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 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 twelve 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.

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 97, Rombach 08]. The students (many times 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 during the course by the students 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 compares 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) concludes that the changes in quality are most plausible regarding mastering PSP techniques rather than programming repetition [Grazioli 12].

Our work aims at contributing in this same direction but using 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 twelve 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 not an 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 contributed to the performance improvements.

2 EXPERIMENT SETUP

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

2.1 Goals, Metrics and Hypotheses

The goal of our experiment is to know if the software engineers improve their performance when they develop the programs used in the PSP course due to programming repetition in the same application domain. The aspects of performance that are considered are quality of the product and the effort required in unit testing.

In order to know the quality of the products we use 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 is measured as the number of defects per every thousand lines of code (KLOC). The effort used in unit testing is measured also 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 non-random cause. The aim of the hypothesis test is to determine whether it is possible to reject the null hypothesis H0 [Juristo 01].

The experiment raises the null hypotheses and their respective alternative hypotheses for each of the four mentioned metrics. The hypotheses aim at knowing if comparing a developed program to another one developed previously, the software engineer improves his performance in any of the aspects mentioned.

So, we compare programs by pairs to find if the changes in each performance 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 programs (1 to 8) and i < j

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

2.2 Subjects

The subjects of the experiment are Computer Science undergraduate students of the Universidad de la República of Uruguay, all of them advanced students since they are in 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 advancement in the career.

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, 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 even gave their written consent for it.

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

2.3 Experimental Material

The experimental material is made up by 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 exactly the same as the one that is used in the PSP for Engineers I/II courses (in the 8-program version). The tool for data collection is the one distributed by the SEI (PSP support tool developed in Microsoft Access).

2.4 Experimental Design

The design of this experiment is a repeated measures design. Twelve students develop 8 software programs following an established process. The 8 programs are the same for the 12 subjects and are developed in the same order. These programs, as it has already been mentioned, are the ones used in the PSP for Engineers I/II course.

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.

We refine our goal using the GQM approach [Basili 94] as:
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 twelve undergraduate students.

3 RESULTS AND DISCUSSION

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

Table 1 – Median and interquartile range 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	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

The students of our experiment are 12 (few samples) and the data of each one in the 8 exercises of the PSP are 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 45]. 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 non-parametric test equivalent to the dependent t-test. We used the 2-tailed Wilcoxon test because we do not know a priori if the dependent variables will increase or reduce their values.

Table 2 presents the result 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 (2-tailed) of the Wilcoxon test. The cells in green or red indicate that the null hypothesis has been rejected ($p<=0.05$). The green ones also indicate that there has been an improvement in defect density in UT as the students advance in the exercises; the red ones indicate the opposite. The grey cells indicate that it has not been 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 in developing a program to count lines of code of a program. Although this can be a cause for a higher defect density, we cannot assure so.

Table 2 – Wilcoxon test for DDUT

Prog.	**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

It can also be observed that in program 5 the defect density in UT is statistically higher than the one 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 exercise 2 from the analysis, no difference can be detected between exercise 3 and the following, or between exercise 4 and the following, or 6 and the two following, neither between exercises 7 and 8. The differences found between exercises 5 and 6, and between exercises 5 and 8 may be due to the characteristics of exercise 5. However, other experiments are necessary to prove it. This is different from the improvements found when the regular course is used [Hayes 97, Rombach 08].

Table 3 presents the result 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

Prog.	**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 a higher number 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 record 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 fewer number of injected defects.

Table 4 presents the result 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 grey indicates that the null hypothesis could not be rejected.

Table 4 – Wilcoxon test for TSUT per KLOC

Prog.	**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

As it can be observed, in this case there is not a steady improvement in the performance either. In this case 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 also 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 result 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 3.

Table 5 – Wilcoxon test for average TSUT per defect

Prog.	**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 the one used in programs 1 to 5. Therefore, the results show that in the last programs the efficiency of UT (defects found per unit of time) decrease. This can be due to several reasons: less number of 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 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 here. On the other hand, the effort per KLOC in UT even decreases for some pairs of programs that appear in red here. 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.
- Seems to improve in the last 3 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.
- It seems to deteriorate the efficiency of UT.

4 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 result agree with a previous one [Grazioli 12], which indicates that the practices introduced by the PSP and not programming repetition would contribute to the improvement of individual performance. Moreover, as both studies show the same kind of results by following different approaches, the confidence of what is being conclude increases. Besides, it is 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 performing new controlled experiments.

Besides, this experiment shows that without the adequate practices the quality of software and the performance of the process cannot be improved by the simple reason of the programming learning effect. Someone once said, "Insanity is when you keep doing the same things 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 which generate the improvements in the performance of the software engineer.

[1] This quote or a variant of the same is attributed to different persons, among them, Albert Einstein, Rudyard Kipling, Rita Mae Brown, Benjamin Franklin and a Chinese proverb. I could not find out who the real author of that phrase is

Our future work is to 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.

5 REFERENCES/BIBLIOGRAPHY

[Basili 94]
Basili, Victor; Caldiera, Gianluigi; Rombach Dieter: The Goal Question Metric Approach, in Encyclopedia of Software Engineering (John J. Marciniak, Ed.), John Wiley & Sons, Inc., Vol. 1, pp.528-532, 1994.

[Grazioli 12]
Grazioli, Fernanda; Nichols, William; A Cross Course Analysis of Product Quality Improvement with PSP. TSP Symposium 2012 Proceedings, Special Report, Software Engineering Institute, Carnegie Mellon, CMU/SEI-2012-SR-015: 76-89, 2012.

[Hayes 97]
Hayes, Will; Over, James; The Personal Software Process: An Empirical Study of the Impact of PSP on Individual Engineers. Technical Report, Software Engineering Institute, Carnegie Mellon University, CMU/SEI-97-TR-001, 1997.

[Juristo 01]
Juristo, Natalia; Moreno, Ana M.; Basics of Software Engineering Experimentation. Kluwer Academic Publishers, 2001.

[Rombach 08]
Rombach, Dieter; Munch, Jurgen; Ocampo, Alexis; Humphrey, Watts S.; Burton, Dan; Teaching Disciplined Software Development. The Journal of Systems and Software 81, (5): 747-763, 2008.

[Wilcoxon 45]
Wilcoxon, Frank; Individual comparisons by ranking methods. Biometrics Bulletin 1 (6): 80-83, 1945.