Hindawi Publishing Corporation
Advances in Software Engineering
Volume 2014, Article ID 861489, 10 pages
http://dx.doi.org/10.1155/2014/861489

Research Article

The Impact of the PSP on Software Quality: Eliminating the Learning Effect Threat through a Controlled Experiment

Fernanda Grazioli, Diego Vallespir, Leticia Pérez, and Silvana Moreno

Facultad de Ingeniería, Universidad de la República, Julio Herrera y Reissig 565, 11300 Montevideo, Uruguay

Correspondence should be addressed to Diego Vallespir; dvallesp@fing.edu.uy

Received 30 May 2014; Revised 5 September 2014; Accepted 5 September 2014; Published 30 September 2014

Academic Editor: Robert J. Walker

Data from the Personal Software Process (PSP) courses indicate that the PSP improves the quality of the developed programs. However, since the programs (exercises of the course) are in the same application domain, the improvement could be due to programming repetition. In this research we try to eliminate this threat to validity in order to confirm that the quality improvement is due to the PSP. In a previous study we designed and performed a controlled experiment with software engineering undergraduate students at the Universidad de la República. The students performed the same exercises of the PSP course but without applying the PSP techniques. Here we present a replication of this experiment. The results indicate that the PSP and not programming repetition is the most plausible cause of the important software quality improvements.

1. Introduction

The Personal Software Process (PSP) is a software development process for the individual [1]. The PSP helps the engineer to control, manage, and improve his or her work and it is taught through a course. The students (many times software engineers) perform several programming exercises in which techniques and phases of the PSP are added as the exercises advance. For each exercise, process data are collected.

Data from the courses indicate that the PSP improves the quality of the products developed [2–5]. One way used to determine this is through statistical analysis of the evolution of the results obtained by the students in each program of the course. For example, if the programs developed 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 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 quality improvements observed in the PSP courses due to the introduction of the phases and techniques of the PSP or due to programming repetition? To investigate this we designed and performed a controlled experiment with software engineering undergraduate students at the Universidad de la República. The students performed the exercises from the course without applying the PSP techniques. This makes it possible to know if quality improves by the simple fact of programming repetition. In the context of this study, product quality is measured as defect density.

The designed experiment was executed in 2012 [6], and an exact replication was executed in 2013. The subjects of our experiment perform 8 program assignments without applying the PSP techniques. Then, the collected data are statistically analyzed and compared with the data collected during the regular PSP course. In this paper, we present the analysis of both executions of the experiment. The most important result indicates that the PSP and not programming repetition is the most plausible cause of the important software quality improvements.

Section 2 briefly presents the PSP. The related works are presented in Section 3. Section 4 presents the way in which the study was designed. Sections 5, 6, and 7 present the results of the research. Threats to validity are presented in Section 8 and the conclusion is in Section 9.

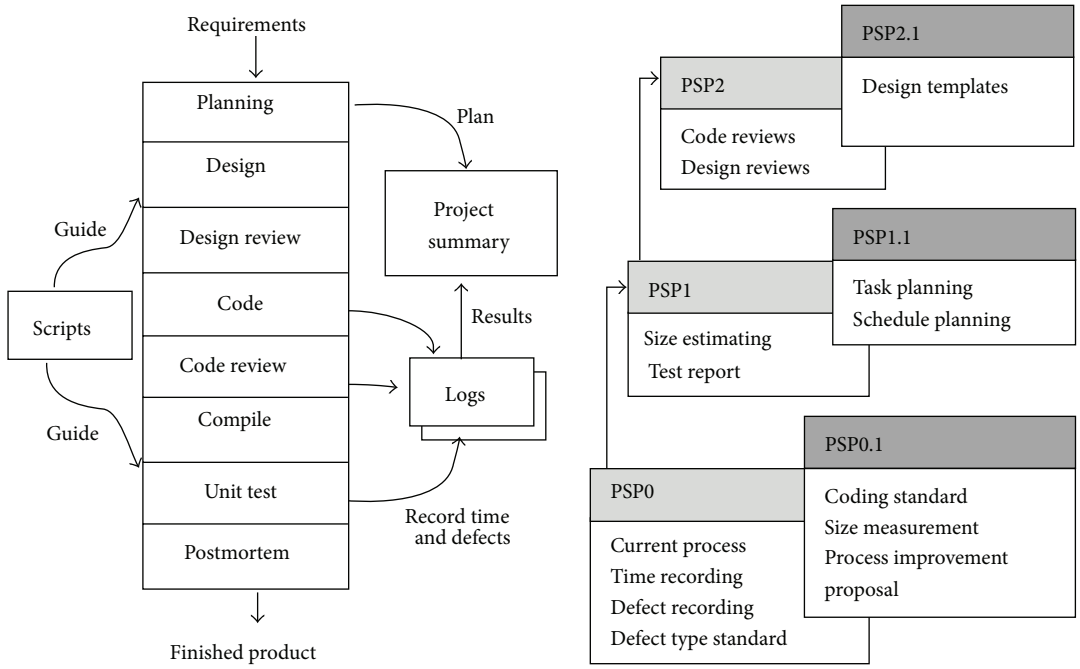

Figure 1: The PSP phases and the PSP process levels.

2. The Personal Software Process

The PSP is a software development process for the individual [1, 7]. The PSP establishes a highly instrumented development process that includes a rigorous measurement framework for effort and quality. The process includes phases and techniques that the engineer follows and uses while building the software.

For each phase, the PSP has scripts that help follow the process correctly. The phases are planning, detailed design, detailed design review, code, code review, compile, unit test, and postmortem. In each phase, the engineer collects data on the time spent in the phase and the defects injected and removed, as it is shown in Figure 1.

During the PSP course, the engineer builds programs while he progressively learns the PSP phases and techniques. For the first exercise, the engineer starts with a simple and defined process. As the course progresses, new process phases and elements are added, from estimation and planning to code reviews and to design and design review. As these elements are added, the process changes.

There are six PSP processes, also called PSP levels: PSP0, PSP0.1, PSP1, PSP1.1, PSP2, and PSP2.1. Each process builds on the prior process by adding engineering or management activities, as shown in Figure 1. The PSP is in fact the PSP2.1 level. The other PSP levels exist exclusively for the purposes of teaching the PSP.

The course has changed twice. The first version of the course, PSP for engineers I/II original, involved 10 program exercises. The second version, PSP for engineers I/II revised, involved 8 programs. The third version, PSP fundamentals and advanced, involved 7 program assignments. All three course versions have been taught in different environments, for different kinds of people all around the world: undergraduate students, graduate students, and professional software engineers [8, 9]. The SEI partners generally use in their classes the last course version, while in the academy the 10 program course version is commonly used. We know this by internal communications with the SEI and by reviewing the published articles related to PSP analysis.

As an example, Table 1 shows which PSP level is applied on each assignment in the 8-program course.

3. Related Works

The PSP has several published studies showing improvement in developer performance with process insertion. In particular, some studies show that the PSP improves the quality of the developed products [2–5].

Normally, in these studies the data are grouped by PSP level in order to be able to evaluate the different techniques and phases that are introduced and used on each PSP level. Rather than analyzing changes in group averages, these kinds of studies focus on the average changes of individual engineers, that is, analyze the improvement in developer performance.

Defect counts and measures of defect density (i.e., defects per KLOC) have traditionally served as software quality measures. The PSP uses this method of measuring product quality, as well as several process quality metrics.

Hayes and Over analyzed the data of 298 engineers who attended the PSP for engineers I/II original course until 1997 [2]. Their investigation focuses on the changes across the first three major PSP levels (0, 1, and 2). They grouped individual

Table 1: PSP levels applied on each program assignment in the PSP I/II revised course.

1	2	3	4	5	6	7	8
PSP0	PSP0.1	PSP1	PSP1.1	PSP2	PSP2.1	PSP2.1	PSP2.1

data according to these levels and then examined the change in individual performance that occurred from level to level.

The analysis of overall defect density revealed a statistically significant difference between PSP levels 0 and 1, as well as between PSP levels 0 and 2. They found that the median reduction in total defect density is factor of 1.5 between PSP levels 0 and 2. On the other hand, they were not able to reject the null hypothesis between PSP levels 1 and 2.

Analysis of both defect density in the compile phase and defect density in the test phase revealed a statistically significant difference in defect density across the three PSP levels comparisons. They found that the median reduction in defect density for the compile phase is a factor of 3.7, and, for the test phase, the median reduction is a factor of 2.5, both between PSP levels 0 and 2.

In 2007, Rombach et al. performed a replication and extension of the Hayes study [3]. They analyzed the data of 3090 engineers who attended the PSP for engineers I/II original course between 1994 and 2005. The paper does not clarify whether they use all the 3090 engineers' data for the defect density analyses or not. We know that authors began with that size sample but we do not know if they discarded some data for quality analysis.

They observed similar and significantly decreasing trends concerning defect density for the compile and test phases. For those variables, the null hypotheses were rejected for all the PSP level comparisons.

In the case of overall defect density, a general decreasing trend was observed in almost all PSP level comparisons where the null hypotheses were rejected. It was only between PSP levels 1 and 2 that they were not able to reject the null hypothesis as no significant difference was found.

In 2006, Paulk analyzed the data of 1345 engineers who attended the PSP for engineers I/II original course between 1994 and 2001 [4]. He only used defect density in test phase as the dependent variable to measure software quality.

He analyzed the changes across the four major PSP levels (0, 1, 2, and 3). A decreasing trend could be observed between all PSP level comparisons. Null hypotheses were rejected for all cases, using both statistical tests named above. He found a decrease in defect density in testing of more than 75 percent between PSP level 0 and 3.

In a later work, Paulk considered a data set of 2435 programs developed by engineers who performed the PSP for engineers I/II original course between 1994 and 2001 [5]. The article does not clarify how many engineers were involved in those developments. The data set is a subset of his previous study and in this instance he only considered programs written in C language. He analyzed the changes across the four major PSP levels like in his previous study, using the same two statistical methods. He found a statistically significant decreasing trend between all PSP level comparisons. He found that quality improved by 79 percent and that variability decreased by 81 percent between PSP level 0 and 3.

A retrospective analysis of these related works left some threats to external validity. One threat is the confounding of the effect of introducing process phases and techniques insertions with the gaining of domain experience as related programs are developed. Therefore, the question is if the improvements are due to the phases and techniques or due to programming repetition during the course (learning effect).

4. Methodology and Study Setup

The main goal of our research is to know if the software engineers improve the quality of the developed products due to the PSP itself or due to programming repetition in the same application domain. In order to answer this research question, our study is composed of three main steps:

(i) analyzing the data of the PSP for engineers I/II revised course,

(ii) designing, executing, and analyzing the data of a controlled experiment that allows us to know if quality improves by the simple fact of programming repetition,

(iii) comparing both results in order to know if there are differences between them.

In order to analyze the PSP for engineers I/II revised course, the methodology consists of analyzing the available data of students that performed the PSP complete course and looking for changes in the individual performance. The method and the hypotheses are presented in detail in the following paragraphs. The results, analysis, and discussion of the PSP regular course are presented in Section 5. Regarding the experiment that allows us to look at the effects of programming repetition, the methodology and experimental design are presented in Section 6.1, while the experiment results, analysis, and discussion are presented in Section 6.2.

We use three measures to evaluate the quality of the products: defect density in compile, defect density in unit test, and total defect density. As we show in the related works section, these are generally used for software quality assessment. The defect density is measured as the number of defects per every thousand lines of code (KLOC). The consequence of high defect density in software engineering is typically seen in the form of defect-fixing or rework effort incurred in projects, which results in poor quality products [2].

Because the PSP was developed to improve individual performance through the gradual introduction of practices, we decided to follow a similar approach to analyze the regular PSP course, examining the change in individual performance as these practices are introduced by the different PSP process

levels. We group the available data of the program assignments according to the major PSP level that was followed on that assignment. That is, we generate three groups: one with the programs developed using PSP0 and PSP0.1, another one with PSP1 and PSP1.1, and the last one with PSP2 and PSP2.1. Table 2 shows the program assignment numbers for each group according to the PSP for engineers I/II revised course.

Our PSP course analysis raises the null hypotheses and their respective alternative hypotheses for each of the three mentioned quality metrics. The hypotheses aim at knowing if, when comparing a PSP level to another one applied previously, the software engineer improves the quality of the developed products. So, we compare groups by pairs to find if the changes in each dependent variable are 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 major PSP levels (0, 1, or 2) and $i < j$.

The same type of null and alternative hypotheses is raised for the other two dependent variables.

In our experiment, the subjects perform the same eight exercises from the regular course without applying the PSP techniques. In order to get comparable results, to analyze the data of the experiment, we decide to group the data using the program assignment grouping done in the PSP I/II revised course, regardless of the PSP level. That is, we generate three groups: one with programs 1 and 2, another one with assignments 3 and 4, and the last one with exercises 5 to 8. Given these groups, for the experiment we raise the same hypotheses stated above for the three product quality measures under study.

In both cases, in the PSP regular course and in the experiment, we have a context of repeated measures samples. So, the first statistical analysis method we considered to compare defect density was the parametric ANOVA for repeated measures test, which allows investigating changes in mean scores over time points. Repeated measures ANOVA carries a set of assumptions such as multivariate normality, homogeneity of covariance matrices, and independence. The results of the Shapiro-Wilk normality test for our experiment sample indicate that it is not normally distributed. So, it does not fulfill the normality assumption for ANOVA. The same happened when testing the normality of the PSP course data. Based on the normality test results and the small size samples, we discarded ANOVA for repeated measures as the analysis method. Then we analyzed the data of the PSP course and the data of the experiment using the nonparametric Wilcoxon signed-rank test [10]. This test is used to compare two sets of scores that come from the same subjects and it does not require any assumptions about the shape of the distribution. The Wilcoxon signed-rank test uses the median in the null and in the alternative hypotheses instead of the mean.

After analyzing and discussing each study separately, the comparison between the course and the experiment is performed. This comparison is presented in Section 7, using a descriptive method and includes a complete results discussion.

Table 2: Program assignments grouped by PSP major level.

Group	Program assignments
PSP0	1 and 2
PSP1	3 and 4
PSP2	5, 6, 7, and 8

5. PSP I/II Revised Course Quality Results and Discussion

We used data from the eight-program course version, PSP for engineers I/II revised, taught between June 2006 and June 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 made several cuts and ran data cleaning algorithms to include only the subjects who had completed all programming exercises, in order to clean and 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 [11]. After this cleaning process, the data set was composed of 40 engineers.

Table 3 presents median, interquartile range, and mean of the three variables under study for the three major PSP levels.

Tables 4, 5, and 6 present the results of applying the Wilcoxon test to each pair of PSP levels for the hypothesis of defect density in compile (DDComp), defect density in unit test (DDUT), and total defect density (TDD), respectively. The tables present the comparison between pairs of PSP levels. Each cell contains the (2-tailed) P value of the Wilcoxon test and the effect size of the changes when applicable. The cells in $*$ or $***$ indicate that the null hypothesis has been rejected ($P < 0.05$). The $*$ ones also indicate that there has been an improvement as the subjects advance in the PSP levels; the $***$ ones indicate deterioration. The $**$ cells indicate that it has not been possible to reject the null hypothesis. The effect sizes of the changes are presented based on Cohen's d measure.

The analysis of defects per KLOC in compile phase reveals a statistically significant improvement in defect density between all the PSP level comparisons ($P < 0.05$), all with medium and large effect sizes following Cohen classification [12].

The analysis of defects per KLOC in unit test phase reveals a statistically significant improvement in defect density between all the PSP level comparisons ($P < 0.05$). The effect size of the improvement is large between PSP0 and PSP2.

The analysis of overall defect density reveals that the differences between PSP levels 0 and 1 and between PSP levels 0 and 2 are statistically significant ($P < 0.05$) but that the difference between PSP levels 1 and 2 is not. The effect size

Table 3: Descriptive statistics for the three variables under study.

	PSP0	PSP1	PSP2
	Defect density in compile (number of defects found in compile/KLOC)		
Median	26.67	10.03	0.00
IQR	23.37	17.90	4.76
Mean	29.67	14.63	3.50
	Defect density in unit testing (number of defects found in UT/KLOC)		
Median	18.76	9.93	5.41
IQR	16.29	15.22	14.03
Mean	20.60	12.94	8.04
	Total defect density per KLOC (number of defects found/KLOC)		
Median	48.32	24.73	31.77
IQR	28.67	27.69	37.42
Mean	58.05	29.90	35.64

Table 4: DDComp analysis.

Level	PSP1	PSP2
PSP0	$P = 0.000, d = 0.7^{*}$	$P = 0.000, d = 1.4^{*}$
PSP1		$P = 0.000, d = 1.0^{*}$

Table 5: DDUT analysis.

Level	PSP1	PSP2
PSP0	$P = 0.001, d = 0.5^{*}$	$P = 0.000, d = 1.0^{*}$
PSP1		$P = 0.021, d = 0.4^{*}$

Table 6: TDD analysis.

Level	PSP1	PSP2
PSP0	$P = 0.000, d = 0.9^{*}$	$P = 0.000, d = 0.7^{*}$
PSP1		$P = 0.072^{**}$

of the improvement is large between PSP0 and PSP1 and medium between PSP0 and PSP2.

All these observations support the PSP course benefits regarding product quality. It is important to note that, because of the followed approach, the improvements that were observed represent real change in individual performance, not a change in the average performance of the group. These analyses throw very similar software quality results to those obtained by Hayes, Rombach, and Paulk.

However, as it was stated earlier that, in this kind of study, we cannot affirm that improvements are achieved exclusively by the PSP in itself. There is a possibility that the improvements are achieved due to the programming learning effect produced by the programming repetition during the course.

6. The PSP0.1 Experiment

We designed and performed a controlled experiment at the Universidad de la República. In this experiment, the subjects performed the exercises from the PSP for engineers I/II revised course without applying the PSP techniques.

6.1. Design, Experimental Material, and Subjects. The experimental material is 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 is the same as the one that is used in the PSP for engineers I/II revised courses for the PSP0 and PSP0.1 levels. The tool for data collection is the one distributed by the SEI (PSP support tool developed in Microsoft Access).

The design of this experiment is a repeated measures design. Students develop 8 software programs following an established process. The 8 programs are the same for all the subjects and are developed in the same order. The students use the PSP0 for the first program and the PSP0.1 for the remaining seven programs. These two levels of the PSP only aim at collecting data of the process (time, defects, etc.) but they do not introduce the practices of the PSP (reviews, design, PROBE, etc.). This design of the experiment makes it possible to know if the students improve the performance due to programming repetition.

For our experiment we decided to use 8 program assignments because we consider that 10 assignments (as it has the first PSP course version) could be too much for a student that is always applying the same baseline process, and as a consequence students could lose the interest or the motivation at the last part of the experiment. We also preferred to use this instead of 7 assignments (as it has the last PSP course version) because it better fits the context of a university subject of one semester. Furthermore, 8 programs are enough to analyze whether there are improvements due to programming repetition.

Regarding the environment, the students must perform the assignments individually in their houses, but they are permanently monitored by a tutor. It is a controlled environment because there is a constant feedback with the tutor, unlimited email exchanges to evacuate doubts (sometimes also phone calls or meetings), corrections, and redelivery requests when necessary at the end of each assignment. Students are not in a time-limited classroom, so in this way the time records and the amount of defects found are not biased by the available time of class. That is, the student performs the assignment at their own peace and records their real time and defects without being pressured by the clock or by other students who finish earlier. That allows us to have reliable measures.

A total of 22 subjects performed the PSP0.1 experiment: 12 during the first execution that was performed in 2012 and the other 10 during the second execution in 2013. These executions were conducted in the same way and only the subjects changed.

The subjects are software engineering undergraduate students of the Universidad de la República of Uruguay; all of them advanced students since they are in their fourth or fifth year. They have completed the course Programming Workshop in which they learn Java language and they have at least completed three more programming courses and a course on object oriented languages. We consider

therefore that the group that participates in the experiment is homogeneous due to their similar advance in their career.

Some qualitative analyses have shown that undergraduate students in PSP courses were concerned more with programming than with software process issues and that they were not ready to appreciate the benefits of addressing those issues [13–15]. In these works, the students were applying PSP in introductory programming courses during their first year of university studies. Our experiment is different from that in several aspects. We only use a small part of the PSP framework which does not require new knowledge by students. They must apply their own process just adding discipline aspects to record their data, and they do not have to learn complex techniques applied in PSP such as the PROBE method for time and size estimation, performing neither reviews nor detailed design techniques. Another different aspect is that our students already know how to program and they are advanced students who had previously completed several programming courses. Our students are focused on making the best software development they can and focused on recording the data correctly. During the execution of our experiment we did not identify similar problems to those described by these authors.

The students participate in the experiment in order to obtain credits for their career and that is their motivation. It is mandatory for them to attend the theory classes (lectures) where the software development process used (PSP0 and PSP0.1) is presented. It is also mandatory for them to follow the scripts provided and to collect the data using the tool for that purpose. The students do not know they are taking part in an experiment and they think they are taking a course with an important component of laboratory practices. They do know, however, that the data they collect will be used in research work and they indeed give their written consent for such purpose.

Finally, participation in the course by the students is voluntary. This course is not mandatory for their Software Engineering Degree; therefore enrolling in it is optional.

6.2. Results and Discussion. In this section, we present the quality analysis results of the 22 subjects that performed the experiment, which will allow us to know if quality improves by the simple fact of programming repetition.

For this analysis the labels "progs1-2," "progs3-4," and "progs5–8" represent the program assignment grouping done as in the PSP I/II revised course. That is, progs1-2 is the grouped data of program assignments 1 and 2, progs3-4 is the grouped data of program assignments 3 and 4, and progs5–8 is the grouped data of program assignments 5 to 8.

Table 7 presents median, interquartile range, and mean of the three variables under study for the three groups.

Tables 8, 9, and 10 present the result of applying the Wilcoxon test to each pair of groups and the effect size for the three variables under study. The colors are used in the same way as in Table 4.

The analyses of defects per KLOC in compile phase show a statistically significant improvement when comparing the first two programs with the following programs (i.e. progs1-2 versus progs2-3 and progs5–8). However, there is no improvement after the first programs as it was not possible to reject the null hypothesis when comparing progs3-4 and progs5–8.

This means that programming repetition and data collection of time and defects following the process PSP0.1 reduce the defects found in compile phase with statistical significance. We understand that the most plausible reason for the improvement in the first programs is that the subjects record their own injected defect types found in the programs. By recording each injected defect and data related to them, the engineer becomes aware of the most common defects and reduces the injection or easily finds them before compilation. Nevertheless, further experiments are needed to confirm if this is the reason for the improvement.

Following Cohen classification, the effect size of the improvement is small between progs1-2 and progs3-4 and medium between progs1-2 and progs5–8.

Table 7: Descriptive statistics for the three variables under study.

	Progs1-2	Progs3-4	Progs5–8
	Defect density in compile (number of defects found in Compile/KLOC)		
Median	56.58	39.72	37.83
IQR	75.48	45.32	39.15
Mean	78.21	59.08	44.12
	Defect density in unit testing (number of defects found in UT/KLOC)		
Median	39.10	16.45	19.82
IQR	17.58	20.01	18.30
Mean	54.57	20.13	27.38
	Total defect density per KLOC (number of defects found/KLOC)		
Median	114.40	64.33	66.50
IQR	82.13	51.87	48.87
Mean	134.91	79.96	72.07

Table 8: DDComp analysis.

Group	Progs3-4	Progs5–8
Progs1-2	$P = 0.04, d = 0.3^{*}$	$P = 0.001, d = 0.6^{*}$
Progs3-4		$P = 0.296^{**}$

Table 9: DDUT analysis.

Group	Progs3-4	Progs5–8
Progs1-2	$P = 0.000, d = 0.9^{*}$	$P = 0.001, d = 0.7^{*}$
Progs3-4		$P = 0.012, d = 0.4^{***}$

Table 10: TDD analysis.

Group	Progs3-4	Progs5–8
Progs1-2	$P = 0.000, d = 0.6^{*}$	$P = 0.000, d = 0.7^{*}$
Progs3-4		$P = 0.961^{**}$

The analyses of defects per KLOC in unit test phase reveal a statistically significant improvement between progs1-2 and progs3-4 and between progs1-2 and progs5–8. This could also be due to the defect recording of every injected defect.

Between progs3-4 and progs5–8 there is a significant difference but it refers to a deterioration, which means that programs 5–8 show a higher number of detected defects in unit test than programs 3-4. This shows that programming repetition (using these programs) does not result in a continuous improvement of defect density in unit testing. One possible explanation for this behavior is that defect types that are not recognizable by the compiler (for example, function, data, or checking defects) are not as easy to learn to avoid their injection as the defect types that are detectable at the compile phase (for example, syntax defects). This combined with the increasing complexity of the program assignments tasks could be the reason for the deterioration. There could be many other possible reasons for this deterioration, but further studies are necessary.

According to Cohen classification, the effect sizes of the improvements are large, and the effect size of the deterioration is medium.

The analyses of total defects per KLOC show the same behavior as the analyses of DDComp. A statistically significant improvement was found when comparing the progs1-2 and progs2-3 and progs5–8, and it was not possible to reject the null hypothesis when comparing progs3-4 against progs5–8. This shows that the overall defect density of the developed programs is reduced with statistical significance by the programming repetition and data collection of time and defects following the PSP0.1 process. The effect size of the improvement is large between progs1-2 and progs3-4 and medium between progs1-2 and progs5–8.

All these observations reveal that there is an improvement regarding product quality with the use of PSP0.1, but this improvement is not continuous as it is in the PSP for engineers I/II revised course.

7. Comparative Discussion

In this section we compare the regular PSP course with our experiment. We want to know whether the quality improvement is because of the PSP practices or because of other characteristics.

Figure 2 summarizes the hypotheses tests results and effect sizes that were presented earlier in the PSP I/II revised course and the PSP0.1 experiment. Only the comparisons between progs1-2 against progs3-4 and progs3-4 against progs5–8 are included in the tables. Remember that, in the case of the PSP course, those comparisons refer to PSP0 against PSP1 and PSP1 against PSP2 levels, respectively.

A bar-whisker chart of defect density per group and a bar-whisker chart of individual improvement are also included in the figure comparing the PSP course and the experiment for each analyzed variable. These charts are descriptive and allow us to get a clearer idea of the software quality behavior at the individual level.

The individual improvement is calculated as the percentage difference for each subject between two groups (i.e., *% improvement = − 100 ∗ (defect density in group Y − defect Density in group X)/defect density in group X*). An improvement is represented by a positive value, and deterioration is represented by a negative one. Samples with zero defects as the divisor are not considered in the analysis, as division by zero is not defined. We use a proportional representation of individual improvement instead of an absolute difference because we consider it more appropriate in our context (i.e., it does not seem appropriate to consider a reduction in defect density from 455 to 450 defects per KLOC as equal as a reduction from 10 to 5 defects per KLOC.)

The left side charts indicate that, from the beginning, the PSP course is better than PSP0.1 experiment. The PSP courses are generally performed by professional software engineers with several years of experience in the industry. We believe that this factor is making the initial number of defects lower when comparing with undergraduate students. This factor is known to affect productivity performance [16]. This is not a problem for our study because we are interested in the changes in the individual performance. That is, we analyze if there are improvements when introducing techniques or if there are improvements with programming repetition.

For each analyzed variable, we can see an initial improvement (from progs1-2 to progs3-4) in the PSP course as well as in the PSP0.1 experiment. In the right side charts, this improvement effect is clearly visible. In both cases, the reason for the improvement could be the defect recording activity done since PSP0, as in PSP1 only size and time estimation techniques are introduced and these should not have impact on product quality. For DDComp and TDD the effect size in the PSP course is greater, while for DDUT the effect size is greater in the experiment.

There is no statistical evidence of improvement when comparing progs3-4 to progs5–8 in the PSP0.1 experiment in any of the analyzed variables (even statistical evidence of deterioration exists in DDUT). However, when comparing PSP1 to PSP2 in the PSP course we found a statistical improvement in DDComp and DDUT with an effect size of 1.0 and 0.4, respectively. We can see those improvements in the PSP course graphically represented in the right side charts. Design reviews, code reviews, and detailed design techniques are introduced in PSP2 using four specific design templates and individual tailored checklists. This change in the process is the most plausible reason for the difference in the quality improvement between the PSP0.1 experiment and the PSP course. Therefore, the practices introduced by the PSP (and, so, the PSP itself) lead to such a big improvement in product quality, while programming repetition or defect recording does not.

From the perspective of the practitioner at least two interesting conclusions emerge. One is that the use of the PSP supports the development of quality software. We presented the fact that quality improvements are due to the use of this process and, also, that by using the PSP2 a low defect density in unit testing is reached, even when the engineer is incorporating and learning the process (see Table 5).

Defect density in compile		
Course	Progs1-2 versus progs3-4	Progs3-4 versus progs5–8
PSP0.1 expe.	$d = 0.3^{*}$	—**
PSP I/II rev.	$d = 0.7^{*}$	$d = 1.0^{*}$

Defect density in unit testing		
Course	Progs1-2 versus progs3-4	Progs3-4 versus progs5–8
PSP0.1 expe.	$d = 0.9^{*}$	$d = 0.4^{***}$
PSP I/II rev.	$d = 0.5^{*}$	$d = 0.4^{*}$

Total defect density		
Course	Progs1-2 versus progs3-4	Progs3-4 versus progs5–8
PSP0.1 expe.	$d = 0.6^{*}$	—**
PSP I/II rev.	$d = 0.9^{*}$	—**

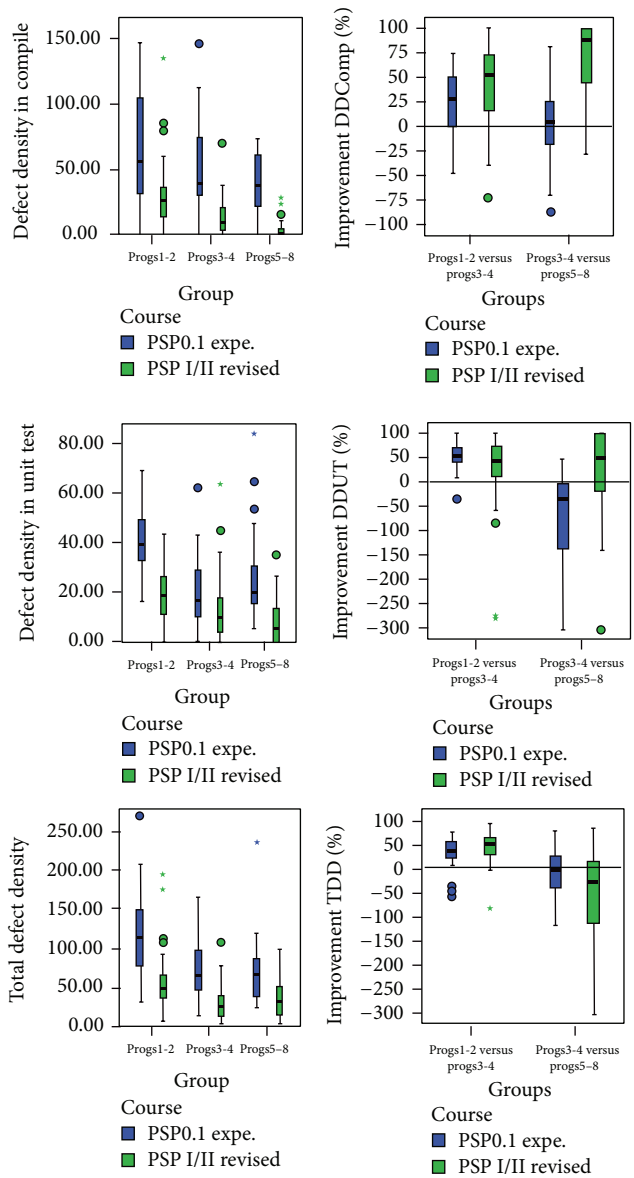

Figure 2: PSP0.1 experiment versus PSP I/II revised course.

The second conclusion from the practitioner's point of view is that the detailed design and the individual design and code reviews are excellent techniques, probably unavoidable, to build quality software. These techniques are the ones added in the PSP2 and they are the ones responsible for the quality improvements detected in our work. Nowadays, with the fashion of agility, developers sometimes do not build detailed designs, let alone reviewing their design or code using checklists.

Still, unfortunately, *the most common approach to software development today is code-and-fix programing* [17], sometimes modernized by the automating of the unit tests. It is time to change.

8. Threats to Validity

Due to space restrictions, we only mention the threats to validity that seem to be most important to us. These include the following: (a) having a small number of subjects performing the experiment, which results in the application of nonparametric tests that are less powerful than parametric ones; (b) all the subjects of the experiment are students with little or no programming experience in the industry; (c) the program assignments were developed at home, which is a threat but is reduced because of the tutor's monitoring during and at the end of each assignment, as it was explained in Section 6.1; and (d) the design method and the design and code review are embedded in the PSP process, so the second practitioner's conclusion is dependent on the PSP.

9. Conclusions and Future Work

The presented results contribute to the elimination of an important threat to the validity of different experiments performed with the PSP. This agrees with previous research works we performed which indicate that the practices introduced by the PSP and not programming repetition are responsible for the improvement of individual performance [6, 18].

In addition, the comparison between our experiment and the regular PSP course reveals that continuous and transcendent product quality improvements cannot be reached simply by the programming learning effect. The use of adequate practices is the cause of the important software quality improvements.

As future work we intend to isolate the PSP techniques (detailed design, design and code review) using a new controlled experiment that will enable us to study the effect of each technique in software quality and the synergy produced between them.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] W. Humphrey, *A Discipline for Software Engineering*, Addison-Wesley, 1995.

[2] W. Hayes and J. Over, "The personal software process: an empirical study of the impact of PSP on individual engineers," Tech. Rep. CMU/SEI-97-TR-001, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa, USA, 1997.

[3] D. Rombach, J. Münch, A. Ocampo, W. S. Humphrey, and D. Burton, "Teaching disciplined software development," *The Journal of Systems and Software*, vol. 81, no. 5, pp. 747–763, 2008.

[4] M. C. Paulk, "Factors affecting personal software quality," *CrossTalk*, vol. 19, no. 3, pp. 9–13, 2006.

[5] M. C. Paulk, "The impact of process discipline on personal software quality and productivity," *ASQ Software Quality Professional*, vol. 12, no. 2, pp. 15–19, 2010.

[6] D. Vallespir, F. Grazioli, L. Pérez, and S. Moreno, "Demonstrating the impact of the PSP on software quality and effort: eliminating the programming learning," in *TSP Symposium*, Software Engineering Institute, Carnegie Mellon University, Dallas, Tex, USA, 2013.

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

[8] "Transition guide for the PSP for engineers course," Internal Document, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa, USA, 2005.

[9] *Transition Guide PSP for Engineers to PSP Fundamentals and PSP Advanced*, Internal Document, Software Engineering Institute, Carnegie Mellon University, 2008.

[10] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[11] C. Valverde, F. Grazioli, and D. Vallespir, "Un estudio de la calidad de los datos recolectados durante el uso del personal software process," in *Proceedings of the 9th Jornadas Iberoamericanas de Ingeniería de Software e Ingeniería del Conocimiento (JIISIC '12)*, pp. 37–44, November 2012.

[12] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*, Lawrence Earlbaum Associates, Hillsdale, NJ, USA, 2nd edition, 1988.

[13] P. Runeson, "Experiences from teaching PSP for freshmen," in *Proceedings of the 14th Conference on Software Engineering Education and Training*, pp. 98–107, February 2001.

[14] S. K. Lisack, "The personal software process in the classroom: student reactions (an experience report)," in *Proceedings of the 13th Conference on Software Engineering Education and Training*, pp. 169–175, 2000.

[15] R. Grove, "Using the personal software process to motivate good programming practices," in *Proceedings of the 6th Annual Conference on the Teaching of Computing and the 3rd Annual Conference on Integrating Technology into Computer Science Education (ITICSE '98)*, pp. 98–101, Dublin, Ireland, September 1998.

[16] R. Mushtaq, F. Joao, and N. William, "Factors affecting productivity performance in PSP training," in *Proceedings of the Team Software Process Symposium (TSP '13)*, Software Engineering Institute, Carnegie Mellon, Pittsburgh, Pa, USA, 2013.

[17] S. McConell and L. L. Tripp, "Software engineering as a profession," in *Software Engineering Essentials, Volume II: The*

Supporting Process, R. H. Thayer and M. Dorfman, Eds., chapter 11, pp. 159–164, 2013.

[18] F. Grazioli and W. Nichols, "A cross course analysis of product quality improvement with PSP," Tech. Rep. CMU/SEI-2012-SR-015: 76–89, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa, USA, 2012, TSP Symposium.