factors analyzed are small, as compared to the weight of the historical productivity in the previous projects, most probably because those factors are known only for a small percentage of the students.

A positive answer to RQ3 is given in Table 13.

Table 13: Residual Standard Error (RSE) Comparison

	Projects with Size and Effort Estimates (2 to 9)	**All Projects**
RSE calculated from students' estimates	2.771	—
RSE calculated from Phase 1 model: $\hat{y}_i$	2.657	2.620
RSE calculated from final model: $\hat{\boldsymbol{z}}_{\boldsymbol{ij}}$	2.314 (17% improvement)	2.282

5.5 Conclusions and Future Work

5.5.1 Findings

By looking into the evolution of the productivity per phase of PSP students along the training, the study shows that the productivity tends to follow a learning curve, with a tendency for productivity to degrade when process changes are introduced in a phase and to improve as time passes. The study also suggests that this learning phenomenon may explain almost all of the most significant productivity changes per phase.

A somewhat surprising result from the study was that process changes were not sufficient to explain some significant variations in the average productivity per phase. We found that a possible explanation for some of the variations found—namely, the significantly higher time per LOC spent in the DLD, Code, and UT phases in Projects 2 and 3 and in the UT phase in Project 6—might be attributed to a higher complexity of those projects. An open problem is how to measure complexity objectively; in particular, we intend to investigate cyclomatic complexity.

Regarding personal factors (personal characteristics and personal choices), we found that both the programming experience (years and amount of code developed) and programming language used have a significant impact on productivity.

By taking into account the non-personal and personal factors identified, we showed that it is possible to obtain, on average, better productivity estimates than the ones done by the students based on personal historical data only.

5.5.2 Future Work

As future work, we intend to formally confirm with hypothesis tests some of the above findings. We intend also to build a quantitative process performance model to help identify and rank root causes of productivity problems (by using the model in the backward direction) and predict the impact of improvement actions (by using the model in the forward direction). A similar analysis for other performance indicators will be conducted based on the SEI data set.

5.6 Acknowledgments

The work of J. Faria and P. Henriques is partly funded by FEDER (Fundo Europeu de Desenvolvimento Regional) through the Portuguese ON.2 Program (Programa Operacional Região do Norte), under project reference SI IDT - 21562/2011. The work of M. Raza is funded by FCT (Fundação para a Ciência e a Tecnologia), under research grant SFRH/BD/85174/2012.

5.7 Author Biographies

João Pascoal Faria
Assistant Professor and Researcher
INESC TEC and Department of Informatics Engineering, Faculty of Engineering, University of Porto
João Pascoal Faria received his PhD in Electrical and Computer Engineering from the Faculty of Engineering of the University of Porto in 1999 and is currently an assistant professor at the university in the Department of Informatics Engineering and a researcher at INESC Porto. He is vice-president of the Sectorial Commission for the Quality in Information and Communication Technologies (CS03) from the Portuguese Quality Institute (IPQ). In the past, he worked with several software companies and cofounded two others. He has more than 20 years of experience in education, research, and consultancy in several software engineering areas. He is the main author of a rapid application development tool (SAGA), based on domain-specific languages, with more than 20 years of market presence and evolution (1989–2011). Since 2008, he has been a certified PSP Developer, authorized PSP Instructor, and trained TSP Coach by the Software Engineering Institute of Carnegie Mellon University. He is currently involved in research projects, supervisions, and consulting activities in the areas of model-based testing, model-driven development, and software process improvement.

Mushtaq Raza
PhD student
MAP-i Doctoral Program, University of Porto, Portugal
Mushtaq Raza is a PhD student in the MAP-i Doctoral Program at the University of Porto, Portugal. He is also serving Abdul Wali Khan University in Mardan, Pakistan, as an assistant professor. His research interests include PSP, global software development, and usability engineering.

Pedro Castro Henriques
Director and Senior Consultant
Strongstep – Innovation in Software Quality
Pedro Castro Henriques has a 5-year degree in software engineering and a postgraduate degree in technology entrepreneurship and commercialization. His thesis was on information systems strategic planning for the health sector. He began his career 12 years ago as a software engineer at Q-Labs Lund/DNV and soon became a consultant working in nine European countries. After his international experience, he returned to Portugal and founded the Oporto software engineering alumni association, which now has more than 1,600 members. Afterward he further specialized in process improvement, implementation, and certification in software quality. His studies in internationalization and innovation of companies and his participation in an entrepreneurship semester in Porto Business School grounded his career in this critical area. Extremely committed to innovation and entrepreneurship, he cofounded Strongstep – Innovation in Software Quality and Portic in 2010. He is currently the director of Strongstep and president of Portic. He was a facilitator in bringing SEPG Europe to Portugal for the first time in 2010. He is also a professor at FEUP in the Services Management Engineering Master, focusing on service requirements, and an invited lecturer in the Software Quality and Tests at Master.

William Nichols
Bill Nichols joined the Software Engineering Institute (SEI) in 2006 as a senior member of the technical staff and serves as a PSP Instructor and TSP Coach with the Team Software Process (TSP) Initiative. Prior to joining the SEI, Nichols led a software development team at the Bettis Laboratory near Pittsburgh, Pennsylvania, where he had been developing and maintaining nuclear engineering and scientific software for 14 years. His publications include the interaction patterns on software development teams, design and performance of a physics data acquisition system, analysis and results from a particle physics experiment, and algorithm development for use in neutron diffusion programs. He has a doctorate in physics from Carnegie Mellon University.

5.8 References

[Duarte 2012a]
Duarte, C. "Automated Software Processes Performance Analysis and Improvement Recommendation," MSc thesis, Faculty of Engineering of the University of Porto, 2012.

[Duarte 2012b]
Duarte, C. B., Faria, J. P., & Raza, M. P. "PSP PAIR: Automated Personal Software Process Performance Analysis and Improvement Recommendation." *Proceedings of the 8th International Conference on the Quality of Information and Communications Technology - QUATIC 2012.* Lisbon, Portugal, September 2012. Conference Publishing Services, 2012.

[Duarte 2012c]
Duarte, C. B., Faria, J. P., Raza, M. P., & Henriques, C. "Model and Tool for Analyzing Time Estimation Performance in PSP," 21–40. In *TSP Symposium 2012 Proceedings* (CMU/SEI-2012-SR-015). Software Engineering Institute, Carnegie Mellon University. http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=34091

[Hayes 1997]
Hayes W. & Over, J. W. *The Personal Software Process*^{SM} *(PSP*^{SM}*): An Empirical Study of the Impact of PSP on Individual Engineers* (CMU/SEI-97-TR-001). Software Engineering Institute, Carnegie Mellon University, 1997. http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=12801

[Kang 1990]
Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., & Peterson, A. S. *Feature-Oriented Domain Analysis (FODA) Feasibility Stud*y (CMU/SEI-90-TR-21). Software Engineering Institute, Carnegie Mellon University, 1990. http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=11231

[Raza 2012]
Raza, M. "Automated Software Process Performance Analysis and Improvement Recommendation," PhD pre-thesis proposal, Faculty of Engineering of the University of Porto, 2012.

[Rombach 2008]
Rombach, D., Münch, J., Ocampo, A., Humphrey, W. S., & Burton, D. "Teaching Disciplined Software Development." *Journal of Systems and Software 81*, 5 (May 2008): 747–763.

[Wen-Hsiang 2011]
Wen-Hsiang S., Nien-Lin H., & Wei-Mann L. "Assessing PSP Effect in Training Disciplined Software Development: A Plan–Track–Review Model." *Information and Software Technology 53*, 2 (February 2011): 137–148

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	January 2014	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
TSP Symposium 2013 Proceedings	FA8721-05-C-0003

6. AUTHOR(S)

Sergio Cardona, João Pascoal Faria, Fernanda Grazioli, Pedro Henriques, James McHale, Silvana Moreno, William Nichols, Leticia Pérez, Mushtaq Raza, Rafael Rincón, Diego Vallespir

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2013-SR-022

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFLCMC/PZE/Hanscom Enterprise Acquisition Division 20 Schilling Circle Building 1305 Hanscom AFB, MA 01731-2116	

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

The 2013 TSP Symposium was organized by the Software Engineering Institute and took place September 16–19 in Dallas, Texas. The goal of the TSP Symposium is to bring together practitioners and academics who share a common passion to change the world of software engineering for the better through disciplined practice. The conference theme was "When Software Really Matters," which explored the idea that when product quality is critical, high-quality practices are the best way to achieve it. In keeping with that theme, the community contributed a variety of technical papers describing their experiences and research using the Personal Software Process (PSP) and Team Software Process (TSP). This report contains the four papers selected by the TSP Symposium Technical Program Committee. The topics include demonstrating the impact of the PSP on software quality and effort by eliminating the programming learning effect, analyzing student performance during the introduction of the PSP using an empirical cross-course comparison, incorporating PSP practices into introductory programming courses, and analyzing factors affecting productivity performance in PSP training.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Team Software Process, TSP, Symposium, Personal Software Process, PSP	57

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL