PEDECIBA Informática
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Tesis de Maestría
en Informática

An analysis of student performance during the introduction of PSP : An empirical cross course comparison

Fernanda Grazioli

2013

Fernanda Grazioli
An analysis of student performance during
The introduction of the PSP : an empirical
Cross course comparison
ISSN 0797-6410
Tesis de Maestría en Informática
Reporte Técnico RT 13-07
PEDECIBA
Instituto de Computación – Facultad de Ingeniería
Universidad de la República.
Montevideo, Uruguay, 2013

Tesis de Maestría

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

Fernanda Grazioli

Director de Tesis: William Nichols

Director de Estudios: Héctor Cancela

Programa de Maestría en Informática

Junio de 2013

UNIVERSIDAD DE LA REPÚBLICA
FACULTAD DE INGENIERÍA

El tribunal docente integrado por los abajo firmantes aprueba la Tesis de Investigación:

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

Autor: Ing. María Fernanda Grazioli Pita
Director de Tesis: Dr. William Nichols
Director Académico: Dr. Héctor Cancela
Carrera: Maestría en Informática - PEDECIBA
Calificación: _____

TRIBUNAL

Dra. Juliana Herbert (Revisora) _____

Dra. Cristina Cornes _____

M.Sc. Omar Viera _____

Montevideo, _____

Agradecimientos [Acknowledgements]

Quiero agradecer primero a las personas del Grupo de Ingeniería de Software de la UdelaR que trabajaron conmigo y me ayudaron en distintas oportunidades: Cecilia Apa, Lucía Camilloni, Silvana Moreno, Leticia Pérez, Sebastián Pizard, Rosana Robaina y Carolina Valverde. Agradezco especialmente a Diego Vallespir por confiar en mí para este proyecto, motivarme y alentarme en todo momento. Su gran dedicación, experiencia y su apoyo durante estos meses de trabajo hicieron posible esta tesis.

A Gabriela Mathieu y a Jim McCurley por las discusiones sobre los distintos métodos estadísticos posibles para este trabajo.

A Marina Melani y Alejandra Baccino por las correcciones de la escritura en inglés.

Agradezco a Adriana Marotta, quien me guió durante los cursos de la maestría y estuvo siempre a disposición.

A William Nichols por ser el tutor y guía de este trabajo de tesis y a Héctor Cancela por ser mi director de estudios. Pude sentir en todo momento la confianza que tuvieron en mí y su apoyo constante.

Agradezco a mi hija Agustina y a Fernando por aguantar y entender.

Resumen en Español

Hoy en día, cada vez más empresas desarrollan, combinan e incluyen software de distintas formas en sus productos. Además, muchos de los sistemas de los que nuestras vidas y medios de vida dependen, son administrados por software. Como los componentes de software de casi cualquier producto crecen y se vuelven cada vez más complejos, los retrasos en el cronograma, los sobre-costos y los problemas de calidad ocasionados por el desarrollo de software se están convirtiendo en uno de los principales problemas que deben afrontar las empresas. Los productos de software van desde cientos a millones de líneas de código, donde cada línea es escrita por un ingeniero de software. Las empresas de software dependen de las personas, por lo que sus técnicas y su experiencia son determinantes para el resultado del proceso de desarrollo.

El Personal Software Process (PSP) es un proceso de desarrollo de software definido, medible, y diseñado para ser utilizado por un ingeniero de software de forma individual. El PSP aborda directamente las necesidades de las empresas de software mediante la mejora de las técnicas y las habilidades individuales de los ingenieros de software, proporcionando además una base cuantitativa para la gestión del proceso de desarrollo. Al mejorar el desempeño individual, el PSP puede mejorar el rendimiento de la organización.

Durante muchos años, el Software Engineering Institute (SEI) de la Universidad Carnegie Mellon ha capacitado a ingenieros de software en el Personal Software Process. Los cursos de PSP han cambiado a lo largo de los años. Varias versiones del curso utilizan los mismos ejercicios de programación, pero introducen las fases del proceso y las técnicas en momentos diferentes. La primera versión del curso cuenta con varios estudios publicados que demuestran una mejora en el desempeño del desarrollador luego de la inserción del proceso, pero el análisis retrospectivo deja algunas amenazas a la validez externa. Dado que los programas del curso son en un mismo dominio de aplicación, una amenaza que surge es que la mejora podría ser causada por la repetición de la programación y no necesariamente por la introducción de técnicas y de fases del proceso. Por lo tanto, la pregunta es si las mejoras se deben a las fases y técnicas o debido a la repetición de programación durante el curso (efecto de aprendizaje por repetición).

Dada esta situación, el objetivo principal de esta tesis consiste en utilizar los datos de las últimas dos versiones de cursos de PSP para determinar si las diferentes técnicas introducidas mejoran aspectos del desempeño de los desarrolladores, o si esa mejora surge como consecuencia de ganar experiencia en el dominio de aplicación. El segundo objetivo es documentar observaciones y resultados de estas dos versiones más recientes, las cuales aún no tienen estudios publicados.

Analizamos el desempeño de 347 ingenieros durante el entrenamiento del PSP con respecto a cuatro dimensiones: la densidad de defectos en pruebas unitarias, el *yield* del proceso, la tasa de producción, y la precisión en la estimación de tamaño. Utilizamos los datos de los cursos de PSP que fueron dictados entre junio de 2006 y junio de 2010. Estos cursos fueron brindados por el SEI o por socios (*partners*) del SEI.

Respecto a las observaciones de los cursos, encontramos una reducción en la media de la densidad de defectos en pruebas unitarias de un factor de 2.3, un aumento en la media del *yield* del proceso en un factor de 1.9 y una reducción en la media de la precisión de la estimación de tamaño en un factor de 2.6 . Estos tres resultados muestran mejoras significativas, que siguen la misma línea que los resultados publicados del curso anterior. También encontramos una reducción en la media de la tasa de producción en un factor de 0.7, lo que difiere de los estudios anteriores y muestra un deterioro en la productividad de los ingenieros.

Además, nuestros resultados sugieren que las mejoras en la densidad de defectos en pruebas unitarias y en el *yield* del proceso son más plausibles por las técnicas del PSP que por la repetición de la programación. Para la precisión en la estimación de tamaño y para la tasa de producción no pudimos descartar por completo el efecto del aprendizaje en el dominio de aplicación como la principal causa de los cambios. Sin embargo, los resultados nos llevan a pensar que las fases del proceso son probablemente una de las principales razones de los cambios. Como trabajo futuro proponemos varios experimentos controlados, los que permitirán evaluar en profundidad los efectos de la repetición de la programación.

Abstract

Nowadays, more and more businesses develop, combine and include software in their products in different ways. Also, many of the systems, on which our lives and livelihoods depend, are run by software. As the software component of almost any product grows and gets more complex, schedule delays, cost overruns, and quality problems caused by software development are becoming a main problem for business. Software products are made of hundreds to millions of lines of code, each one handcrafted by a software engineer. Software businesses depend on people, so their technical practices and their experience strongly determine the outcome of the development process.

The Personal Software Process (PSP) is a defined and measured software process designed to be used by an individual software engineer. The PSP directly addresses the software businesses needs by improving the technical practices and individual abilities of software engineers, and by providing a quantitative basis for managing the development process. By improving individual performance, PSP can improve the performance of the organization.

For many years, the Software Engineering Institute (SEI), of the Carnegie Mellon University has trained software engineers in the Personal Software Process. The PSP courses have changed throughout the years. Several versions of the course use the same programming exercises, but introduce process phases and techniques in modified sequences. An earlier version of the course has several published studies demonstrating improvement in developer performance with process insertion, but the retrospective analysis 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 the programming repetition during the course (learning effect).

Given this known problem, the main goal of this thesis 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 yet have published studies.

We analyzed the performance of 347 engineers during the PSP training with respect to four performance dimensions: defect density in unit testing, yield, production rate and size estimation accuracy. We used data from PSP courses taught between June 2006 and June 2010. These courses were taught by the SEI or by SEI partners.

Regarding the courses observations, considering both courses we find out that the mean reduction in defect density in unit testing is a factor of 2.3, the mean increase in yield is a factor of 1.9 and the mean reduction in size estimation accuracy is a factor of 2.6. All these three results reveal significant improvements, which follow the same line of the earlier course published results. We also find out that the mean reduction in

production rate is a factor of 0.7, which differs from the previous studies and reveals a deterioration of the engineers' rate.

Also, our results suggest that improvements in defect density in unit testing and yield are most plausible regarding mastering PSP techniques rather than programming repetition. For size estimation accuracy and production rate we were not able to fully discard the domain learning effect as the root causes of the changes; however the results leads us to think that the process phases are probably one of the main reasons of the changes. As future work we propose several controlled experiments, which will allow evaluating the programming repetition effects in depth.

Contents

Chapters

Appendices

Chapter 1

Introduction

Nowadays, more and more businesses develop, combine and include software in their products in different ways. Companies need to develop software to support the design, manufacture or delivery of the products and services they provide. Therefore, without noticing, all businesses are becoming software businesses. As the software component of their business grows, schedule delays, cost overruns, and quality problems caused by software development are becoming their main business problem. This is why despite their best management efforts, their risk of failing increases along with the increase in the size or complexity of the software they produce [1].

In this context, software business needs improved software quality, better cost and schedule management as well as reduced software development cycle time. To reach these needs, it is becoming increasingly important to adopt processes and techniques which allow to improve software quality, do better estimations and keep software development under control. Also, if we consider that *"... many of the systems on which our lives and livelihoods depend are run by software"* [2], know whether the processes and the applied techniques lead to develop high quality products is of vital importance.

Software products are made of hundreds to millions of lines of code, each one handcrafted by a software engineer. Software businesses depend on people, so their technical practice and their experience strongly determine the outcome of the development process.

The Personal Software Process (PSP) is a defined and measured software process designed to be used by an individual software engineer. The PSP directly addresses the software businesses needs by improving the technical practices and individual abilities of software engineers, and by providing a quantitative basis for managing the development process. By improving individual performance[1] PSP can improve the performance of the software development team and the organization [3].

The Team Software Process (TSP) is a software development process for teams that satisfies the aforementioned software business needs and which uses the PSP for each team member [4] [5]. The TSP is used in the software industry and a recent book by Caper Jones, in an independent evaluation, indicates that it is the best software development process for medium and large scale projects [6]. A large percentage of the TSP practices take place at an individual level, that is, practices that arise during the use of the PSP.

---

[1] The term "performance" covers several aspects, such as improve the quality of the produced product, produce better estimations, and increase the code production rate, among others. It should not be confused with productivity.

Therefore, is important to know whether the processes and the applied techniques of the PSP lead to develop high quality products. The main topic of this thesis is to know if the different techniques and phases of the PSP (and so, the PSP itself) produce positive changes in the aforementioned aspects of the software development.

1. Context and Motivation

"*The Personal Software Process is a self-improvement process that helps you to control, manage, and improve the way you work."* - W. S. Humphrey, 2005.

The PSP is a software development process for the individual. The process helps the engineer to control, manage and improve his or her work. The PSP establishes a highly instrumented development process that includes a rigorous measurement framework for effort and defects. This process includes phases that the engineer completes while building the software.

For each software development phase, the PSP has scripts that help the software engineer to follow the process correctly. The phases include Planning, Detailed Design, Detailed Design Review, Code, Code Review, Compile, Unit Test, and Post Mortem. For each phase, the engineer collects data on the time spent in the development phase and the defects injected and removed, as is shown in Figure 1.

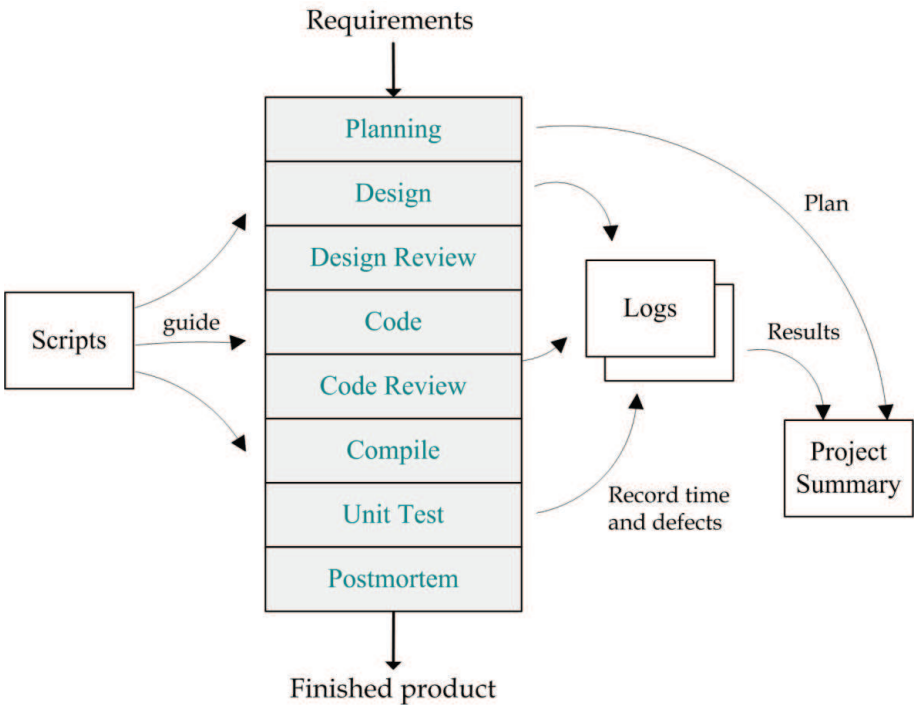


Figure 1: The PSP phases, scripts, logs and project summary

The PSP is taught through a course. There are three different course versions. During the course, the engineer builds programs while progressively learning PSP planning, development, and process assessment practices. For the first exercise, the engineer starts with a simple and defined process. As the class progresses, new process phases and elements are added, from Estimation and Planning to Code Reviews, 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 2. By gradually adding techniques, the developer is able to analyze the impact of the new techniques on his or her individual performance. The PSP is in fact the PSP2.1 level. The other PSP levels exist exclusively for the purposes of the teaching the PSP.

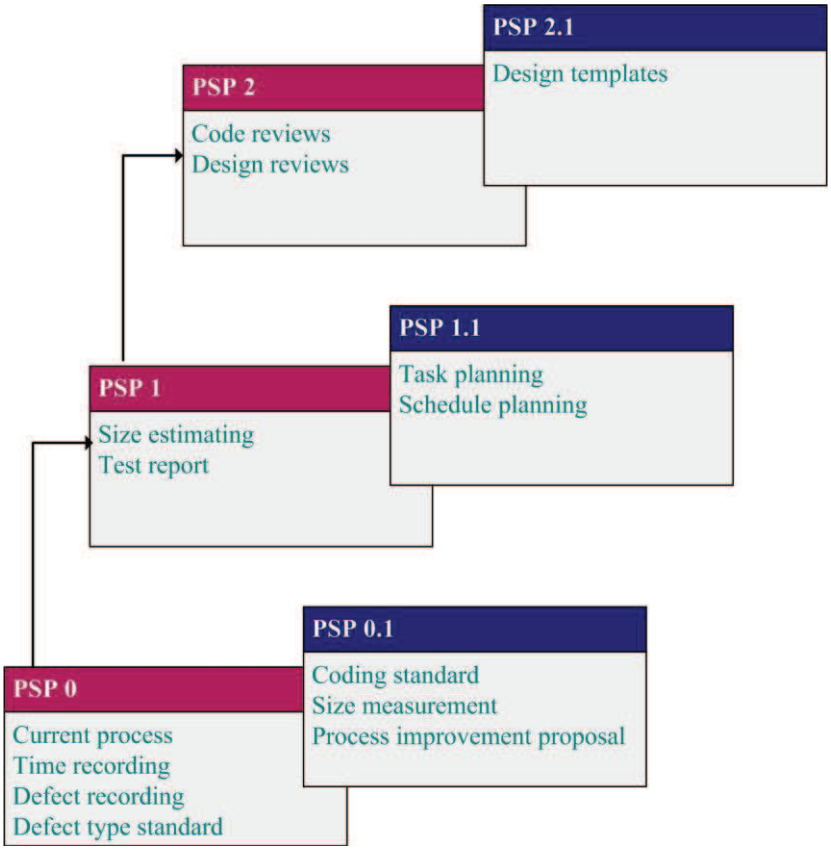


Figure 2 : The PSP Process Levels

In the last 20 years, thousands of software engineers attended the different courses and were trained in the techniques of the PSP. The PSP courses are taught by the Software Engineering Institute (SEI), of the Carnegie Mellon University, or by SEI partners. A systematic data collection is conducted for all training participants. Data from which learning effectiveness and quality improvements of the software being developed can be derived.

Data of the first version of the PSP course show that PSP can improve the business of software development in several ways:

- Data from the PSP improve planning and tracking of software projects.
- Early defect removal results in higher quality products, as well as reductions in test costs and cycle time.
- PSP provides a classroom setting for learning and practicing process improvement. Short feedback cycles and personal data make it easier to gain understanding through experience.
- PSP helps engineers and their managers learn how to practice quantitative process management. They learn to use defined processes and collect data to manage, control, and improve the work.

Several versions of the course use the same programming exercises, but introduce process phases and techniques in modified sequences. An earlier version of the course has several published studies demonstrating improvement in developer performance with process insertion [7] [8] [9] [10] [11] [12], 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. Therefore, the question is if the improvements are due to the phases and techniques or due to the programming repetition during the course.

To clarify the threat to validity mentioned above, we present a simple example. Suppose that when studying the PSP course data through a statistical analysis we find a significant improvement in some aspect of the engineers' performance. Looking in detail, we see that such improvement is happening between the fourth and the fifth program. If in that course, a PSP phase or a new technique was introduced in the fifth program (i.e. a PSP level change has occurred between programs 4 and 5), then we might think that that PSP level change is the responsible for the improvement on the engineers' performance. However, it may happen that the improvement is occurring because the engineers tend to improve that particular aspect of the performance in the fifth program due to the programming repetition, independently if a technique (or a PSP phase) is introduced or not in that assignment.

It is important to dispose of this threat to validity and be able to know if the introduction of the phases and the techniques are the root cause of the improvements of the engineers' performance. It is essential to know that the PSP and the techniques that are applied during the process are the ones that produce these improvements in order to provide research opportunities to the scientific community of software engineering and software development companies as well as supply opportunities to adopt these techniques and acquire best software products in controlled costs, on schedule and with the desired quality.

2. Problem and Goals

Based on studies of the first PSP course version, the community assumes that current PSP courses and the PSP are working due to the specific techniques introduced, and that the process insertion has positive and substantial benefits.

Several empirical studies on the effects of PSP were published during the last 15 years, among others by Hayes et al. in 1997 [7] and by Rombach et al. in 2007 [8]. Hayes identified five hypotheses for validating the effectiveness of the PSP based on the first course version data. In 2007 Rombach, with a larger data set of the same course version, re-analyzed Hayes' hypotheses and suggested an additional hypothesis. For some hypotheses, their findings confirm each other while other hypotheses are answered in contradictory ways.

Something that has not yet been extensively studied is the effect of the exercise repetition during the course. We believe that gaining experience in the problem domain could not yet be discarded as one of the reason of the improvements. It is necessary to analyze the data of the courses in order to separate the different technique introduction effects from the domain learning effects. In this way, we will be able to evaluate the main reasons of improvements, and also make a contribution by giving more tools to the community and the industry to evaluate the cost and benefits of using PSP.

Given this known problem (validity threat to prior experiments in PSP), the main goal of this thesis 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. More specifically, we intend to conduct an empirical investigation of the data collected during the courses to evaluate the effectiveness of the techniques and phases that are involved in the process, and the impact of programming repetition during the course. A secondary goal is to document observations and results of the two recent course versions, which do not have yet published works.

For our thesis, we decided to base our work in the aforementioned studies of Hayes and Rombach, and evaluate the effects of the last two PSP course versions by selecting a group of four hypotheses. But in our study we focus on determining the main reason of the improvements and not just evaluating the effect size[2] of the improvements.

Therefore, we defined the particular goals of this thesis as:

- 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 ***defect density in unit testing / yield / production rate / size estimation accuracy*** from the viewpoint of a researcher in the context of the PSP training of engineers in "PSP for Engineers I/II revised" course and the training of engineers in "PSP Fundamentals and Advance" 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 in the problem domain.

3. Proposal and Development

Based on the previous studies of the effectiveness of PSP we defined our particular goals. We define one hypothesis test for each of the variables under study (defect density in unit testing, yield, production rate and size estimation accuracy).

So as to reach our particular goals it was necessary to carry out statistical analysis of the data collected in the last two PSP courses versions, the SEI provided us all necessary data from PSP classes, with appropriate precautions to protect the students' privacy.

Even knowing that the PSP brings a tool for each student to collect the process data during the assignments, it could be of poor quality. We consider that it was important to find a way to ensure that the statistical analyses were based on quality data. Therefore, in order to reach this, we developed an integrated data storage model. We designed the model in order to support the analysis and the assessment of data quality, based on the data quality theory [13].

The first step towards identifying quality problems was to understand the reality and context to analyze. This includes the PSP in itself, exploring the tool the students use for recording their data and the model of the database. Afterwards, we analyzed the dimensions and quality factors of this set of data, which are interesting to measure and

[2] An effect size is a measure of the strength of a phenomenon. This is explained in detail in Chapter 5, Section 2.1.

consider. In this way, we thoroughly identified and defined possible quality problems that the data under study might contain, we implemented the algorithms required for measuring, cleaning and collecting the metadata and executed those algorithms afterwards. After the data cleaning process we obtain a data set with the necessary quality for our statistical analyses. Our proposal to assess the quality of the collected data and the cleaning procedure are presented in Chapter 4and in Appendix 3.

Differences in performance between engineers are typically the greatest source of variability in software engineering research, and this study is no exception. However, the design of the PSP training class and the standardization of each engineer's measurement practice allow the use of statistical models which are well suited for dealing with the variation among engineers.

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 engineers performed better than others from the first assignment, and some improved faster than others during the course. In order to discover the pattern of improvement in the presence of these natural differences between engineers, the statistical method known as the repeated measures analysis of variance (ANOVA for repeated measures) is used [14]. Briefly, the repeated measures analysis of variance takes advantage of situations where the same subject (in this case the students) are measured over a succession of trials. By treating previous trials as baselines, the differences in measures across trials (rather than the measures themselves) are analyzed to uncover trends across the data. This allows for differences among baselines to be factored out of the analysis. In addition, the different rates of improvement between people can be viewed more clearly. If the majority of people change substantially (relative to their own baselines), the statistical test will reveal this pattern. If only a few people improve in performance, the statistical test is not likely to suggest a statistically significant difference, no matter how large the improvement of these few people may be.

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

The first step tries to find out whether are there differences between the two courses by comparing the variable under study for each program assignment. If there are significant differences in the dependent variable with the exercise sequence but without change in the PSP level, then the PSP level cannot be the root cause of the differences in the variable under study. On the other hand, when we find differences through the exercises with different PSP level, 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. The second step looks at each course separately, and tries to find if the differences between the course programs assignments are taking place when the PSP level has changed or if the differences are taking place even when the PSP level has not changed between two assignments. If there are significant changes between programs assignments with the same PSP level, this can lead us to think that the effects on the dependent variable are due to the repetition of exercises and not due to a specific

technique introduction. Otherwise, if the significant changes are only between programs assignments with different PSP level, 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.

The third and last step looks at each course separately again, and tries to find if the differences between the PSP levels are taking place 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 be showing that the technique introduced is the factor affecting the introduced engineers' performance and not the program repetition. The hypothesis test, the design of the experiment and the proposed analysis method are detailed in Chapter 5.

4. Results

In order to determine if there are improvements in the individual engineers' performance through the courses, the data of the last two course version were analyzed, both separately and together. This analysis corresponds to our secondary particular goal. When performing the analysis we found:

- A significant improvement in defect density in unit testing with a mean reduction of a factor of 2.3
- A significant improvement in process yield with a mean increase of a factor of 1.9
- A significant deterioration in production rate with a mean reduction of a factor of 0.7
- A significant improvement in size estimation accuracy with a mean reduction of a factor of 2.6

Our findings in regards to defect density in unit testing and yield are consistent with both Hayes and Rombach findings. Regarding to production rate, our findings differ from both previous studies, as Hayes findings reveals no gain or loss while Rombach findings reveals an improvement. On the other hand, our findings regarding size estimation accuracy are consistent with Hayes findings, but differ from Rombach's.

In order to reach our main goal, in regards to the study of the programming repetition effects and the effects of the phases and techniques introductions, we followed the three step analysis approach that we defined for each hypothesis to determine the main reason of the changes. Our results suggest that:

- The improvements in defect density in unit testing are most plausible regarding mastering PSP techniques rather than programming repetition
- That design and code reviews techniques are the main reason of the improvements in process yield rather than the learning effect.
- We cannot affirm that the PSP level is the main reason of the production rate changes, although 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
- We were not able to discard the domain learning effect as the root causes of the size estimation accuracy improvements, as the estimation

technique introduced in the PSP courses is based on historical data and needs repetition.

The results for both, the first and the second objective of this thesis are presented in Chapter 5.

5. Publications

During the thesis, two articles were published. The first one was accepted for the proceedings of the TSP Symposium 2012 and included in a SEI Special Report. The other was accepted and presented in the IX Jornadas Iberoamericanas de Ingeniería de Software e Ingeniería del Conocimiento (Iberoamerican Conference in Software Engineering and Knowledge Engineering), 2012. Both articles are included in Appendix 6.

- ***A Cross Course Analysis of Product Quality Improvement with PSP***
  Fernanda Grazioli, William Nichols.
  Proceedings TSP Symposium 2012: Delivering agility with discipline (Special Report Software Engineering Institute, Carnegie Mellon University, CMU/SEI-2012-SR-015), pp.76—89, Saint Petersburg, Florida, EEUU, September 2012.

- ***Un Estudio de la Calidad de los Datos Recolectados durante el Uso del Personal Software Process (An Study of the Quality of the Data Collected During the Use of the Personal Software Process)***
  Carolina Valverde, Fernanda Grazioli, Diego Vallespir
  IX Jornadas Iberoamericanas de Ingeniería de Software e Ingeniería del Conocimiento (JIISIC), pp. 37—44, Lima, Perú, Noviembre de 2012.

6. Document Structure

This document consists of this introduction and six more chapters. The chapter *"Background on Software Quality and Planning using the PSP"* presents the necessary background in software quality and PSP in order to understand the thesis work.

The chapter *"Background in Empirical Software Engineering and Statistical Analysis Methods"* contains fundamental terms and concepts of Empirical Software Engineering (ESE). It also presents the statistical data analysis methods that were used as a basis for the approaches applied in this study.

In the chapter *"Data Quality in the PSP"* the data quality theory is introduced, and presents the impact of data quality in ESE. The data error analysis that was done for the available data, the applied methodology and the error types that were defined for this study are explained. A brief summary of the metric definitions, measures and data cleaning for the data cut-offs is also included.

The *"Data Analysis"* chapter presents the analysis, results, threats to validity and conclusions for each of the four hypotheses studied. Also an explanation of the data set and an indirect statistical method proposed for the analysis are presented.

The chapter *"Related Work"* presents several empirical studies on the effects of PSP and their results are compared against the results that were obtained in this work.

Finally, in the chapter *"Conclusions and Future Work"* the conclusions, the contributions of the research, as well as the future work are presented.

Chapter 2

Background on Software Quality and Planning using the PSP

There are models and processes that seek to improve the quality of software products. Many of these models have been used successfully in the industry. The Personal Software Process is one of them. This chapter introduces the concept of software quality, in order to understand the need for these models at different organizational levels. Then, the PSP is presented in detail in order to know its principles, phases, techniques and metrics. Since the PSP is taught through the courses, and it is during the courses that the data analyzed in this work is collected, in the final section the different formats for the existing courses are presented. It is essential to understand and know the PSP, as well as knowing how the different techniques are introduced during the several courses, in order to define how to evaluate the PSP application process.

1. Software Quality

Until shortly after World War II, the quality strategy in most industrial organizations was based almost entirely on testing. Groups would typically establish special quality departments to detect and fix problems after products had been produced. It was not until the 1970s and 1980s that W. Edwards Deming and J.M. Juran convinced the U.S. industry to focus on improving the way people did their jobs [15] [16]. In the succeeding years, this focus on working processes has been responsible for major improvements in the quality of automobiles, electronics, or almost any other kind of product. The traditional test-and-fix strategy is now recognized as expensive, time-consuming, and ineffective for engineering and manufacturing work.

Watts Humphrey said that even though most industrial organizations have now adopted modern quality principles, the software community has continued to rely on testing as the principal quality management method [17]. For software, the first major step was taken by Michael Fagan when in 1976 he introduced software inspections [18] [19]. By using inspections, organizations have substantially improved software quality. Another significant step in software quality improvement was taken with the initial introduction of the Capability Maturity Model (CMM) for software in 1987 [20] [21]. The CMM's principal focus was on the management system and the support and assistance provided to the development engineers. The CMM has had a substantial positive effect on the performance of software organizations [22]. Later in time, the Capability Maturity Model Integration (CMMI) project was formed to sort out the problem of using multiple models for software development processes, thus the CMMI

model has superseded the CMM model, though the CMM model continues to be a general theoretical process capability model used in the public domain.

A further significant step in software quality improvement was taken with the Personal Software Process (PSP) [3]. The PSP extends the improvement process to the people who actually do the work—the practicing engineers. The PSP concentrates on the work practices of the individual engineers. The principle behind the PSP is that to produce quality software systems, every engineer who works on the system must do quality work.

The PSP is designed to help software professionals to consistently use sound engineering practices. It shows them how to plan and track their work, use a defined and measured process, establish measurable goals, and track performance against these goals. The PSP shows engineers how to manage quality from the beginning of the job, how to analyze the results of each job, and how to use the results to improve the process for the next project.

Following the PSP, a further important step in software process improvement was the introduction of the Team Software Process (TSP) [23]. The TSP provides a disciplined context for engineering work. The principal motivator for the development of the TSP was the conviction that engineering teams can do extraordinary work, but only if they are properly formed, suitably trained, staffed with skilled members, and effectively led. The objective of the TSP is to build and guide such teams.

The following section presents the Personal Software Process created by the Software Engineering Institute (SEI), as this process is the one that is used in the frame of this thesis. For completeness, other models and processes created by the SEI are presented, as the CMMI and the TSP in Appendix 1. All these models and processes that seek to improve the quality of software development, have been successfully used in the industry.

Before starting to discuss the model it is appropriate to give a definition of software quality. The IEEE definition is provided below:

Quality – IEEE 610.12 [24]

1) The degree to which a system, component, or process meets specified requirements.
2) The degree to which a system, component, or process meets customer or user needs or expectations.

The following category models are applicable to software quality and its improvement: quality models, maturity models for process improvements, process models and models of quality management not specific for software.

As it was mentioned in the previous paragraphs, the SEI created maturity models for process improvements as CMM and CMMI and they also created process models as PSP and TSP.

The Capability Maturity Model Integration (CMMI) provides the overall improvement framework needed for effective engineering work. The Personal Software Process (PSP) provides the engineering disciplines that engineers need for consistently using a defined, planned, and measured process. The TSP couples the principles of integrated product teams with the PSP and CMM methods to produce effective teams. In essence, the CMM and PSP provide the context and skills for effective engineering

while the TSP guides engineers in actually doing the work. Thus, the TSP capitalizes on the preparation provided by the PSP and CMM, while also providing explicit guidance on how to do the work. This is clearer in Figure 3.

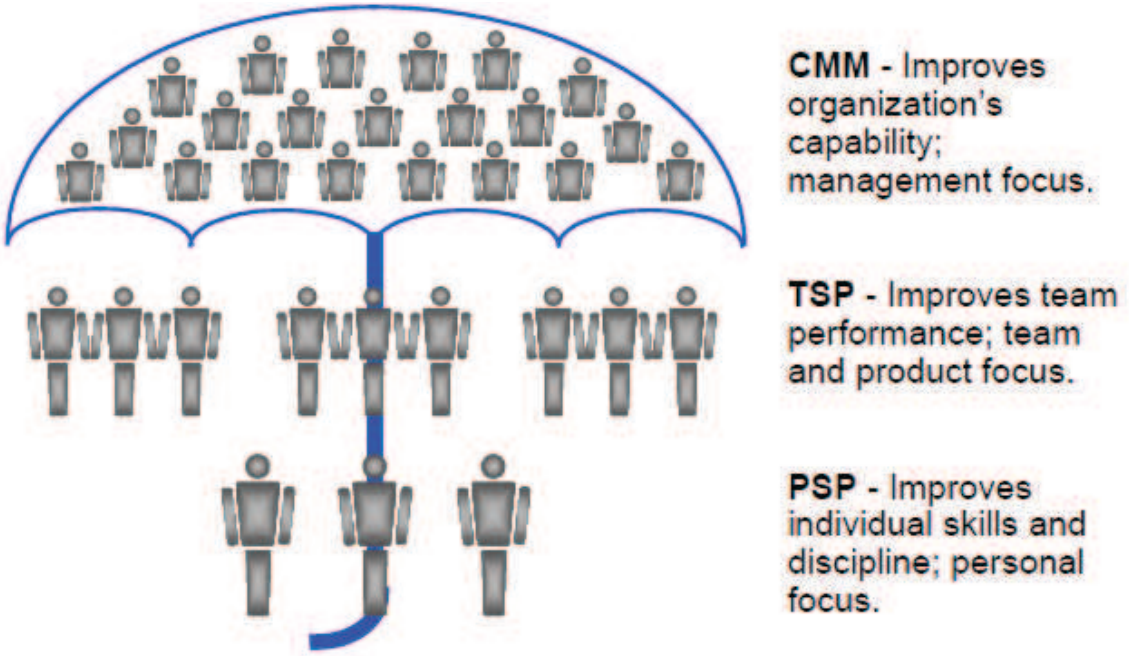


Figure 3: Process Improvement Methods

2. The Personal Software Process

This section presents the main characteristics of the Personal Software Process (PSP). The PSP was created by Watts Humphrey, at the SEI. This description of the PSP is based on the SEI technical report The Personal Software Process [17]. A complete presentation can be found at [3].

The PSP is a defined and measured software process designed to be used by an individual software engineer. Its intended use is to guide the planning and development of software modules or small programs, but it is adaptable to other personal tasks.

Like the SEI Capability Maturity Model for Software, the PSP is based on process improvement principles. While the CMMI is focused on improving organizational capability, the focus of the PSP lies on the individual engineer. To foster improvement at the personal level, PSP extends process management and control to the software engineer. With PSP, engineers develop software using a disciplined, structured approach.

2.1. The Principles of the PSP

These principles are extracted from [17].

"The PSP design is based on the following planning and quality principles:

- Every engineer is different; to be most effective, engineers must plan their work and they must base their plans on their own personal data.

- To consistently improve their performance, engineers must personally use well-defined and measured processes.
- To produce quality products, engineers must feel personally responsible for the quality of their products. Superior products are not produced by mistake; engineers must strive to do quality work.
- It costs less to find and fix defects earlier in a process than later.
- It is more efficient to prevent defects than to find and fix them.
- The right way is always the fastest and cheapest way to do a job."

2.2. The PSP Process Structure

The structure of the PSP process is shown conceptually in Figure 4. The process starts with the requirements and several phases are executed, being planning the first one. PSP provides scripts (instructions) that guide the work. They record their time and defect data on the time and defect logs. During the postmortem phase, they summarize the time and defect data from the logs, measure the program size, and enter these data in the plan summary form.

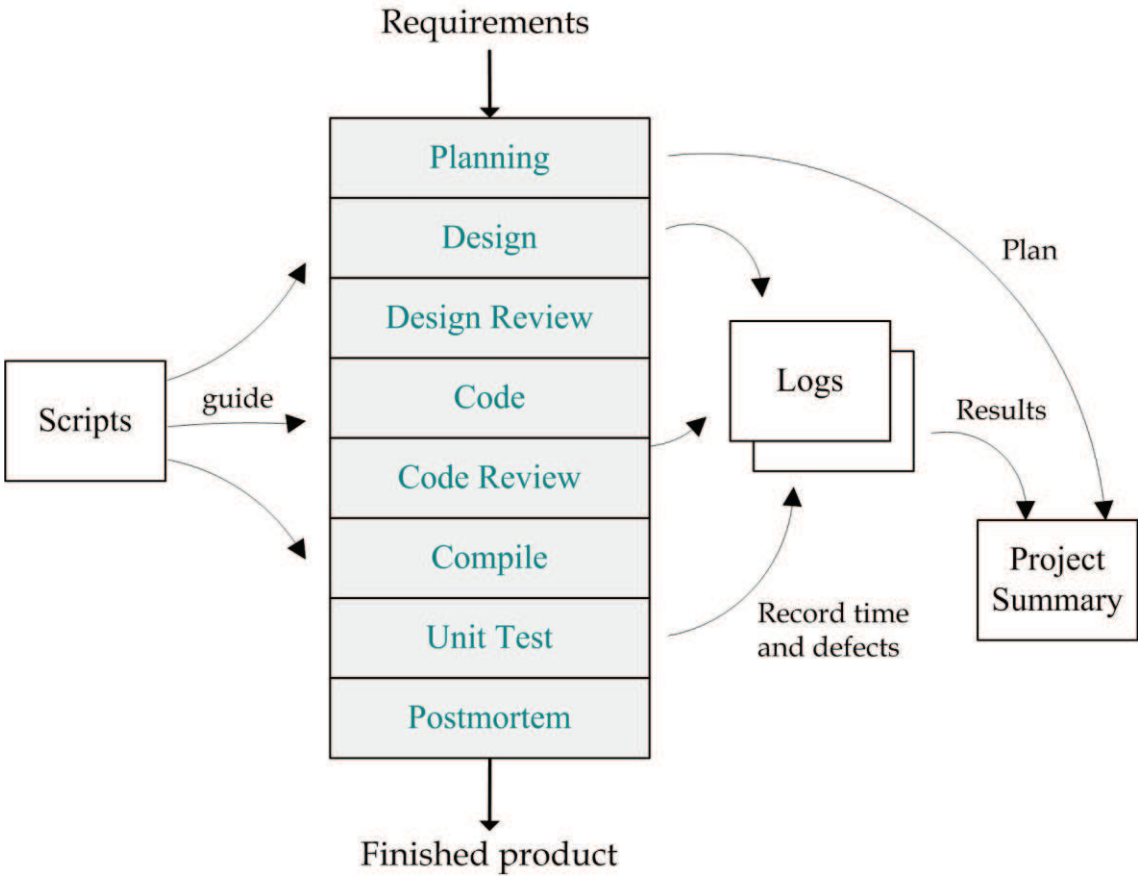


Figure 4: The PSP phases, scripts, logs and project summary

The planning phase has requirements as input. In this phase, the developer completes a conceptual design, the product size estimation, the resources estimation and development schedule. After the product is developed, the real size of the product and the consumed time are stored in the historical database. These are useful to estimate the product size and resources for future tasks. The planning process is shown in Figure 5.

The conceptual design is produced during planning based on the requirements. That is a first approach to problem solution. Then, in the design phase, the engineer examines design alternatives and produces a complete product design. This is useful to make the necessary estimations to create the schedule.

In the PSP the estimation of the size of the software product to be produced and the estimation of the resources needed to complete the software project are done using the PROxy Based Estimating method (PROBE). First are defined the necessary objects to build the product described in the conceptual design. Then, based on the historical data of similar objects and using linear regression, the final size of the product is estimated. With this estimation, the historical data of productivity and linear regression and the time needed for each process phase is calculated. When the engineer finishes this process, he has an estimation of the total product size, total development time and time required for each phase.

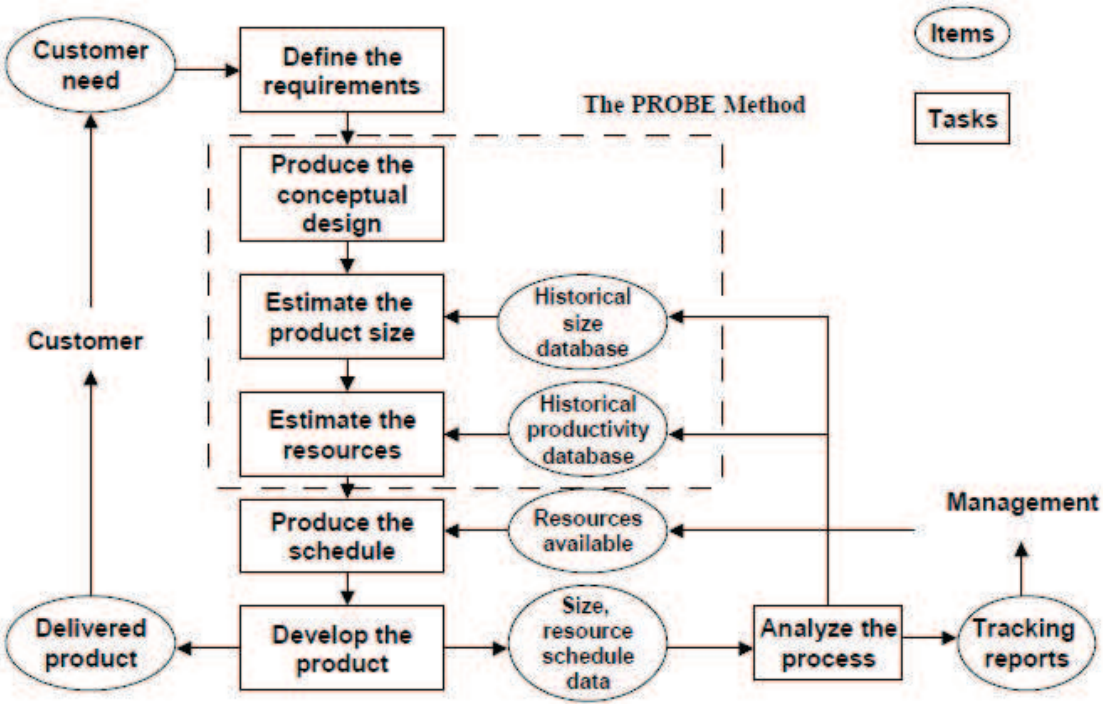


Figure 5: Project Planning Process

The development schedule is created from the time estimations for each phase. It is also necessary to know how much time the developer can devote to the project per day or week.

2.3. PSP Basic Measures

There are three basic measures in the PSP: development time, defects, and size. All other PSP measures are derived from these three basic measures.

2.3.1. Development Time Measurement

Minutes are the unit of measure for development time. Engineers track the number of minutes they spend on each PSP phase, except the time spent on any interruptions such as phone calls, coffee breaks, etc. Some call this “direct time”. In TSP it is converted to hours and called “task-hour”. A form called “Time Recording Log” is used to record development time.

The advantages of this approach to measuring development time are:

- Using minutes is precise and simplifies calculations involving development time.
- Recording interruptions to work reduces the number of time log entries, provides a more accurate measure of the actual time spent, and a more accurate basis for estimating actual development time.
- Tracking interruption time separately can help engineers to deal objectively with issues that affect time management, such as a noisy work environment or inappropriate mix of responsibilities (e.g., software development and help desk support).
- Time log entries take substantially less than a minute to record, but provide a wealth of detailed historical data for planning, tracking, and process improvement.

2.3.2. Defect Measurement

The principal quality focus of the PSP is on defects. A defect is defined as any change that must be made to the design or code in order to get the program to compile or test correctly.

To manage defects, engineers need data on the defects they inject, the phases in which they injected them, the phases in which they found and fixed them, and how long it took to fix them. With the PSP, engineers record data on every defect found in every phase, including reviews, inspections, compiling, and testing. Defects are recorded on the Defect Recording Log as they are found and fixed.

2.3.3. Size Measurement

The primary purpose of size measurement in the PSP is to provide a basis for estimating development time. Lines of code are used for this purpose because they meet the following criteria: they can be automatically counted, precisely defined, and are well correlated with development effort based on the PSP research [3]. Size is also used to normalize other data, such as productivity (LOC per hour) and defect density (defects per KLOC).

PSP uses a LOC accounting scheme:

- **Base LOC** are any LOC from an existing program that will serve as the starting point for the program being developed.
- **Deleted and modified LOC** are those base LOCs that are being deleted or modified in the already existing programs or modules.
- **Added LOC** is the sum of all newly developed object, function, or procedure LOC, plus additions to the base LOC.
- **Reused LOCs** are the LOC taken from the engineer's reuse library and used without modification. If these LOC are modified, then they are considered to be base LOC.
- **Added and Modified LOC** is the sum of added LOC and modified LOC. Added and Modified LOC, not total LOC, is the most commonly used size measure in the PSP.
- **Total LOC** is the total program size.

- **Total new reused LOC** are those added LOC that were written to be reused in the future.

2.4. PSP Quality Process and Product Measures

With size, time, and defect data, there are many ways to measure, evaluate, and manage the quality of a program. The PSP provides a set of quality measures that helps engineers examine the quality of their process and programs from several perspectives. While no single measure can adequately indicate the overall quality of a process or a program, the aggregate picture provided by the full set of PSP measures is a generally reliable quality indicator.

The principal PSP quality measures are:

- Defect density
- Review rate
- Development time ratios
- Defect ratios
- Yield
- Defect removal leverage
- Appraisal to failure ratio (A/FR)

Only the Defect Density and Yield are described in the following paragraphs, as these are the ones that are used in this thesis work. A complete description of the PSP quality measures is presented in Appendix 1.

Defect Density. Defect density refers to the defects per Added and Modified KLOC found in a program. Thus, if a 150 LOC program had 18 defects, the defect density would be 1000*18/150 = 120 defects/KLOC. Defect density is measured for the entire development process and for specific process phases. Since testing only removes a fraction of the defects in a product, when there are more defects that enter a test phase, there will be more remaining after the test phase is completed. Therefore, the number of defects found in a test phase is a good indicator of the number that remains in the product after that test phase is completed.

Yield. In the PSP, yield is measured in two ways. *Phase yield* measures the percentage of the total defects that are found and removed in a phase. For example, if a program entered unit test with 20 defects and unit testing found 9, the unit test phase yield would be 45%. Similarly, if a program entered code review with 50 defects and the review found 28, the code review phase yield would be 56%. *Process yield* refers to the percentage of defects removed before the first compile and unit test. Since the PSP objective is to produce high quality programs, practiced reviewers can find 70% or more of the defects before compiling or testing.

2.5. The PROxy Based Estimating Method

With PROBE, engineers use the relative size of a proxy to make their initial estimate, and then use historical data to convert the relative size of the proxy to LOC. Example proxies for estimating program size are objects, functions, and procedures. For object-oriented languages, the relative size of objects and their methods is used as a proxy. For procedural languages, the relative size of functions or procedures is used as a proxy. Any proxy for size may be used so long as the proxy is correlated with effort, it

can be estimated during planning, and it can be counted in the product. Other examples include screens or screen objects, scripts, reports, and document pages.

The PROBE method requires some preliminary design that requires associating functionality with physical components. This is a design activity requiring detailed understanding of the requirements. It is quite plausible that this activity will reduce the defects injected in later design and code.

Using PROBE, the size estimate is made by first identifying all of the objects that must be developed. Then the type and relative size of the object are determined. The type refers to the general category of component—e.g., computational, input/output, control logic, etc. The five relative size ranges in the PSP are: very small, small, medium, large, and very large. The relative size is then converted to LOC using a size range table based on historical size data for the proxy. The estimated size of the newly developed code is the sum of all new objects, plus any modifications or additions to the existing base code. Predicted program size and effort are estimated using the statistical method linear regression. Linear regression makes use of the historical relationship between prior estimates of size and actual size and effort to generate predicted values for program size and effort. Finally, a prediction interval is calculated that gives the range around the estimate, based on the variance found in the historical data. The prediction interval can be used to assess the quality of the estimate.

The process explained above about how to get the estimation using PROBE is represented in Figure 6.

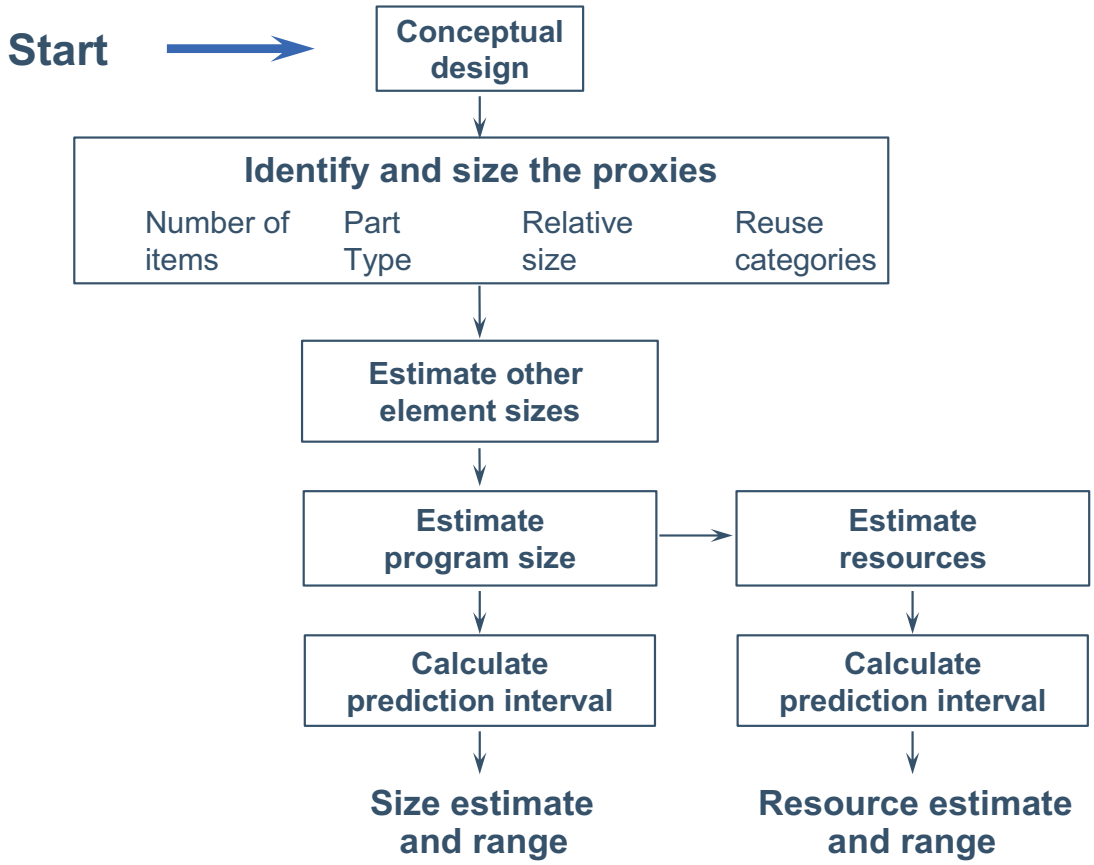


Figure 6: The PROBE estimating method

The PROBE method is not just one technique. In fact, it is a package of different methods:

- PROBE A - regression with estimated proxy size
- PROBE B - regression with plan added and modified size
- PROBE C - the averaging method
- PROBE D - engineering judgment

The method selection procedure depends on the quality of the data:

- The method D must be used when there is no historical data. It should only be used when it is not possible to use methods A, B, or C.
- The method C uses a ratio to adjust size or time based on historical averages. The averaging method is easy to use and requires only one data point.
- The method B applies a regression by using the relationship between plan added and modified size and the actual added and modified size, and also the actual development time. The criteria for using this method are three or more data points that correlate and have reasonable regression parameters.
- The method A applies a regression by using the relationship between estimated proxy size (E) and actual added and modified size, and also the actual development time.

The criteria for using this method are also three or more data points that correlate and reasonable regression parameters. More details about the regression parameters for each PROBE method are explained in [3].

3. The PSP Courses

The PSP is taught through a course. During the course, the engineers build programs while they are progressively learning PSP planning, development, and process assessment practices. For the first exercise, the engineer starts with a simple, defined process (the baseline process, called PSP 0); as the class progresses, new process phases and elements are added, from Estimation and Planning to Code Reviews, to Design, and Design Review. As these elements are added, the process changes. The name of each process and which elements are added in each one are presented in the following subsection. The PSP 2.1 is the complete PSP process. Then, the different PSP courses version that the SEI offers are presented.

3.1. The PSP Process Levels

The six process levels used to introduce the PSP are shown in Figure 7 [3]. Each level builds on the prior level by adding a few process phases to it. This minimizes the impact of process change on the engineer, who only needs to adapt the new techniques into an existing baseline of practices.

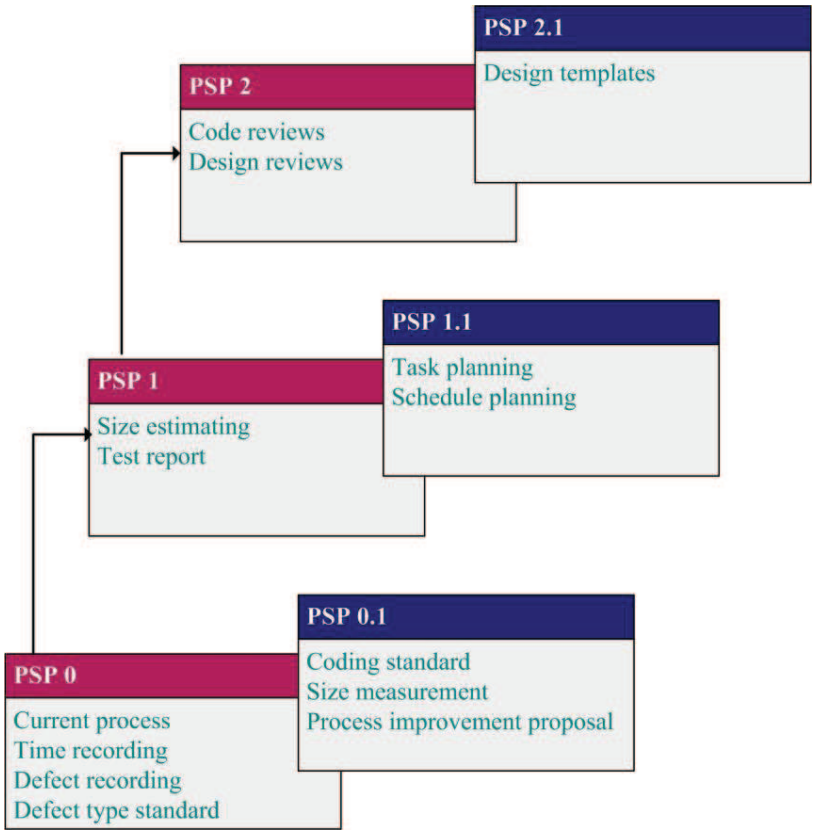


Figure 7: The PSP Process Levels

3.1.1. The Baseline Personal Process - PSP0 and PSP0.1

The baseline personal process (PSP0 and PSP0.1) provides an introduction to the PSP and establishes an initial base of historical size, time, and defect data. Engineers are allowed to use their current methods, but they do so within the framework of the six phases in the baseline process shown in Table 1.

Phase	Description
Plan	Plan the work and document the plan
Design	Design the program
Code	Implement the design
Compile	Compile the program and fix and log all defects found
Test	Test the program and fix and log all defects found
Postmortem	Record actual time, defect, and size data on the plan

Table 1: Phases in the Baseline PSP

PSP0 introduces basic process for measurement and planning. Development time, defects, and program size are measured and recorded on provided forms. A simple plan summary form is used to document planned and actual results. A form for recording process improvement proposals (PIPs) is also introduced (PSP0.1). The PIP form provides engineers with a convenient way to record process problems and proposed solutions.

3.1.2. Personal Project Management - PSP1 and PSP1.1

PSP1 and PSP1.1 focus on personal project management techniques, introducing size and effort estimating, schedule planning, and schedule tracking methods. Size and effort estimates are made using the PROBE method.

PSP uses the earned value method for schedule planning and tracking. The earned value method is a standard management technique that assigns a planned value to each task in a project. A task's planned value is based on the percentage of the total planned project effort that the task will take. As tasks are completed, the task's planned value becomes earned value for the project. The project's earned value then becomes an indicator of the percentage of completed work. When tracked week by week, the project's earned value can be compared to its planned value to determine status, to estimate rate of progress, and to project the completion date for the project.

3.1.3. Personal Quality Management - PSP2 and PSP2.1

PSP2 and PSP2.1 add quality management methods to the PSP: personal design and code reviews, a design notation, design templates, design verification techniques, and measures for managing process and product quality.

The goal of quality management in the PSP is to find and remove all defects before the first compile. The measure associated with this goal is yield. Yield is defined as the percentage of defects injected before compile that were removed before compile. A yield of 100% occurs when all the defects injected before compile are removed before compile.

Two new process phases, design review and code review, are included at PSP2 to help engineers achieve 100% yield. These are personal reviews conducted by an engineer on his/her own design or code. They are structured, data-driven review processes that are guided by personal review checklists derived from the engineer's historical defect data.

Starting with PSP2, engineers also begin using the historical data to plan for quality and control quality during development. Their goal is to remove all the defects they inject before the first compile. During planning, they estimate the number of defects that they will inject and remove in each phase. Then they use the historical correlation between review rates and yield to plan effective and efficient reviews. During development, they control quality by monitoring the actual defects injected and removed versus planned, and by comparing actual review rates to established limits (e.g., less than 200 lines of code reviewed per hour). With sufficient data and practice, engineers are capable of eliminating 60% to 70% of the defects they inject before their first compile.

Reviews are quite effective in eliminating most of the defects found in compile, and many of the defects found in test. But to substantially reduce test defects, better quality designs are needed. PSP2.1 addresses this need by adding a design notation, four design templates, and design verification methods to the PSP. The intent is not to introduce a new design method, but to ensure that the designer examines and documents the design from different perspectives. This improves the design process and makes design verification and review more effective. The design templates in the PSP provide four perspectives on the design: an operational specification, a functional specification, a state specification, and a logic specification.

3.2. Courses structures and assignments

Since the beginnings up to now, the course has changed twice. The first version of the course is called PSP I/II original. Second version is called PSP I/II revised. And finally, the third version is called PSP Fundamentals and Advanced

The first version of the PSP course involved preparing an engineer to apply the PSP in practice. The course followed a staged learning strategy described in the textbook A Discipline for Software Engineering [3]. The text was designed to be used in graduate and senior-level undergraduate courses. Because the textbook was self-contained, experienced engineers could use the textbook to help them learn the PSP on their own, but most engineers needed the structure and support of a formal training course to complete the training.

The first version of the PSP course incorporated what had been called a "self-convincing" learning strategy, which used data from the engineer's own performance to improve learning and motivate use. The course introduced the PSP practices in phases corresponding to six PSP process levels. Each level was built on the capabilities developed and historical data gathered in the previous level. Engineers learnt to use the PSP by writing ten programs, one or two at each of the seven levels, and by preparing five written reports. Engineers could use any design method or programming language in which they were fluent. The programs were typically around one hundred lines of code (LOC) and required a few hours on average to be completed. While writing the programs, engineers gathered process data that were summarized and analyzed during a postmortem phase. With such a short feedback loop, engineers could quickly see the effect of PSP on their own performance. They convinced themselves that the PSP could help them to improve their performance; therefore, they were motivated to begin using the PSP after the course. Table 2 shows the course structure for the PSP I/II Original course. This is extracted from the book "A Discipline for Software Engineering", by Watts Humphrey [3].

Program	PSP Level	Description
1A	PSP0	Calculate the mean and the standard deviation of N real numbers stored in a linked list
2A	PSP0.1	Count the LOC in a program source file
3A	PSP0.1	Enhance program 2A to count object LOC or function/procedure LOC
4A	PSP1	Calculate the linear regression parameters for N pairs of real numbers stored in a linked list
5A	PSP1.1	Numerical integration using Simpson's rule
6A	PSP1.1	Enhance program 4A to calculate a 90% and 70% prediction interval
7A	PSP2	Calculate the correlation of N pairs of real numbers stored in a linked list
8A	PSP2.1	Sort a linked list
9A	PSP2.1	Chi-square test for normality
10A	PSP3	Calculate the multiple linear regression parameters for N sets of four real numbers stored in a linked list

Table 2: PSP I/II Original course structure

An important change in the revised PSP I/II course is the reduction from ten to eight programming assignments. The completion rate for the course had been identified as a problem and the take home assignments had been identified as a root cause. This version of PSP has no programming assignments on the last day of each week. In principle, this enables students to complete the programming assignments in class rather than as take home work. The reduction in assignments was achieved by modifying several assignments and combining the counting exercises. One programming assignment was removed from PSP process levels 0.1 and 1.1. The result is a more

rapid process introduction sequence. The following table summarizes the 8 programming assignments for this version of PSP I/II and identifies relationships with the previous programming assignments. Table 3 shows the PSP I/II Revised course structure.

Program	PSP Level	Description
1	PSP0	mean and standard deviation (same as 1A)
2	PSP0.1	size counting for a program and its constituent parts (same as 3A)
3	PSP1	linear regression parameters and estimation (4A plus first step of 6A)
4	PSP1.1	relative size table (new)
5	PSP2	Simpson's rule integration with t distribution (similar to 5A)
6	PSP2.1	Integrate to find x value for a given area (second step of 6A)
7	PSP2.1	Correlation, significance and prediction interval (rest of 6A and 7A)
8	PSP2.1	Multiple regression (same as 10A)

Table 3: PSP I/II Revised course structure

Table 4 highlights the changes to the programming assignments from the Original 10 assignment PSP I/II through the 8 assignment PSP I/II and the 7 assignment PSP Fundamentals/Advanced. Programming assignment 8, the multiple regressions was eliminated from PSP Advanced. The remaining program requirements were not changed between the 8 assignment course, and Fundamentals/Advanced but the process sequence used to develop the assignments was changed. In the first four exercises, the students are now required to use PSP0 through PSP2 in order to prepare them to conduct reviews and inspections on TSP teams. In the last three exercises, the students are required to use PSP2.1 for all programming assignments.

Prog.	PSP I/II original		PSP I/II revised		PSP Fundamentals	
	PSP Level	Assignment	PSP Level	Assignment	PSP Level	Assignment
1	0	Std dev/mean	0	Std dev/mean	0	Std dev/mean
2	0.1	Size Count Program	0.1	Size counter	1	Size counter
3	0.1	Size count Program and Parts	1	Correlation/ significance	2	Correlation/ significance
4	1	Linear regression	1.1	Relative size table	2	Relative size table
5	1.1	Simpson's rule	2	Simpson's rule	2.1	Simpson's rule
6	1.1	Prediction Interval	2.1	Search algorithm	2.1	Search algorithm
7	2	Correlation/ significance	2.1	Prediction interval	2.1	Prediction interval
8	2.1	Sort List of Pairs	2.1	Multiple regression		
9	2.1	Chi square				
10	2.1	Multiple regression				

Table 4: Programming assignments of all three PSP courses

Figure 8 shows a line chart of the PSP level vs. Program assignment number, for the three PSP courses. In this chart, we can see graphically how each PSP course evolves and how the different PSP levels are introduced among the exercises.

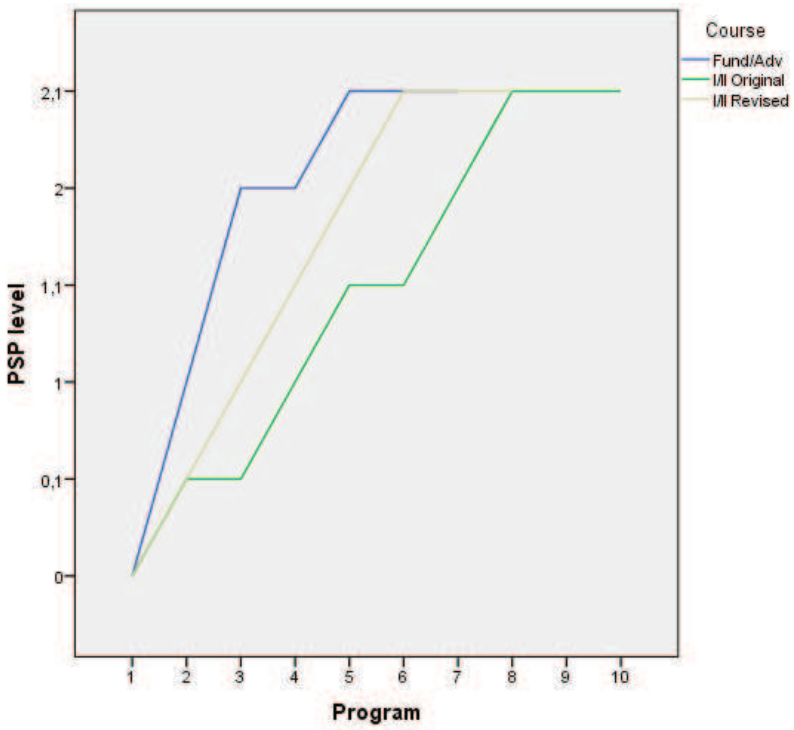


Figure 8: PSP level vs Program number for the three PSP courses

It is interesting to see how differently engineers are introduced to the specific techniques. For example, according to program number 3, a student of the PSP I/II Original course is still in PSP0.1, recollecting data to find out how his base process behaves. On the other hand, a student of the PSP I/II revised course is on PSP1, applying the PROBE method for size and time estimations. And a PSP Fund/Adv student is not only applying the PROBE method for estimations, but also applying design and code reviews. It is important to notice this because in our work we are going to use the PSP levels and the program assignment numbers correspondence in order to analyze the effects of the PSP-learning effects on the engineers' performance, as well as we will try to determine if the introduced techniques are the root cause of the improvements of if they are only a consequence of gaining experience in the problem domain.

In our thesis work, we use the data of the last two versions of the PSP courses: the PSP I/II revised course and the PSP Fundamentals and Advance course.

Chapter 3

Background in Empirical Software Engineering and Statistical Analysis Methods

Carrying out a formal statistical analysis requires an understanding of concepts, techniques and tools normally used in the Empirical Software Engineering (ESE).

The objective of this section is to present the basics of ESE, in order to be able to apply all these concepts in our main work. This chapter is almost completely based on the books “Experimentation in Software Engineering: An Introduction” [25], “Basics of Software Engineering Experimentation” [26], “Software Metrics - A Rigorous and Practical Approach” [27] and “Using Multivariate Statistics” [14].

In the following sections an introduction to the empirical methods and controlled experiments is presented. Also, the main concepts of statistical data analysis are presented, including different statistical data analysis methods that have been used to complete this work. Finally, an explanation of how the concepts are applied in our study is presented.

1. Introduction to the Empirical Methods

The ESE uses methods and experimental techniques as tools for research. The empirical evidence provides support for the evaluation and validation of attributes (e.g. cost, efficiency, quality) in various types of software engineering elements (e.g. products, processes, techniques). It is based on experimentation as a method to match ideas or theories with reality. Such experimentation refers to the speculations shown with facts, assumptions and beliefs about building software.

We can distinguish two different approaches to empirical research: the qualitative and the quantitative approach. The qualitative approach is based on studying the nature of the object and interpreting a phenomenon based on the concept that people have of it. The data obtained from these investigations are mainly composed of text, graphics and images, among others. The quantitative approach implies finding a numerical relationship between two or more groups. It relies on quantifying a relationship or comparing variables or alternatives under study. The data obtained in these studies are always numeric values, allowing comparisons and statistical analysis.

The controlled experiments are one of the strategies for empirical research. The experiments are often performed in a laboratory environment, which allows having a great degree of control. The aim of an experiment is to manipulate one or more

variables and to control the rest. An experiment is a rigorous and controlled formal technique used to carry out an investigation. More strategies for empirical research can be found in Appendix 2.

2. Experimentation Concepts

Before software engineers can experiment, they must be acquainted with experimental design terminology. The most commonly used terms in experimental design are discussed below.

Unit of Analysis: The unit of analysis is the major entity that you are analyzing in your study. For instance, any of the following could be a unit of analysis in a study: individuals, groups, artifacts (books, photos, newspapers), geographical units (town, census tract, state), social interactions (dyadic relations, divorces, arrests). It is called unit of analysis because it is the analysis you do in your study that determines what the unit is. For instance, if you are comparing the children in two classrooms on achievement test scores, the unit is the individual child because you have a score for each child. On the other hand, if you are comparing the two classes on classroom climate, your unit of analysis is the group, in this case the classroom, because you only have a classroom climate score for the class as a whole and not for each individual student. For different analyses in the same study you may have different units of analysis.

Unit of Generalization: "*A factor in deciding the unit of analysis is the level of generalization that the researcher seeks to make*" [28]. Consider a researcher who measures 10 students in 10 classes from 10 different class types, or 1000 students in all. There are three possible levels of generalizations: the student, the classes, and the class types. "*One simple rule is to conduct the analysis at the level at which one wants to make generalizations*" [28]. So if one wants to draw conclusions about students, students should be the unit of analysis. "*However, as will be seen, this simple rule cannot always be followed. The conclusions drawn from an analysis conducted at a group level may not apply at the individual level. Conversely, analyses at the individual level may not apply to the group level. In principal, the analysis should be conducted at the level at which generalizations should be made.*" [28]

Unit of Measurement: Another consideration is the unit of measurement. Again returning to the example of students, classes, and class types, some variables may be measured on students (e.g., achievement), some on the classes (e.g., instructor's gender), and some on the class type (e.g., evaluation method). "*Just because one measures a variable at a certain level does not imply that the variable operates at that level. Consider the variable group size. Presumably this variable operates at the group level. However, if a researcher changed the unit of measurement of the variable and asked persons how big the group was, the variable will still likely operates at the group level, not at the individual level. A related issue is that sometimes a researcher aggregates across units (i.e., averages) and so changes the unit of measurement. For example, to measure organizational climate, the mean of individual measures might be used. Just because the mean is at the level of the organization, does not mean that it, in fact, operates at that level.*" [28]

Experimental unit: The objects on which the experiment is run are called experimental units or experimental objects. For example, patients are experimental units in medical experiments, as it is each piece of land in agricultural experiments. SE

experiments involve subjecting project development or a particular part of the above development process to certain conditions and then collecting specific data set for analysis.

Experimental subjects: The person who applies the methods or techniques to the experimental units is called experimental subject. Unlike other disciplines, the experimental subject has a very important effect on the results of the experiments in SE and, therefore, this variable has to be carefully considered during experiment design.

Response variable: The outcome of an experiment is referred to as a response variable. This outcome must be quantitative. The response variable of an experiment in SE is the project, phase, product or resource characteristic that is measured to test the effects of the provoked variations from one experiment to another. Each response variable value gathered in an experiment is termed observation, and the analysis of all the observations will decide whether or not the hypothesis to be tested can be validated. The response variable is sometimes called dependent variable.

Parameters: Any characteristic (qualitative or quantitative) of the software project that appears invariable throughout the experimentation will be called parameter. These are, therefore, characteristics that do not influence or that we do not want to influence the result of the experiment or, alternatively, the response variable.

Factors: Each software development characteristic to be studied that affects the response variable is called a factor. Each factor has several possible alternatives. Experimentation aims to examine the influence of these alternatives on the value of the response variable. Therefore, the factors of an experiment are any project characteristics that are intentionally varied during experimentation and that affect the result of the experiment. Another term used for the factors is independent variables.

Alternatives or levels: The possible values of the factors during each elementary experiment are called levels. This means that each level of a factor is an alternative for that factor. The term treatment is often used for this concept of alternatives of a factor in experimental design.

Undesired variations or blocking variables: Although the aim to set the characteristics of an experiment that are not intended to be examined at a constant value, it is not always possible to do so. There are inevitable, albeit undesired variations from one experiment to another. These variations can affect several elements of the experiment: the subjects who run the experiment (not enough subjects with similar characteristics can be found to apply the different techniques); the experimental unit (it is not possible to get very similar projects on which to apply the different alternatives); the time when the experiment is run (each alternative has to be applied at different points in time), etc. In short, these variations can affect any conditions of the experiment. These variations are known as blocking variables.

3. Validity Assessment

A fundamental question before moving on to run the experiment is how the results would be validated. There are four categories of threats to validity: conclusion validity, internal validity, construct validity and external validity.

The threats that affect the validity of the conclusions refer to statistical conclusions. Threats that affect the ability to determine whether a relationship exists between the alternative and the result, and if the conclusions reached in this regard are

valid. Examples of these are the choice of statistical methods and the choice of sample size, among others.

The threats that affect the internal validity are those related to observing relationships between the alternative and the results that are product of chance and not the result of applying a factor. This "accident" is caused by unknown elements that influence the results without the knowledge researchers. That is, internal validity is based on ensuring that the alternative in question produces the observed results.

Construct validity indicates how measurement relates to others in accordance with the theory or hypotheses concerning the concepts being measured. An example can be observed when selecting the subjects in an experiment. If the number of approved courses in college was used as a measure of the subject's experience, one would not be using a good measure of experience. In contrast, a good measure would be to use the number of years of industry experience or a combination of both.

External validity is related to the ability to generalize the results. It is affected by the experimental design. The three main risks that have external validity are having the wrong participants as subjects, running the experiment in a wrong way, and making the experiment in a time to affect the results.

4. Introduction to the Controlled Experiments

The process for conducting an experiment consists of several stages: definition, planning, operation, analysis and interpretation, and presentation.

The first phase is the definition, which defines the experiment in terms of the problem, objectives and goals. The next phase is planning, which determines the design of the experiment. In phase operation the experimental design is performed, where data are collected to be analyzed further at the stage of analysis and interpretation. In this last phase, statistical concepts are applied to analyze the data. Finally, the results obtained are presented in the presentation phase.

As in this thesis the focus lies on the analysis and interpretation phase, only the necessary concepts and methods of that phase are going to be presented. More details about the stages of the controlled experiments can be found in Appendix 2.

5. Scales

After data has been collected, we must start the analysis phase. An important aspect to consider in the analysis of the data is the measurement scale. The measurement scale of the data restricts the type of statistical calculations that can be performed. A measure is a mapping of an attribute of an entity as a value, usually a numeric value. Entities are objects that are observed in reality.

The purpose of mapping attributes to a value of measurement is to characterize and manipulate the attributes formally. The measure selected should be valid, therefore, must not violate any property necessary for the attribute that measures, and should be a proper mathematical characterization of the attribute.

Mapping an attribute to a measurement value can take many forms. Each type of mapping of an attribute may be known as scale. The most common types of scales are:

Nominal level: It is the least powerful of the scales. It only maps the attribute of the entity in a name or symbol. The mapping can be viewed as a classification of entities according to the attribute. Examples of nominal scale are graded, labeled, among others.

Ordinal scale: The entities are categorized as a sort. It is mightier than the nominal level. Examples of sorting criteria are "greater than", "best" and "more complex". Examples of ordinal scales are degrees nominal scale, complexity of software, among others.

Interval scale: The interval scale is used when the difference between two measurements is significant, but not the value itself. This type of scale orders the values in the same way that the ordinal scale, but there is a notion of "relative distance" between two entities. This scale is more powerful than the ordinal one. Examples of interval scale are the temperature measured in Celsius or Fahrenheit.

Scale ratio (the ratio of two numbers): If there is a significant zero and the division between two measures is significant, you can use a ratio scale. Ratio scale examples are distance and temperature measured in Kelvin.

After obtaining the necessary data we need to interpret them to draw valid conclusions. The interpretation is done in three stages: characterize the data set using descriptive statistics, reduce the data set and conducting hypothesis tests.

6. Descriptive Statistics

Descriptive statistics are used prior to hypothesis testing, to understand better the nature of the data and to identify abnormal or false data. The main points discussed are: central tendency, dispersion and dependence. Below are the most common measures of each of these aspects. To do this it is assumed that there are x1. . . xn samples.

The measures of central tendency indicate "the middle" of a dataset. Among the most common are: the arithmetic mean, median and mode.

The arithmetic mean is known as the average and it is calculated by adding all samples and dividing the total by the number of samples:

$$\overline{x} = \frac{1}{n}\sum_{i-1}^{n} xi$$

The mean, denoted $\overline{x}$, sums up the characteristics value of a variable taking into account all cases. It is significant for interval and ratio scales.

The median represents the average value of a data set, so that the number of samples that are greater than the median is the same as the number of samples that are less than the median. It is calculated by ordering the samples in ascending or descending order and selecting the observation of the environment. This calculation is well defined if n is odd. If n is even, the median is defined as the arithmetic mean of the two values. This measure is meaningful for ordinal scales, interval and ratio.

The mode is the most common sign. It is calculated by counting the number of samples for each unique value and selecting the value with more quantity. The trend is well defined if there is only one common value. If this is not the case, it is calculated as

the median of the most common samples. The trend is significant for nominal scales, ordinal, interval and ratio.

The arithmetic mean and median are equal if the sample distribution is symmetric. If the distribution is symmetrical and has a single maximum value, the three measures are equal.

The central tendency measures do not provide information on the dispersion of the data set. The greater the dispersion, more variables are the samples, the smaller the dispersion, more homogeneous are the samples.

Dispersion measures the level of deviation of the central tendency, or how scattered or concentrated the data are from the central value. Among the main measures of dispersion are: variance, standard deviation, range and coefficient of variation.

The variance (s2), which is the distribution from its mean, is calculated as the average of the deviations of the samples with respect to the arithmetic mean. Since the sum of the deviations is always zero, we use the squared deviations:

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(xi - \overline{x})^2$$

Divided by $n-1$, not n, because dividing by $n-1$ provides the variance of desirable properties. The variance is significant for interval and ratio scales. The standard deviation, denoted s is defined as the square root of the variance:

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(xi - \overline{x})^2}$$

Often this measure is preferred over variance because it has the same dimensions (unit) that the values of the samples. In contrast, the variance is measured in units squared. The standard deviation is significant for interval and ratio scales.

The range of a data set is the distance between the maximum and minimum:

$$range = x_{max} - x_{min}$$

It is a meaningful measure for interval and ratio scales. When the data set consists of samples related in pairs $(x_i; y_i)$ of two variables, X and Y, it may be interesting to examine the dependence between these variables. The main measures of dependence are: linear regression, covariance and the linear correlation coefficient.

6.1. Reduced Data Set

For hypothesis testing we must use statistical methods. The result of applying these methods depends on the quality of the data. If the data do not represent what is believed, the conclusions drawn from the results of the methods will be incorrect.

Descriptive statistics are strongly influenced by those observations which its values are significantly far from the rest of the values collected. These observations are named outliers.

The outliers influence the measures of dispersion, increasing the variability of what is being measured. In some cases, an analysis on these values that differ significantly from the mean is done and it can be decide to remove them from the data

to be analyzed because they are not representative of the population as they were caused by some kind of problem: measurement error, non desired changes of the characteristics of the subjects, among others. This would be a special or assignable cause. Typically a data point that is more than 3 standard deviations away from the mean, can be considered as suspect.

Once an outlier is identified, its origin should be determined in order to decide what to do with it. If it is consequence of a rare or unusual event that will not happen again, the point can be excluded. If it consequence of a rare event that can happen again, it is not advisable to exclude the value of analysis, because it has relevant information. If it is consequence of a variable that was not considered, it should be considered for base calculations and models also in this variable.

Other methods to reduce the data set are also included in this work. Such methods are based on the Data Quality theory, and are presented in the next chapter.

7. Hypothesis Tests

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 by statisticians to accept or reject statistical hypotheses.

The best way to determine whether a statistical hypothesis is true would be to examine the entire population. Since it is often impractical, researchers typically examine a random sample from the population. If sample data are not consistent with the statistical hypothesis, the hypothesis is rejected.

There are two types of statistical hypotheses:

- **Null hypothesis**. The null hypothesis, denoted by H_0, is usually the hypothesis that sample observations result purely from chance.
- **Alternative hypothesis**. The alternative hypothesis, denoted by H_1, 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. If the null hypothesis is not rejected, nothing can be said about the results. In contrast, if rejected, it can be declared that the hypothesis is false with a given significance (α). This significance level is also called probability of error, because there is a risk of rejecting the hypothesis when in fact is true. This level is controlled by the experimenter.

To test H0, a test unit t and a critic area C must be defined, which is part of the area where t varies. From these definitions, the significance test formula is as follows:

If $t \in C$, reject H0

If t $\notin C$, not reject H0

There are several statistical methods, denoted tests that can be used to evaluate the results of an experiment, more specifically to determine whether to reject the null hypothesis. When carrying out a test it is feasible to calculate the lowest possible value of significance (denoted p-value) which determines when is possible to reject the null hypothesis. Null hypothesis is rejected if the p-value associated with the observed result is less than or equal to the significance level set.

The following are three important chances to test hypotheses:

α = P (commit error type I) = P (reject H0 | H0 is true). It is the probability of rejection H0 when H0 is true.

β = P (commit error type II) = P (not reject H0 | H0 is false). It is the probability of acceptance H0 when H0 is false.

Statistical Power = $1 - \beta$ = P (reject H0 | H0 is false). The statistical power of the test is the probability of rejection H0 when H0 is false.

The experimenter must choose a test with a statistical power as high as possible. High is desirable, but it often comes at a cost to alpha. Power >0.8 is typically chosen as acceptable and it is a common convention.

Several factors affect the power of a test. First, the test itself may be more or less effective. Second, the amount of samples: most samples equals more statistical power. Another aspect is the selection of an alternative hypothesis unilateral or bilateral. A unilateral hypothesis gives a greater power than a bilateral.

The probability of committing a type I error can be controlled and reduced. If the probability is very small, the null hypothesis will only be rejected if we obtain very strong evidence against this hypothesis. The maximum probability of committing a type I error is known as the significance of the test (α).

The most commonly used values for the significance of a test are 0.01, 0.05 and 0.10. The significance is sometimes presented as a percentage, such as 1%, 5% or 10%. This means that the experimenter is willing to allow a probability of 0.01, 0.05, or 0.10 to reject the null hypothesis when it is true, that is, committing a type I error. The significance value is selected before starting to do the experiment in one of several ways.

The α value can be established in the area of research, for example, it can be obtained from articles published in scientific journals. It can also be imposed by the person or company for which one works. Finally, it can be selected taking into account the cost of committing a type I error. The higher the cost, the smaller probability of committing a type I error should be the. The usual value in natural and social sciences is 0.05. In Software Engineering, the value of α is has not yet been established.

There are two types of tests: parametric and nonparametric. Parametric tests are based on a model that involves a specific distribution. In most cases, it is assumed that some of the parameters involved in a parametric test are normally distributed. Parametric tests also require that the parameters can be measured at least in an interval scale. If parameters cannot be measured in at least an interval scale, generally a parametric test cannot be used. In this case there are a wide range of nonparametric tests available. In this work nonparametric tests are not going to be used, therefore they will be left out of the scope of this thesis.

8. Parametric tests

This section presents only the parametric tests that are used in this work.

8.1. Analysis of Variance

ANOVA (ANalysis Of VAriance) is one of the most used parametric tests in Software Engineering experiments for comparing the means of groups of measurement data.

In a one-way ANOVA there is one dependent variable (measurable variable) and one nominal variable (factor). Multiple observations of the measurement variable are made for each value of the nominal variable. The statistical null hypothesis states that the means of the measurement variable are the same for the different categories of data; the alternative hypothesis is that they are not all the same.

The basic idea is to calculate the mean of the observations within each group, then compare the variance among these means to the average variance within each group. Under the null hypothesis that the observations in the different groups all have the same mean, the weighted among-group variance will be the same as the within-group variance. As the means get further apart, the variance among the means increases. The test statistic is thus the ratio of the variance among means divided by the average variance within groups, or Fs. This statistic has a known distribution under the null hypothesis, so the probability of obtaining the observed Fs under the null hypothesis can be calculated.

A two-way ANOVA is used when there is one measurement variable and two nominal variables. The nominal variables are found in all possible combinations.

Repeated measures ANOVA is referred to as within-subjects. In order to analyze data, repeated measures ANOVA for two types of study design can be used. Studies that investigate either:

- changes in mean scores over three or more time points, or
- differences in mean scores under three or more different conditions.

For example, for (1), one might be investigating the effect of a 6-month exercise training program on blood pressure and want to measure blood pressure at 3 separate time points (pre-, midway and post-exercise intervention), which would allow the person to develop a time-course for any exercise effect. For (2), one might get the same subjects to eat different types of cake (chocolate, caramel and lemon) and rate each one for taste, rather than having different people flavor each different cake. The important point with these two study designs is that the same people are being measured more than once on the same dependent variable.

In the broadest terms, all the ANOVA statistical models assume that:

- Subjects are representative of the population of interest and are randomly selected
- Observations on these subjects are independent (from subject to subject)
- Dependent variables are normally distributed in the population
- The variance-covariance matrices of dependent variables are identical

8.2. Analysis of Covariance

The goal of a correlation analysis is to see whether two measurement variables covary, and to measure the strength of any relationship between the variables.

The results of correlation are expressed as a P-value (for the hypothesis test) and an r-value (correlation coefficient) or r2 value (coefficient of determination). The goal of linear regression is to find the equation (slope and intercept) of the line that best fits the points; this line is then used as a visual summary of the relationship between the variables, or for estimating unknown values of one variable when given the value of the other.

ANCOVA (ANalysis of COVAriance) is used when you have two measurement variables and two nominal variables. One of the nominal variables groups is the "hidden" nominal variable that groups the measurement observations into pairs, and the other nominal variable divides the regressions into two or more sets.

The purpose of ANCOVA is to compare two or more linear regression lines. It is a way of comparing the Y variable among groups while statistically controlling for variation in Y caused by variation in the X variable.

Two null hypotheses are tested in an ANCOVA. The first is that the slopes of the regression lines are all the same. If this hypothesis is not rejected, the second null hypothesis is tested: that the Y-intercepts of the regression lines are all the same.

There are five assumptions that underlie the use of ANCOVA:

- The residuals (error terms) should be normally distributed.
- The error variances should be equal for different treatment classes.
- The slopes of the different regression lines should be equal.
- The regression relationship between the dependent variable and concomitant variables must be linear.
- The error terms should be uncorrelated.

9. Application of Concepts

Although we do not went through all the process stages for conducting an experiment, our study can be considered a controlled experiment. Both courses are executed in a controlled environment, with the appropriate tools that allow engineers to collect the process data. In our experiment, we have 169 engineers that executed the PSP Fund/Adv course and 178 engineers that executed the PSP I/II revised course. With the recollected data, we are able to analyze these engineers during the PSP training with respect to the four performance dimensions that we want to study: defect density in unit testing, yield, production rate and size estimation accuracy.

Each dimension arises as a hypothesis in itself. As it is really important to ensure that the statistical analyses are based on quality data, a data cleaning process was defined and executed to obtain a data set with the necessary quality. Therefore, the data set is reduced differently for each dimension that is going to be studied. This data quality process is explained in *Chapter 4*.

To analyze the impact of the different introduced techniques on engineers' performance and the effects of programming repetition, we will be looking for relationships between PSP level, program assignment and course version. These relationships allow us to define the hypotheses tests for each dimension to be studied.

In our study, the primary unit of experiment is the student (engineer). The student is also a unit of analysis. Although we are looking for relationships with PSP

level, program assignment and course version, these are units of generalization. Our data is at the individual level. By aggregating data with means, however, we shift the unit of analysis to PSP Level, program assignment or course version. We will conduct analysis at the individual level, the PSP level, the program assignment and the course version. That is four units of analysis, but we have a single unit of observation, the individual student.

In *Chapter 5* each hypothesis is presented, as well as are defined the related dependent variables, factors, parameters, threats to validity as well as the hypotheses tests to be executed. The parametric tests are applied according to a defined indirect analysis approach that we propose. During the execution of this approach, null hypothesis are being rejected or accepted in order to get to know if for the dependent variable under study the main reason of the changes is the PSP level or the programming repetition. Also when executing this approach, the effects sizes of the changes are visible and discussed. This allows us to contribute to the observations of the last two course formats, which do not have published studies.

Chapter 4

Data Quality in the PSP

The quality of the information is a factor of great significance for any activity based on such information. The quality of the information in every informatics system is fundamental and becoming more and more important every day.

In the Empirical Software Engineering (ESE), conclusions about the experiments can be drawn based on the great amount of the collected data. For our work, it is necessary to analyze and measure the quality of the data recorded by students that participate on the different PSP courses. The data collection is performed by the students using a Microsoft Office Access tool implemented for such purpose: the PSP Student Workbook.

PSP is a defined and measurable software process designed to meet the needs of the software business through the improvement of the practices, techniques and individual skills of the engineers, and by providing a quantitative base to manage the development process. However, if this data has quality problems (data with errors or suspicious of having errors), it would be essential to identify them and put them down for further analysis. It is necessary assess the quality of the data and even clean the data if necessary, so that the conclusions of experiment are based on acceptable quality data rather than on poor quality data.

As part of the data quality analysis, errors are categorized either as real errors related to the data, or as suspect data that may contain errors. The former implies that there is certainty about the existence of an error regarding the data quality, and therefore, correcting it would be the following step. The latter implies that it is not possible to ensure the existence of an error. For both categories, it will be defined which cases would cause the data not to be considered for the statistical analysis in question.

To carry out this study we applied knowledge regarding Data Quality and Data Quality in Software Engineering. The background of the data quality theory, as well as its importance and the basic concepts are introduced in the first section. Then, the data quality impact on ESE is presented. The last section presents the methodology that was applied to identify the data quality problems and to define the data quality metrics. The results the data quality assessment of the available data for both courses are presented, as well as the procedure that has been followed to perform the data cleaning and the data set cut-offs in order to prepare the data for the following statistical analyses.

1. Background on Data Quality

Before any data analysis, it is important to know about the relevance of the data quality. Therefore, it will be necessary to briefly mention what data quality is and the main concepts framed in this field of study. This section is strongly based on [13] [29].

The data represents objects from the real world. Such representations are applicable to contexts of different and varied characteristics. The data can be either stored or put under certain processes or transformations. Use of such data is always essential to guarantee the survival and success of organizations.

The problem of the data quality has been subject of study from different perspectives, and by different areas throughout the years, such is the case of Statistics, Management, or Computer Science. While its importance becomes more evident to the eyes of these and other areas, investigations and improvement intentions increase as well.

It is unquestionable that the storage and/or processing of data is of vital relevance in every person and organizations' lives, within a great range of activities (beyond informatics and informatics systems). There are many examples from our daily life that require storing, processing, transmitting and using data. One of them is when we make the grocery list, because we store data regarding which products to buy, how many of them, and which brands.

Regarding the concept of data quality, it usually happens that one intuitively thinks about certain aspects of the data. Most frequently one tends to think that the data is exact. However, it is necessary to delve deeper into this concept, so as to understand that there are several sides or aspects (the so-called dimensions) that make to the quality of the data. Along the document, some dimensions are explained (accuracy, uniqueness, completeness and consistency). As a trivial example, the act of making a grocery list can be considered:

- If a product or the amount to be bought of a certain product is omitted, it would be an example of completeness problem.
- If there is a mistake in the amount of a product, or if its brand is misspelled, it would be an example of accuracy problem.
- If there are many lists containing the same products, it would be an example of uniqueness problem.

Therefore, it can be said that the definition of data quality is strongly related to accuracy, completeness, consistency and uniqueness of the data (among others). It is because of this that the data quality is called a multifaceted concept, since it depends on the dimensions that define it.

1.1. The Importance of Data Quality

It is in few occasions that there is awareness of the consequences that poor data quality entails. Nevertheless, being able to identify its causes to either eliminate or improve the root of the problem is of vital importance.

In the previous grocery list example, the poor data quality may entail unwanted consequences (such as omitting buying a needed product, or buying the wrong amount of it). In this example, none of these would be too serious. But it is not hard to imagine other situations (lists regarding bulk products importation, duplicated client names, payment mistakes, medical errors), where a non quality data could provoke serious problems.

1.2. Data Quality Dimensions

In the previous section, some example concepts such as accuracy, completeness and consistency were introduced. All these characteristics (and much more) of the data, are called dimensions of the data quality.

Each dimension reflects a different aspect of the data quality. It can refer to the extension of the data (its value), or the intention (its schema). In this way we can distinguish from data quality and schema quality. The focus of this study lies on the quality inherent to the data.

A quality factor is defined as a particular aspect of a dimension. This means that a dimension can be seen as a group of quality factors that share the same purpose.

It is clear that poor data quality can entail various problems, just as the poor schema quality (for example a schema of a relational database without normalizing) can cause bigger problems, such as redundancies. Both types of dimensions, the ones referring to the data and the ones referring to the schema, provide a qualitative view of the quality; while the quantitative measurements are represented by metrics.

A metric is an instrument that defines how to measure a quality factor. The same quality factor can be measured with different metrics. On the other side, we define a measuring method as a process that implements a metric. And at the same time, the same metric can be measured with different methods.

Measurements in a relational database can be performed at various levels of granularity: cell, tuple, table, or even at the level of the entire database. Thus aggregation functions are defined, which move from one level of granularity of data to another, getting a quality summary for that new level. For example, it is possible to obtain a measure of quality of a tuple based in the quality measures of each of their cells.

There are many dimensions that reflect the different aspects of the data quality. This does not come as a surprise due to the fact that the data tries to represent every characteristic about the reality, from spatial and temporal, to social ones. The following contains description of some dimensions of the data quality in which our study will focus on.

1.2.1. Accuracy and Uniqueness

Accuracy can be defined as the closeness between a value v from the real world and its representation v. According to the theoretical approach, accuracy is defined as the correct and precise association between the information system states and the real world objects.

There are three accuracy factors: semantic accuracy, syntactic accuracy and precision accuracy.

The syntactic accuracy refers to the closeness between a value v and the elements of a domain D. This is, if v corresponds to any valid value from D (regardless if such value corresponds with one from the real world). In order to be able to measure the syntactic accuracy, the function comparison can be used. This is the metric that measures the distance between a value and the values of a domain D. Other possible alternatives imply using dictionaries that accurately represent the domain, or checking data against syntactic rules.

The semantic accuracy refers to the closeness between a value v and a real value v. This dimension is mainly measured by the boolean values (indicating if it is a correct value or not). For this, it is necessary to know which real values need to be considered.

In this case, it becomes relevant to measure how well the real world states are represented. One of the used metrics is the comparison of the data with references considered as valid.

The precision, on the other hand, refers to the level of detail of the data.

In order to clarify the concepts, a simple example is presented. Take a database that stores the name and age of certain people. For the information related to the Age of the people, it is specified that its value must be between 0 and 120. It is also known that there is a person called Oscar Javier Morales, who is 23 years old. The following cases are considered:

- If there was a record for a person where the age field had a 234 value, then it there would be a syntactic error (out of the range 0 to 120)
- If there was a record for Oscar where the age field was 19, then it would be a semantic error, because it is known that Oscar is not 19 but 23 (in this case there is no syntactic error because 19 is a valid value for age)
- If there was interest in knowing the age of Oscar in days (or month), there would be a precision problem because that information is given in years only, not months or days.

Despite the fact that semantic accuracy is usually more complex to measure than the syntactic one (because it implies knowing the real world values), when a typing error occurs, both types of accuracies coincide. When a value is altered, the syntactic accuracy will be achieved, because the correct written value will correspond with one from the domain. Semantic accuracy will also be achieved, because there will be a real value associated with the correct written value.

One way of checking the semantic accuracy is comparing different data sources, and from that, finding the desired correct value. This also depends on the resolution of the identification of objects problem, which consists on identifying if two tuples represent the same object from the real world.

Considering accuracy among a group of values, it would be necessary to consider duplication as well. Such problematic occurs when an object from the real world appears more than one time (more than one tuple represents exactly the same object).

However, tuples that represent the same object from the real world but with a different key could also exist. This aspect is considered by the Uniqueness dimension. It is important to highlight that different situations may lead to data duplication:

- When the same entity is identified in different ways
- When errors occur in the primary key of an entity
- When the same entity is repeated with different keys

There are two factors of the Uniqueness dimension to point out:

- Duplication: the same entity appears equally repeated
- Contradiction: the same entity appears repeated with contradictions

1.2.2. Completeness

Completeness can be defined as the measure to which the data is of sufficient scope and depth. According to the theoretical approach, this dimension is defined as the capacity of the informatics system to represent every significant state of a given reality.

There are two factors related to completeness: coverage and density.

Coverage refers to the part of the reality data that are contained in the informatics system. Just as for semantic accuracy, coverage involves a comparison of the informatics system with the real world. Once again, a reference is required. As it is usually hard to obtain, the alternative lies on estimating the size of such reference.

Density refers to the amount of information contained, and the lack of information about the informatics system entities.

Given a relational model, completeness can be characterized by null values, which could mean different things. A null value might indicate that such value does not exist in the real world, or that it does exist but it is unknown, or that there is no certainty about its existence in the real world. It is important to know the cause of its appearance.

For instance, if it was required to record all data about the Earth inhabitants (name, age, gender) within a database, each non-registered person would reduce the completeness of the data (this would be completeness at the relational level). Moreover, completeness would also be reduced if the age or gender of certain people were unknown.

1.2.3. Consistency

This dimension refers to the compliance with the semantic rules defined over the data. According to the theoretical approach, the inconsistency of data appears when there is more than one informatics system state associated with the same object from reality. Incorporating external data as well as data with different formats could lead to inconsistency.

A simple example: if there are data about certain people stored in a table - such as information about date of birth and age-, and there is a record that shows 2005.01.01 as the date of birth, and 42 as the age, there is an inconsistency (it would be a violation of the intra-relational rule, as it is explained below).

- Integrity constraints define properties that must be met by all the stages of the relational schema. There are three types of integrity constraints:
- Domain constraints: it refers to the compliance with the rules about the content of the relation attributes.
- Intra-relational constraints: it refers to the compliance with the rules about one or more than one relation attributes.
- Inter-relational constraints: it refers to the compliance with the rules about different relations attributes.

1.3. Data Cleaning

The activities related to data quality refer to any process (or transformation) applied to the data, with the purpose of improving its quality. In order to carry out such activities, different techniques are used.

Data cleaning is fundamental to achieve the improvement of data quality. It is because of this that it is very useful to address this topic, in order to know and understand the problems that must be faced.

Data cleaning tries to solve the problem related to detection and correction of errors and inconsistencies within the data, with the purpose of improving its quality. These activities are more important in databases in which information has been entered in a way that creates room for errors. For instance, when the information is entered by people using a keyboard, or when it is obtained from non-reliable sources, or when different information sources are integrated. The latter also implies consolidating the data that has the same meaning (but different representation), as well as discarding duplicated data. Data warehouses and informatics systems based on the web are an example of that.

There are different tools that provide support to data cleaning. Nevertheless, it is important to bear in mind that besides the use of certain tools, this task also implies tough manual work or a low level programming work for its resolution.

Error Detection, Correction and Prevention

Finding (or detecting) and correcting errors is done for data that it is rarely created or updated. However, when dealing with data that is frequently updated and created, preventing errors through processes management is widely used. Control stages in the processes of data creation and/or updating take place in order to avoid inconsistencies.

To locate errors, the Data editing technique is used, which consists on defining the rules (edits) that must be respected by certain group of data. In this way it is possible to detect inconsistencies. The edits represent error conditions, for which they must be consistent, and not redundant.

The edits can also be used for error prevention and improvement of processes, avoiding inconsistencies on the database. We will not provide details about the error prevention techniques as we do not use them on our work.

For anomalies' detection and correction, which is when the value of a single datum or more widely differs from the rest of the data, the situation can be any of the ones that follow:

- The value was incorrectly measured, or incorrectly entered on the database
- The value corresponds to a sample that is different from the rest
- The value is incorrect and it simply corresponds to some unusual event from reality

These data can be identified based on two different measurements: by measuring the distance from the registered values to the expected values (internal deviation), or by measuring the variation of the data throughout time in comparison with other data (relative deviation). There are different techniques for that. One of them calculates the average value and the standard deviation of a certain group of data, to identify those

values that deviate too much from the average value. A limit value could be defined so that if the data were beyond the boundary, the data is suspected of being incorrectly recorded. Other techniques also use the time factor to identify outlier data. Some are based on the fact that certain measured or registered data on a specific lapse of time can be strongly related to each other. Others take into account possible cycles where peak values appear, for instance the use of cell phones during Christmas or New Year's Eve.

Dealing with these anomalies implies double effort: first they must be identified, and then it must be decided if they correspond either to correct data of unusual events from reality, or incorrect data, which should be corrected. These are the cases that we categorize as suspect of containing errors, given that we cannot ensure that there is indeed an error in the data.

2. Impact of Data Quality in Empirical Software Engineering

Regarding Software Engineering, the need to improve the quality of the obtained products as well as the Software Development process itself, has become a critical and fundamental aspect. This tendency can also be seen in the Quality Data field, given the growing amount of information generated and stored, increasing as well its value and importance for organizations.

The analysis presented here is no exception in that sense. It is highly important to obtain good quality data in every experiment. This is due to the fact that such data is the starting point for further statistical analysis, comparative studies and data analysis, in which the results of the experiences are based on. It would be worthless to draw conclusions based on poor quality data. Furthermore, it would be detrimental because a false reality would be represented.

Unfortunately, in an extensive and very recent literature review, Bachmann realized there are very few studies that examine the quality of the data collected during the use of a software development process [30]. Most of the few works are studies about the defect records' data and about the data of the software for version control, ignoring other data generated during the use of a software development process.

In another study, the authors found that poor quality data collected during the development process affects the quality of the software product developed [31]. It is due to the severe negative impact that poor data quality can have, that Shepperd says *"... I therefore suggest that this topic [data quality] should become a higher priority amongst empirical software engineering researchers*" [32].

This situation encourages us to move forward and analyze the quality of the PSP data before running the statistical analysis. Particularly we focus on knowing and improving the quality of the data recollected by the students during the execution of a discipline software development process[3] as it is the PSP. The way ahead implies analyzing and measuring the data quality generated by different PSP courses.

In this way, it will be possible to identify certain errors within the data that will cause them to be dismissed for further analysis. Moreover, it will allow the obtained results to produce a better and more faithful representation of the reality.

[3] It should be clear that the data quality of a process' execution, the quality of the process and the quality of a product are very different concepts. We are assessing the quality of the data generated by the execution of the PSP during the PSP courses.

Undoubtedly, making analysis over data that contains errors might cause taking wrong corrective actions. In other words, we would be might take actions to improve aspects that should not be the focus, or we might take actions that do not have the intended effect. This might cause not only a loss in important resources (time, effort, dedication) over actions that will not help to improve the critical aspects -because we are focusing on incorrect improvement actions- but also may damage those aspects that did not needed to be acted upon.

Therefore, the main issue to consider is that the sample of data from which the statistics analysis are going to be made, must be of high quality. In order to achieve this, the first step would be to measure the quality over the relevant data for further studies, with the purpose of detecting data containing errors, and data suspicious of containing errors. Two possible ways appear then. The first consists on identifying corrective actions, or cleaning actions over the detected errors. Another alternative might be not considering such data on the sample. Despite the data cleaning allows having more available data, and clean, many times the cost of this task is too high, or even impossible, given the context and the data origin. Therefore, not considering data with errors ends up being in many cases not only a cheaper alternative, but also an executable one. And besides, it will lead to obtaining results that would contribute on the improvement of the process under study.

3. Data Quality Analysis in the PSP

In order to carry out any data quality study, the first step must be to know the reality and the context under analysis. It is because of this that the study of the Personal Software Process (PSP) is done as deeply as possible, including the use of the tool for data recording. This would allow us to identify the potential quality problems the data under study could have.

After being aware of the domain to be analyzed, an evaluation of the possible dimensions and quality factors interesting to be measured and considered takes place as the first step. It is important to highlight here that the focus lies on how the data is recorded by the tool, given that it is the only way for entering data in the database. This means that all controls made automatically by the tool will not be worth measuring, because we can be sure that there will not be errors on such data (as an example: constraints of non null values, foreign keys, automatic calculations already existing in the database). In this way, the focus lies on the values that are entered manually, because it is here where most errors will occur. Unfortunately no documentation (user's manual, technical and/or functional documentation, database diagram) about the tool was available. The automatic controls identified on the tool -and that are considered in this study- were a consequence of the use of it, meaning they were detected along with the use of such tool. As part of the first analysis, the Grading Checklists are considered input of great contribution. They are used by the instructors for correcting exercises done during the course. Based on these checklists, it is possible to identify possible data quality problems to bear in mind, from which measurements can be defined. Moreover, possible completeness and accuracy problems can be identified. The detailed description of these metrics can be found in Appendix 3.

It is worth mentioning that for many cases, suspect values can be found. This means that they appear to be wrong but it is impossible to ensure if they do represent to a data quality problem, or if they represent an unusual but real event, and therefore

should not be considered as data errors. For such cases, given that access to the sources is not possible (participants of the course) to know the real origin of the error, the proposed alternatives are the following. For those cases in which analyzing manually is feasible (according to the amount of data), they will be considered as a separate sample with the purpose of identifying if they contain -or not- data quality problems. For the rest of the cases, data will be dismissed and will not be considered for the statistical analysis due to its suspiciousness.

Regarding the scope of the data that needs to be measured (the whole database, or only certain data or tables), focus will be on those data in which this thesis is based. Nevertheless, possible relations with other data that might affect the global results will not be left aside.

3.1. Data Quality Problems

A classification and detailed description of the main quality problems found are obtained after analyzing all possible data errors identified in the first stage. For every identified quality problem, the following aspects are included:

- A brief description of what it consists of and why its consideration is to be of importance.
- The probable causes or known causes (if they are already known)
- The metrics used to measure it, including its granularity (which it may be at cell level, tuple, table or entire database). For all cases the result unit of the measure is Boolean, meaning that whether the measured object contains an error or not is indicated.

The quality data dimensions to be measured are: accuracy, completeness, consistency and uniqueness. On Table 5, the quality problems are presented for each dimension and factor.

Dimension	Factor	Quality Problem
Accuracy	Syntactic Accuracy	Out of range value
	Semantic Accuracy	Incorrect project identifier
	Precision	Precision in times
Completeness	Density	Null value
	Coverage	Non-existing record
Consistency	Domain Integrity	Domain integrity rules
	Intra-relationship integrity	Intra-relationship integrity rules
	Referential integrity	Referential integrity rules
		Invalid reference
Uniqueness	Duplication	Duplicated register

Table 5: Data Quality Problems

In the following subsections all the quality problems are presented briefly. A detailed presentation about the data quality problems addressed in our work can be found in Appendix 4. For all cases, the unit of measure of the result is boolean: what is measured is whether the object has a problem or not. In most cases -except if stated not to- problems are measured by the definition and execution of SQL queries.

3.1.1. Out of Range Value

The out of range values are those which are situated outside a previously defined valid range. Such values could correspond to anomalous values and as a consequence, results and conclusions obtained through the data analysis might not accurately reflect reality.

For each of the identified cases (for instance, times), it is necessary to define certain criteria to appropriately determine the range in which values are going to be evaluated. Outliers are going to be identified by SQL queries. An outlier is a value that is unusually higher or lower than the others within a group of data, but that does not necessarily correspond to a wrong value. The range is determined considering the mean value and the standard deviation of the registered values. Values outside such range will be considered as probable of containing errors, and therefore they will be analyzed isolated from the rest.

3.1.2. Incorrect Project Identifier

All users should use the same indicators to refer to the same projects of reality. If not, it is impossible to do data analysis by project. With this problem all projects that have the right PSP process associated are indentified, but their project identifier does not correspond to reality[4].

3.1.3. Precision in Times

With this problem the objective is to measure the precision level of the registered times, given that it is of interest knowing the exact moment in which a defect was registered. Hours, minutes and seconds of the registered times should never be 0.

3.1.4. Null Value

It is of interest knowing which information was registered and which was omitted. For the omitted information, it is important to know the cause of such omission, and if possible, to determine the value it should take instead of the null one.

Fields that admit null values are identified, but in reality they should have a value different from empty (the fact that they admit null values is due to an incorrect design of the database that the PSP course tool uses).

3.1.5. Non-existing Records

Those registers that do not exist in the database but that do exist in reality are identified, meaning its entry was omitted. In other words, there is a portion of data that exists in reality, but is not reflected in the database. If we do not count with the whole universe of data, the statistic analyses that are carried out would only reflect a part of the reality under study.

3.1.6. Domain Integrity Rules

For some attributes it is possible to define the domain to which its values must belong to. In this case, it is defined that the valid domain for certain values must always be greater than zero.

[4] In the PSP student database, each program assignment has a project identifier.

3.1.7. Intra-relationship Integrity Rules

A set of rules are defined under certain attributes of a same table, which must be met within the database under study. If any of these rules were to be violated, data consistency would be affected, and therefore any analysis done over such data would be affected as well.

3.1.8. Referential Integrity Rules and Invalid Reference

A set of rules are defined under certain attributes of a same table, which must be met within the database under study. Particularly, certain references towards certain inexistent tuples in the database are identified; therefore, they end up being invalid references. This is a consequence of an error in the schema design of the database, because the definition of the foreign keys over certain attributes is omitted.

3.1.9. Duplicated Register

This quality problem is identified when there are two or more registers that appear repeated in the exact same way. There are two situations:

- When they have the same value for the key and the rest of the attributes (or null values). This case is dealt with RDBMS[5] controls.
- Despite having different primary key, they refer to the same object from reality and contain the same data regarding the defined fields. For this case, it is necessary to verify that there are not repeated registers (according to defined criteria) in the database under study. Two cases are considered: defects duplication, which corresponds to defects that were registered at the exact same hour, and students duplication, which corresponds to students that have the same name, same instructor and same date of creation of their profile within the tool.

3.2. Data Quality Analysis Results

Ten quality problems were identified, and a total of 91 metrics were defined to measure these problems applied to objects (cells and/or tuples) of the database. We did the measuring of the totality of the metrics that were defined. The execution of the measuring for these metrics was done automatically through SQL sentences in all cases. For the 20% of the cases, PHP programmed algorithms were used. This section presents the most important results in a general way. The complete results of our analysis can be found in Appendix 3.

Table 6 shows the amount of defined metrics for each quality problem, the amount of metrics that correspond to errors, and the amount that correspond to suspicious cases. Besides, it also indicates for each quality problem the percentage of objects with such quality problem according to the metrics that measure errors and according to the metrics that measure suspicious cases.

After executing all measuring we observe that 1.34% of the total of the measured objects has some error or possible error. If we only consider the metrics that measure errors (and we do not count suspect cases) such value decreases to 0.99%, and if we only consider the metrics that measure suspect cases, the percentage of objects with errors increases to 2.10%.

[5] The term RDBMS refers to Relational Database Management System

A prior study on the quality of the PSP data presented a 4.8% of errors within data [33]. In that case, many of the errors in the data were due to manual calculations of derived data, something that no longer occurs with the current PSP tools. This difference in data quality given by the used tools is also mentioned by Bachmann and Bernstein *"We also discussed the impact of varying data characteristics across [the studied] projects and concluded that, e.g., the nature of used software engineering processes, the use of process support tools [...] result in differing data characteristics"* [31].

Quality Problem	# total metrics	# metrics for errors	# metrics for suspicious	% objects with error	% suspicious objects
Out of range value	8	0	8	0,00 %	3,20 %
Incorrect project identifier	1	1	0	8,66 %	0,00 %
Precision in times	1	1	0	7,23 %	0,00 %
Null value	26	26	0	1,32 %	0,00 %
Non-existing record	3	1	2	0,12 %	16,90 %
Domain integrity rules	15	15	0	1,29 %	0,00 %
Intra-relationship integrity rules	5	2	3	1,05 %	0,40 %
Referential integrity rules	10	9	1	11,94 %	3,44 %
Invalid reference	20	20	0	0,04 %	0,00%
Duplicated register	2	2	0	1,62 %	0,00 %

Table 6: Amount of metrics and percentage of objects with error for each quality problem

These quality results, both seen from a global and particular point of view for each quality problem, are indicating that data cleaning is needed, in order to guarantee that the statistic analyses of data from these courses will be done over valid data.

3.3. Data cleaning and cut-offs applied

In order to consider only the data with no known quality problems in our analyses, we performed the appropriate data cleaning procedures as well as discarded the poor data quality that was not possible to be cleaned. Those procedures are based on 91 defined metrics for the 10 data quality problems addressed. These are presented in detail in Appendix 3.

Besides the already explained metrics' classification, we classified the metrics in three groups, according to what aspect of the PSP basic concepts they measure: size, effort or defect. We also associated the four engineers' performance dimension that we are going to analyze with these three concepts as follows:

- Defect density in unit testing is associated with "defect" and "size". This is because this variable depends on the removed defects records and the actual size records.

- Yield is associated with “defect”. This is because this variable depends on the removed and injected defects records.
- Production rate is associated with “size” and “effort”. This is because this variable depends on the actual size and actual effort records.
- Size estimation accuracy is associated with “size”. This is because this variable only depends on the actual size and estimated size records.

According to these associations, we applied the data cleaning and data cut-off for each dependent variable we studied based on the objects with quality errors that were detected by the metrics associated to the same concepts. In the following chapter, the final data set (the one after the data cleaning process) considered for each hypothesis is presented.

The data quality analysis of PSP’s data done in this thesis was published in the Proceedings of the IX Jornadas Iberoamericanas de Ingeniería de Software e Ingeniería del Conocimiento (Iberoamerican Conference in Software Engineering and Knowledge Engineering), 2012 [34].

Chapter 5

Data Analysis

As we know, the strategy of PSP is to improve the work discipline and thus to improve the performance of practicing software developers. The underlying assumption is that such a defined and well-structured process leads to better estimating, better planning and tracking, protection against over commitment, a better personal commitment to quality, and the engineer's involvement in continuous process improvement. Based on these goals, Hayes et al. examined five dimensions of the PSP; size estimation, effort estimation, product quality, process quality, and personal quality [7]. For each of these dimensions he formulated an individual hypothesis. Later Rombach et al. added another dimension and hypothesis, defect estimation [8]. Closer in time, Nichols et all. in a technical report re-analyzed and reflected on each hypothesis with a more complete data set [10].

Our general goal is to know if the different techniques and phases of the PSP (and therefore, the PSP itself) produce positive changes in different software development aspects. More specifically, we intend to conduct an empirical investigation of the data recollected during the courses to analyze and evaluate the effectiveness of the PSP and the impact of the domain experience.

In our work we took some of the hypothesis that Hayes et al. defined, and extended them to analyze whether the introduction of a specific technique improves a dimension or if such improvement is only a consequence of gaining experience in the problem domain.

We analyzed and compared the data from the latest two courses versions of PSP for the purpose of evaluating engineers' performance improvements with respect to defect density in unit testing / yield / production rate / size estimation accuracy from the viewpoint of a researcher in the context of the PSP training of engineers in "PSP for Engineers I/II revised" course and the training of engineers in "PSP Fundamentals and Advance" course.

This chapter is structured as follows. Section 1 defines the four hypotheses to be studied. Section 2 describes the origin of the data, its preparation for further analysis, the statistical model and an indirect statistical method proposed to analyze the data. Section 3 presents general internal and external threats to validity of the data analysis that apply to all the hypotheses. Sections 4 to 7 present the detailed results of the data analysis for each hypothesis. In the General Conclusions of the Data Analysis, the general conclusions are presented.

1. Hypotheses Definition

This section presents each one of the hypothesis defined and studied to evaluate changes in engineers' performance.

1.1 Hypothesis 1 - Defect Density in Unit Testing

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. The consequence of high defect density in software engineering is typically seen in the form of defect fixing or rework effort incurred on projects.

Here we address the impact of PSP application on the defect density of the programs produced by the engineers. In addition to overall defect density, a specific focus on the defect density of programs during the compile and test phases of the life cycle is provided. Defects that remain in the product at the end of the life cycle are the most costly to remove and have frequently been used to estimate the defect density of the delivered product; therefore, a reduction in these 'late' defects has a beneficial effect above and beyond the impact of a reduction in overall defect density.

PSP2.1 introduces design notation, four design templates, and design verification methods to the PSP. These ensure that the designer examines and documents the design from different perspectives. This improves the design process, which makes the engineer consider many perspectives. The design templates in the PSP provide four perspectives on the design: an operational specification, a functional specification, a state specification, and a logic specification. Design is considered a defect prevention activity, so we think that total defect density should decrease significantly.

The hypothesis to be investigated is as follows:

> *As engineers progress through PSP training, the number of defects injected and therefore removed per thousand lines of code (KLOC) decreases. With the introduction of design and code reviews in PSP level 2, the defect densities of programs entering the compile and test phases decrease significantly.*

1.2 Hypothesis 2 - Yield

One of the most powerful process metrics used in the PSP is the pre-compile defect yield (hereafter referred to simply as yield). Yield is the percentage of defects injected before the compile phase that are removed before the first compile. The PSP teaches engineers to examine process quality by quantifying the yield of their personal software process. By understanding how well their process works to prevent defects from "entering" the last phases of the process, engineers can see for themselves the benefit of changes they make to their processes. In general, the goal is to work for a yield of 100%.

The hypothesis to be addressed is as follows:

> *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.*

1.3 Hypothesis 3 – Production Rate

Production rate is a major focus of most organizations that produce goods for customers. The quantification of product output per unit of time spent is as old a metric as can be found in any industry. In PSP, the data collected by the engineers allow them to compute lines of code per hour (LOC/Hr) as a measure of their personal production rate.

We believe that even when PSP increments the amount of design documentation and data tracking, production rate remains unchanged during the PSP course.

The hypothesis to be tested is:

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 spent does not change with a higher PSP level.

1.4 Hypothesis 4 - Size Estimation Accuracy

Estimating the size of a job prior to deciding how long it will take to complete, while logical to most people, seems to be a difficult practice to instill in software engineers. The PSP provides a proxy-based estimation method (introduced during PSP level 1) to help engineers decompose the program and estimate the size of each element, based on historical data. Introduction of this method is designed to enable engineers to become more accurate estimators of their own work.

While there will always be a subjective element to estimation no matter how much data it is based on, the PSP training strives to teach engineers how to make the best use of their own past experience. When the size estimation method is introduced at the start of PSP level 1, the engineers have data from previous assignments as a basis for estimating.

Therefore, the hypothesis tested is as follows:

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, with 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.

2. Data Set and Statistical Model

The data for this analysis was reported and collected by the Software Engineering Institute. 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 we used data from the seven program course version of 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 began with 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 run data cleaning algorithms to include only the students who had completed all programming exercises, in order to clean and remove errors and questionable data.

To determine the cuts on the data set, we first developed an integrated data storage model. We designed such model in order to support the analysis and the assessment of data quality, based on the data quality theory, as it is explained in *Chapter 4*. In this way, we thoroughly identified and defined possible quality problems that the data under study might contain, we implemented the algorithms required for cleaning and collecting the metadata and finally, we executed those algorithms. Major data quality problems were related to the consistency, accuracy, completeness and uniqueness dimensions. This meant that after that data quality process, our data set was reduced as can be seen in Table 7.

Number of Engineers				
Course	**Defect Density in UT**	**Yield**	**Production rate**	**Size Estimation Accuracy**
PSP I/II	48	97	78	163
PSP Fund/Adv	45	120	82	148
Total	**93**	**217**	**160**	**311**

Table 7: Number of engineers who provided complete sets of data for each hypothesis

Differences in performance between engineers are typically the greatest source of variability in software engineering research, and this study is no exception. However, the design of the PSP class, and the standardization of each engineer's measurement practice, allow the use of statistical models which are well suited for dealing with the variation among engineers.

In the summarized analyses presented, we studied the changes in engineers' data over seven programming assignments. Rather than analyzing changes in group averages, this study focuses on the average changes of individual engineers. Some engineers performed better than others from the first assignment, and some improved faster than others during the course. In order to discover the pattern of improvement in the presence of these natural differences between engineers, the statistical method known as the repeated measures analysis of variance (ANOVA) is used [14]. In brief, the repeated measures analysis of variance takes advantage of situations where the same people are measured over a succession of trials. By treating previous trials as baselines, the differences in measures across trials (rather than the measures themselves) are analyzed to uncover trends across the data. This allows differences among baselines to be factored out of the analysis. In addition, the different rates of improvement between people can be viewed more clearly. If the majority of people change substantially (relative to their own baselines), the statistical test will reveal this pattern. If only a few people improved in performance, the statistical test would not be likely to suggest a statistically significant difference, no matter how large the improvement of these few people was.

Below we define some terms and independent variables that must be clear to understand 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. This variable in plots can be seen as 1 and 2 respectively.
- Program Assignment or Program Number – Refers to an exercise that a student has performed during the PSP course. Values go from 1 to 7. Program assignment 8 of the PSP I/II course version is not going to 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, 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 and we group PSP1.0 and PSP1.1, and analyze PSP2.0 and PSP2.1 separately. So the PSP Level variable can be seen in plots as 0, 1, 2 or 3 respectively.

To apply ANOVA, the dependent variable needs to be continuous, and an independent variable is necessary to represent the time points or conditions (categorical). In this study we are considering the PSP Course, the PSP assignment and the PSP level to be the independent variables, and the analyzed hypotheses to be the dependent variables. Table 8 describes the dependent variable in detail.

Dependent Variable	Value
Defect Density in Unit Testing (H1)	1000 * Total defects removed in testing / Actual added and modified LOC
Yield (H2)	100 * Defects removed before the compile phase / Defects injected before the compile phase
Production Rate (H3)	(Actual A&M LOC / Actual Minutes) * 60
Size Estimation Accuracy (H4)	(Estimated LOC – Actual LOC) / Estimated LOC

Table 8: Dependent variables for each hypothesis and their values

2.1 An Indirect Statistical Method of Analysis

As we said in the introduction, the global objective of this study is to use the PSP data from the latest two course formats to demonstrate whether the effects on the engineers' performance are associated only with PSP level and the introduced techniques rather than number of programs experience.

To reach that objective, we considered a direct way through an Analysis of covariance (ANCOVA). As we explained in *Chapter 3*, the covariance is a measure of how much two variables change together and how strong the relationship is between them. The ANCOVA is a general linear model which blends ANOVA and regression. ANCOVA evaluates whether population means of a dependent variable (DV) are equal across levels of a categorical independent variable (IV), while statistically controlling for the effects of other continuous variables that are not of primary interest, known as covariates (CV). Therefore, when performing ANCOVA, we are adjusting the DV means to what they would be if all groups were equal on the CV.

Based on these ideas, we try to solve our problem by using the corresponding dependent variable according to the hypothesis to be studied; using PSPI/II and PSP Fund/Adv as categorical values to segment the data into two parts; using the program assignment as the independent variable; and using the PSP level as the hidden variable.

When attempting to apply ANCOVA, we discovered that the correlation between PSP level and program number across the two courses was stronger than we believed. As explained in Chapter 3, correlations between the factors are a strong anti-indication for ANCOVA [14]. Therefore, as we could not satisfy the ANCOVA assumptions, we decided to create a more indirect procedure and analyze the results based on specific differences between the two courses using the PSP level.

To develop this indirect procedure we examine some relationships between program number, PSP level, course type and engineers' performance, which is represented by the value of the dependent variable under study. The following procedure consists of three steps, each one based on an ANOVA analysis using the corresponding dependent variable for each hypothesis.

To present this approach, let us define DepVar as a dependent variable to be studied. All the three presented steps will be related to the study of this unique dependent variable. If we want to study other dependent variables, we should execute all the three step approach again.

The first step is like a gate, that if we are able to open it, this can lead us to think that the changes are not due to programming repetition. This step tries to find out whether there are differences between the two courses by comparing the dependent variable for each program assignment. That is, we compare the dependent variable at the program number x of PSP Fund/Adv vs. the same program number x of PSP I/II. And we made this comparison for all the program assignments. If there is no statistically significant difference, that means that the dependent variable value in the same program assignment of both courses is not changing. However, in the courses each program assignment has different PSP level. So, it seems that the changes in the PSP levels are not affecting the dependent variable changes. In this case, if any changes in the dependent variable existed through the exercises, then the exercise repetition and domain learning would be the root causes of the changes in the dependent variable. On the other hand, when we find differences, we should move forward to the second step in order to find if the PSP level could be the root cause of the changes.

Figure 9 to Figure 12 show examples of how differently a dependent variable can evolve during the courses.

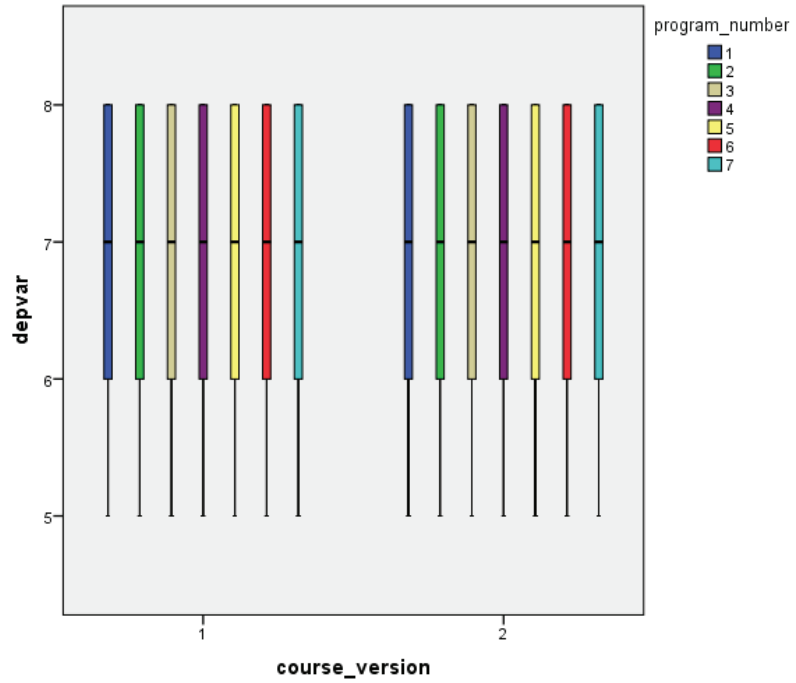


Figure 9: Example - DepVar keeps constant and unchanged during the 7 program assignments on both courses. There are no improvements.

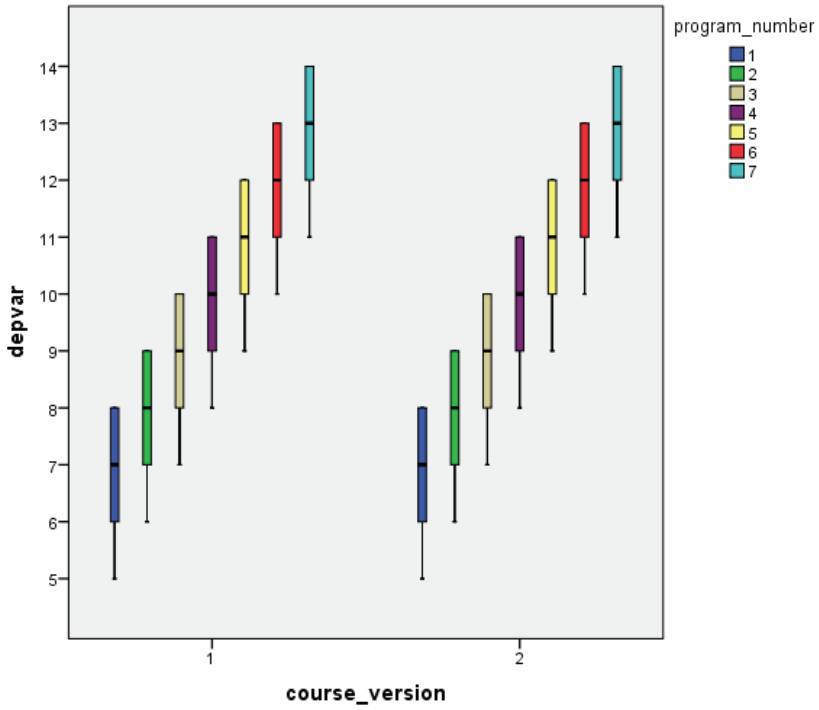


Figure 10: Example - DepVar keeps changes constantly by course during the 7 program assignments, but keeps unchanged between courses. Here the improvements are due to exercise repetition.

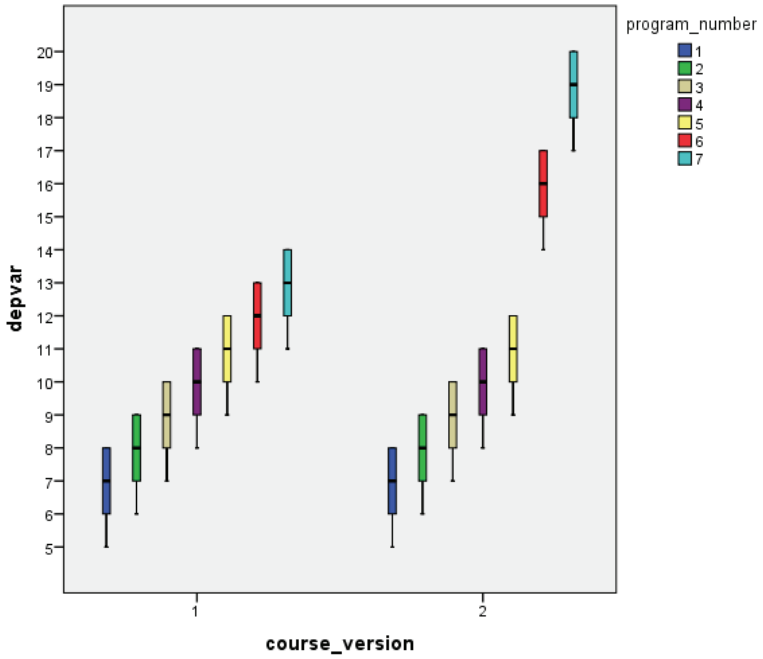


Figure 11: Example - DepVar changes with a different course pattern during the 7 program assignments, but the changes are happening on programs assignments with the same PSP level on both courses: Program 6 with PSP level 2.1, and Program 7 with PSP 2.1. Here the improvements are probably due to exercise repetition

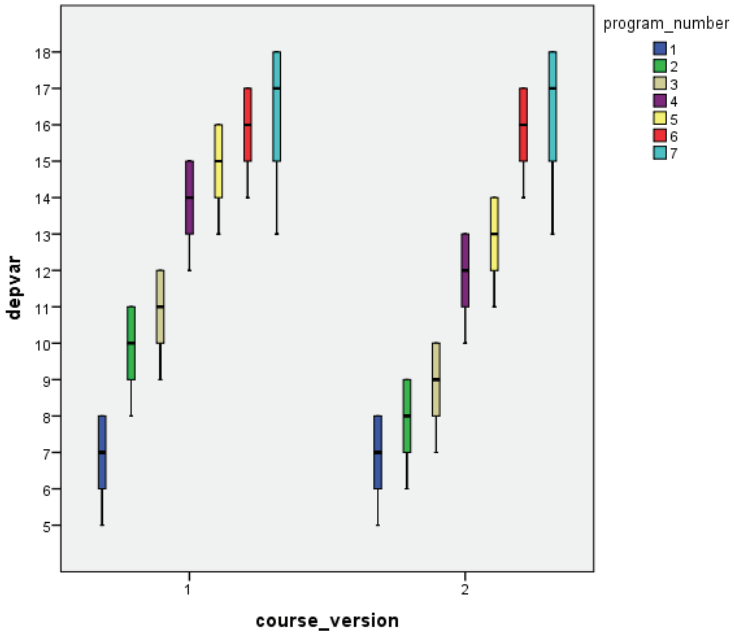


Figure 12: Example - DepVar changes with a different course pattern during the 7 program assignments, and the changes are happening on programs assignments with different PSP level on both courses: Program 2, with PSP1 in PSP Fund/Adv and PSP0 in PSP I/II; Program 2, with PSP2 in PSP Fund/Adv and PSP1 in PSP I/II; Program 5, with PSP2.1 in PSP Fund/Adv and PSP2 in PSP I/II. When this is the situation, we can move forward to the next step.

To perform the analysis for this first step, we use a series of one-way ANOVA between each program number using course as the grouping factor. This establishes if the assignments have statistically significant different means of the dependent variable

between the courses. If there are not different means between courses, then the analysis is done because the program by program results for the two courses do not differ. We must run a set of test, one for each program number. If we find significant differences in at least one program number that has different PSP levels in each course, then we proceed to the next step. It is important to be sure that the significance difference is found in a program assignment that has one PSP level in one course and another PSP level in the other course version, because that is what will allow us to discard the repetition of the main root cause of the changes.

To present the first step in a more formal way, a set of parametric tests of ANOVA are applied, one for each program number, to find out if it is possible to state that DepVar for program assignment x in PSP Fund/Adv course version is different from DepVar for the same program assignment x in PSP I/II with statistical validity. DepVar is the dependent variable and course type is the factor. The null hypothesis H0 states the means are the same; the alternative hypothesis states that they are different.

$H0_x : \mu DepVar_{x,\ PSP\ F/A} = \mu DepVar_{x,\ PSP\ I/II}$

$H1_x : \mu DepVar_{x,\ PSP\ F/A} <> \mu DepVar_{x,\ PSP\ I/II}$

Where the x refers to the program assignment numbers, which generally will go from 1 to 7 (it depends on the dependent variable under study).

Then, if at least one test, where the PSP level of the program number x is different in both courses, rejects H0 with $\alpha \leq 0.05$, we would proceed to the next step.

We know that in each course, each program assignment is completed following a specific PSP level. The second step looks at each course separately, and tries to find out if the differences between the course programs assignments are happening when the PSP level has changed or if the differences are happening even when the PSP level has no changed between two assignments. If there are significant changes between programs assignments with the same PSP level, this can lead us to think that the effects on the dependent variable are due to the repetition of exercises and not due to a specific technique introduction. Otherwise, if the significant changes are only between programs assignments with different PSP level, 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. We should also check that changes are significant according to the first step results.

To perform the analysis for the second step, we must perform a one-way ANOVA for repeated measures for each course type, using the program number as the grouping factor. This establishes a pair by pair comparison between all the assignments of each course separately. If in one course, two assignments have different means of the dependent variable, we should take a look at the applied PSP level of each assignment of that course. When they PSP level is the same, we can think that the effects are not due to the PSP level but due to exercise repetition.

Formally, in the second step, a one-way ANOVA for repeated measures is applied for each course version, to find out if it is possible to state that DepVar for program assignment x in course version Z is different from DepVar for program assignment y in course version Z with statistical validity, for each program assignment x, y where x<y; and Z= belongs to {PSP Fund/Adv, PSP I/II}. DepVar is the dependent variable and program assignment is the factor. The null hypothesis H0 states the means are the same; the alternative hypothesis states that they are different.

$H0 : \mu DepVar_{x,\ Z} = \mu DepVar_{y,\ Z}$

H1 : $\mu DepVar_{x, Z} <> \mu DepVar_{y, Z}$

Where x and y refer to all the program assignment numbers, which generally will go from 1 to 7 (it depends on the dependent variable under study), where x<y; and Z refers to one of both of the courses version, PSP Fund/Adv and PSP I/II.

The third and last step looks at each course separately again, and tries to find out if the dependant variable differences between the PSP levels are happening 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 be showing that the introduced technique is the factor affecting the engineers' performance and not the program repetition. We should also check that changes are significant according to the second step results.

To perform the analysis for this third step, we must perform a two-way ANOVA for repeated measures, using the PSP level and the course type as the grouping factors. This establishes a pair by pair comparison between all the PSP levels of each course separately. If in one course, two PSP levels have no statistically different means, we can say that the techniques introduced in that PSP level are not affecting the engineers' performance that are related to the specific dependent variable under study.

Formally, in the third step, a two-way ANOVA is applied, to find out if it is possible to state that DepVar for PSP level v in course version Z is different from DepVar for PSP level w in course version Z with statistical validity, for each PSP levels v, w where v<>w and v, w belongs to {PSP0, PSP1, PSP2, PSP2.1}; and Z belongs to {PSP Fund/Adv, PSP I/II}. DepVar is the dependet variable and PSP level and course type are the factors. The null hypothesis H0 states the means are the same; the alternative hypothesis states that they are different.

As the engineers' data is recorded by program number during the courses, to have the data by PSP level we calculate the average of the dependent variable of all the programs grouped by PSP level.

H0 : $\mu DepVar_{v, Z} = \mu DepVar_{w, Z}$

H1 : $\mu DepVar_{v, Z} <> \mu DepVar_{w, Z}$

Where v and w refers to all the PSP levels, PSP0, PSP1, PSP2 and PSP2.1; and Z refers to one of both of the courses version, PSP Fund/Adv and PSP I/II.

In the third step, the effect sizes are calculated when significant differences are found. An effect size is a measure of the strength of a phenomenon. It is known that conclusions drawn from hypothesis testing results might be erroneous if effect sizes are not judged in addition to statistical significance [35].

For each ANOVA test specified in this third step approach, we decided to apply the two-tailed significance test at 0.05. When using a two-tailed test, regardless of the direction of the relationship we hypothesize, we are testing the possibility of the relationship in both directions. For example, we may wish to compare the mean of a sample to a given value x using a t-test. Our null hypothesis is that the mean is equal to x. A two-tailed test will test both if the mean is significantly greater than x and if the mean is significantly less than x. The mean is considered significantly different from x if the test statistic is in the top 2.5% or bottom 2.5% of its probability distribution, resulting in a p-value less than 0.05.

Figure 13 shows a flowchart that represents in a clear graphic way the flow of the third step analysis procedure that we propose.

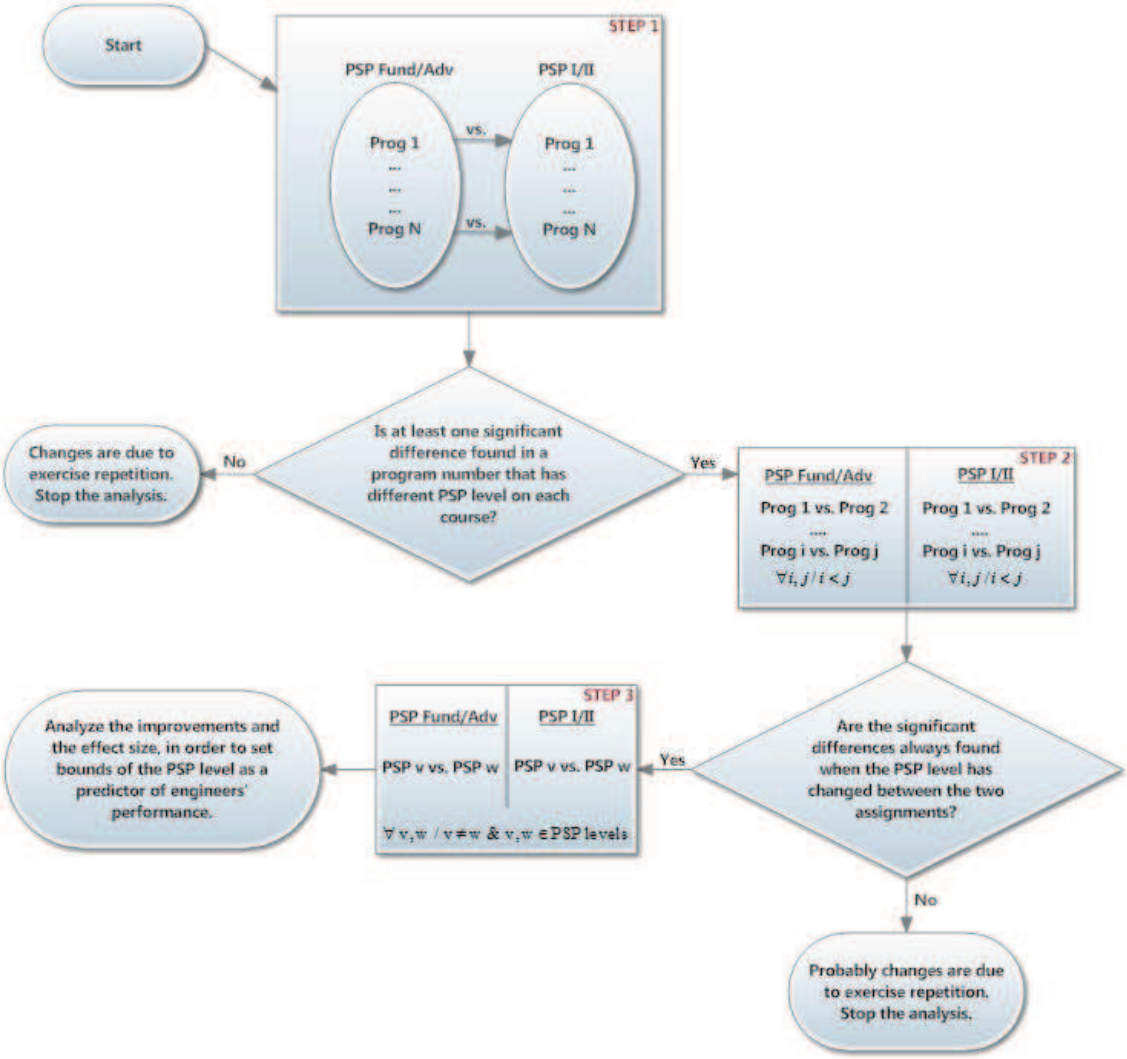


Figure 13: Three Step Analysis Approach Flowchart

3. General Threats to Validity and Limitations

The participating engineers in the PSP course have huge variety in their background, their knowledge and their experience. All those variables are influencing the collected measures and data. PSP application effects might for example be affected depending on the pre-class skills of the participants. Besides, assumptions were made on the validity of the data and the used tests. Although the data is prepared by the above-mentioned data quality cut-offs to be as complete and consistent as possible for the statistical tests, several threats to validity could be identified.

There are some threats to validity and limitations in this study that apply for all the hypotheses analysis. In this section we describe them. Later in other sections of this chapter, for each hypothesis analysis we discuss also the specific threats for that particular hypothesis.

To apply the repeated measures ANOVA some assumptions must be met:

- Subjects must be randomly selected
- Observations on these subjects are independent
- Dependent variables must be normally distributed

- Equality of variances

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

As other potential factors, a completely independent observation of the subject is almost impossible to achieve as classes are working 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 [36] [37] [38]. The transformations to reach a more normal distribution variables are explained later.

The PSP training aims at providing engineers with techniques to improve their daily work with 7 or 8 assignments, depending on the course version. The data is collected within a class set-up where the attendees can concentrate on the assignment and are not distracted by colleagues, working on multiple projects, etc. 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 seen very carefully. The results show trends and it can be interpreted 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.

We are comparing program assignments of two course versions as if they were identical. Some of them are exactly the same assignments, others are very similar. This can be considered a threat to validity, as they are not all exactly the same.

Engineers attending the PSP course are unfamiliar with the process phases and need to evolve them by using PSP and understand more about the process each time it is used. The data of the PSP courses are all collected at a stage were the engineer has the idea of “what he wants to do” but still not enough experience to stabilize his process. With the introduction of each new phase the process is getting more complex and thus, with each PSP-level, the engineer is forced to extend his knowledge on the process.

This situation may have an impact in the way some techniques are applied. So, the collected data is not really showing the use of PSP but the use during the PSP learning (which is not the same). In the practice it is expected that the data not only differ from the ones studied, but also it is expected to be better regarding to the studied hypotheses.

4. Defect Density in Unit Testing

This section presents the analysis, results, threats to validity and conclusions related to the study of the performance of the engineers regarding defect density in unit testing.

4.1 Analyses and Results

This subsection presents in detail each step of the descriptive and the statistical analysis, discussing the results of each step.

4.1.1 Descriptive Statistics

The objective of this study is to demonstrate whether reviews and design improve the quality of a product in test. And we use defect density as a measure of quality, so we define defect density in unit testing (DDUT) as the dependent variable in our analyses:

DDUT = 1000 · Total defects removed in testing / Actual added and modified LOC

As we said earlier, after the data quality process our data set was reduced to 93 subjects in total, 45 from the PSP Fund/Adv course and 48 from the PSPI/II course. The descriptive statistics for the dependent variable are displayed in Table 9.

	N	Min	Max	Media	St. dev.
Defect Density in Unit Testing	651	,00	1000,00	20,1653	54,21639

Table 9: Descriptive statistics of Defect Density in Unit Testing

The independent variables are:

- Course Type – It can be PSP Fund/Adv (labeled by "1" in plots) or PSPI/II (labeled by "2").
- Program Assignment – It can be 1, 2, 3, 4, 5, 6, 7
- PSP Level – It can be 0 (PSP0.1) , 1 (PSP1.0 and 1.1), 2 (PSP2) or 3 (PSP2.1)

In Table 9, the number of samples N is the sum of the assignments' samples considered for each student of both courses. N = 148 * 6 + 163 * 6 = 1866. The minimum and maximum values, the media and the standard deviation are also shown.

Figure 14 shows a box and whisker chart of DDUT grouped by course type and PSP level. Figure 15 shows a box and whisker chart of DDUT too, but in this case grouped by course type and program assignment.

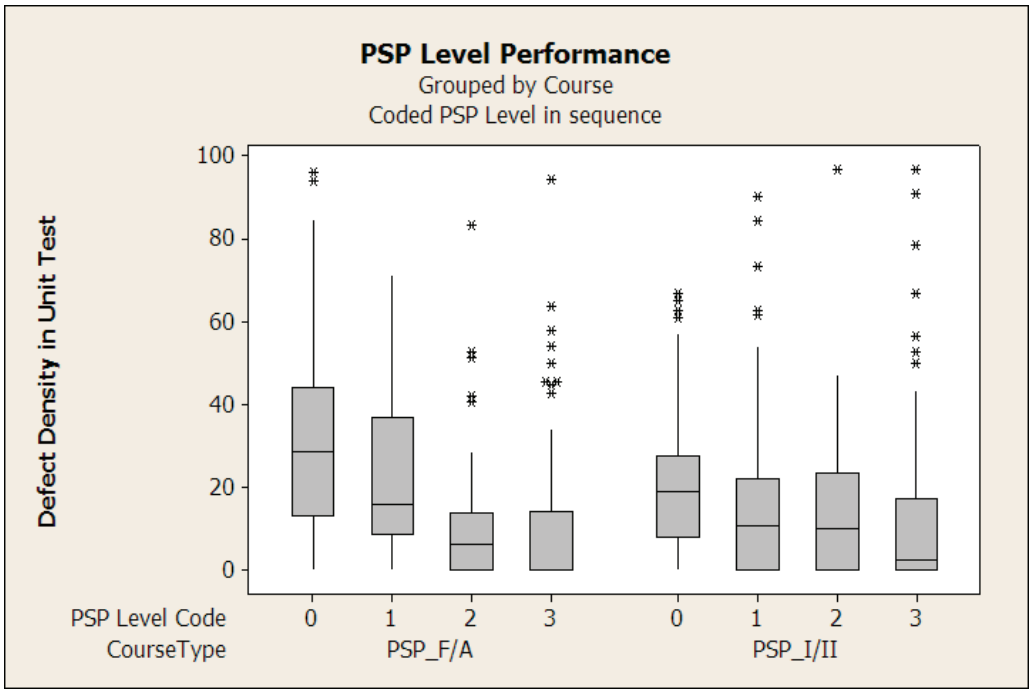


Figure 14: Box and whisker chart of DDUT for each PSP level, for both courses PSP Fund/Adv and PSP I/II

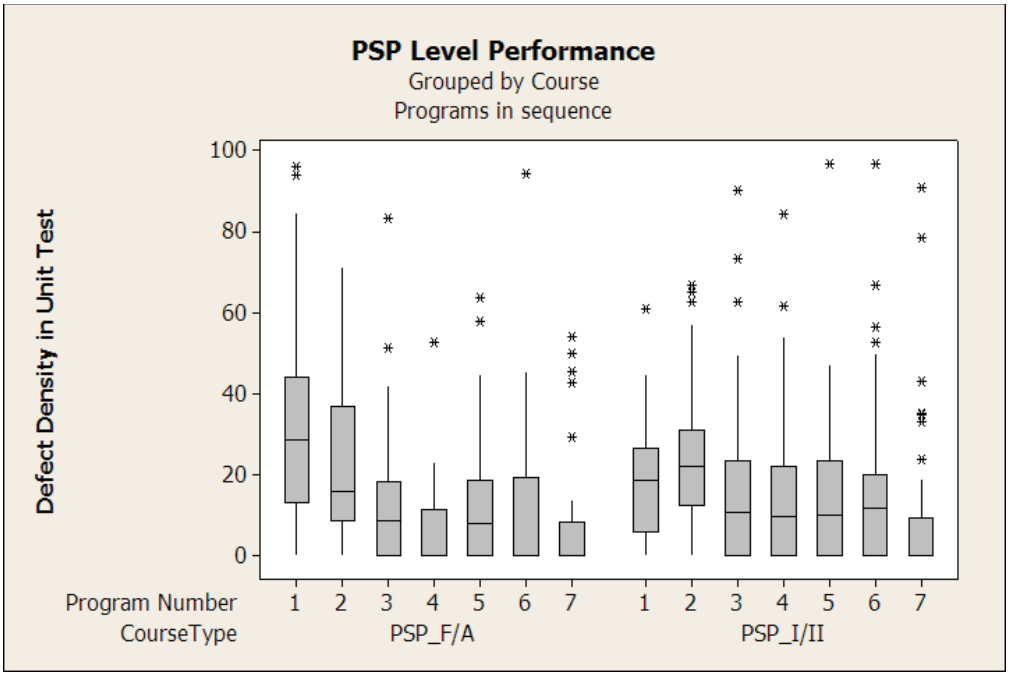


Figure 15: Box and whisker chart of DDUT for each program assignment, for both courses PSP Fund/Adv and PSP I/II

As that variable is not normal, we tried to normalize with a log-transform. Even the normality assumption could not be fully satisfied; this new variable is clearly near to a normal variable.

It is important to clarify that as total defects removed in testing can be zero, this variable is not defined on this point. That is the reason for adding a constant value of 0.5 before the log-transformation.

So, our new dependent variable is:

LN(DDUT) = LN((Total defects removed in testing / Actual added and modified LOC) + 0,5)

Basically, we did the following:

- Prior to transformation, added a constant of 0.5 to everything. This will not alter the shapes or the slopes.
- Log-transformed the data. Now have no infinities at zero.
- Performed the fits
- Untransformed the data
- Subtracted the constant from the mean to calculate the effect size and the confidence intervals.

4.1.2 Three Step Approach Analysis

To complete the statistical analysis, we must follow the three step analysis procedure that was explained in the Section 2.1.

The first step tries to find out whether there are differences between the two courses by comparing the LN(DDUT) for each program assignment. If there is no statistically significant difference, that means that the defect density in UT in the same program assignment of both courses is not changing. However, in the courses each program assignment has different PSP level. So, it seems that the changes in the PSP levels are not affecting the defect density changes. In this case, if any changes in the defect density in unit testing existed through the exercises, then the exercise repetition and domain learning would be the root causes of the changes. On the other hand, when we find differences, we should move forward to the second step in order to find if the PSP level could be the root cause of the changes.

So we applied a set of parametric tests of ANOVA, one for each program number, to find out if it is possible to state that LN(DDUT) for program assignment x in PSP Fund/Adv course version is different from LN(DDUT) for the same program assignment x in PSP I/II with statistical validity. LN(DDUT) is the dependent variable and course type is the factor. The null hypothesis H0 states the means are the same; the alternative hypothesis states that they are different.

$H0_x : \mu LN(DDUT)_{x,\ PSP\ F/A} = \mu LN(DDUT)_{x,\ PSP\ I/II}$

$H1_x : \mu LN(DDUT)_{x,\ PSP\ F/A} <> \mu LN(DDUT)_{x,\ PSP\ I/II}$

Where the x refers the program assignment numbers, which go from 1 to 7.

As at least one test where the PSP level of the program number is different in both courses rejects H0 with $\alpha \leq 0.05$, we proceed to the next step.

The second step looks at each course separately, and tries to find out if the defect density in UT differences between the course programs assignments are happening when the PSP level has changed or if the differences are happening even when the PSP level has no changed between two assignments. If there are significant changes between programs assignments with the same PSP level, this can lead us to think that the effects on the LN(DDUT) are due to the repetition of exercises and not due to the design and code review introduction. Otherwise, if the significant changes are only between programs assignments with different PSP level, then we must study (in the third step)

the behavior of the defect density in unit testing through the PSP levels, when grouping the program assignments by PSP level.

So, for this second step a one-way ANOVA was applied for each course version to find out if it is possible to state that LN(DDUT) for program assignment x in course version Z is different from LN(DDUT) for program assignment y in course version Z with statistical validity, for each program assignment x, y where x<y; and Z= belongs to {PSP Fund/Adv, PSP I/II}. LN(DDUT) is the dependent variable, and program assignment is the factor. The null hypothesis H0 states the means are the same; the alternative hypothesis states that they are different.

$H0 : \mu LN(DDUT)_{x, Z} = \mu LN(DDUT)_{y, Z}$

$H1 : \mu LN(DDUT)_{x, Z} <> \mu LN(DDUT)_{y, Z}$

Where x and y refer to all the program assignment numbers, which go from 1 to 7, where x<y; and Z refers to one of both of the courses version, PSP Fund/Adv and PSP I/II.

Table 10 and Table 11 summarize the ANOVA discussed above for this step for the courses PSP Fund/Adv and PSP I/II respectively.

PSP Fund/Adv				
Program Assignment (I)	**Program Assignment (J)**	**PSP Level**	**Mean difference (I-J)**	**Sig.**
1	2	1	,154	1,000
	3	2	1,364	,003
	4	2	2,348	,000
	5	2.1	1,602	,000
	6	2.1	2,139	,000
	7	2.1	2,347	,000
2	3	2	1,210	,015
	4	2	2,193	,000
	5	2.1	1,448	,001
	6	2.1	1,985	,000
	7	2.1	2,192	,000
3	4	2	,983	,111
	5	2.1	,237	1,000
	6	2.1	,774	,550
	7	2.1	,982	,120

4	5	2.1	-,745	,722
	6	2.1	-,208	1,000
	7	2.1	-,001	1,000
5	6	2.1	,537	1,000
	7	2.1	,744	1,000
6	7	2.1	,207	1,000

Table 10: LN(DDUT) ANOVA outputs for program assignment comparison in PSP Fund/Adv

PSP I/II				
Program Assignment (I)	**Program Assignment (J)**	**PSP Level**	**Mean difference (I-J)**	**Sig.**
1	2	0.1	-,485	1,000
	3	1	,502	1,000
	4	1.1	,730	,822
	5	2	,816	,447
	6	2.1	,948	,153
	7	2.1	1,517	,001
2	3	1	,987	,128
	4	1.1	1,216	,014
	5	2	1,301	,006
	6	2.1	1,434	,001
	7	2.1	2,003	,000
3	4	1.1	,228	1,000
	5	2	,314	1,000
	6	2.1	,446	1,000
	7	2.1	1,015	,093
4	5	2	,0854	1,000
	6	2.1	,2179	1,000
	7	2.1	,786	,599

5	6	2.1	,132	1,000
	7	2.1	,701	1,000
6	7	2.1	,569	1,000

Table 11: LN(DDUT) ANOVA outputs for program assignment comparison in PSP I/II

When we analyze the results at Table 10 and Table 11, we should look at the values that are lower or equal than 0.05 in the significance column. In the PSP Fund/Adv course we found that there is significant difference between Program 1 and Programs 3, 4, 5, 6, 7. We also found that there is significant difference between Program 2 and Program 3, 4, 5, 6, 7. That means that H0 is rejected and the LN(DDUT) means in the PSP Fund/Adv course are significantly different between Program 1 and Programs 3, to 7 and also are significantly different between Program 2 and Programs 3 to Program 7. In PSP Fund/Adv Program 2 is completed following the PSP1 script, Program 3 and 4 are completed following the PSP2 script, and Program 5 to 7 are completed following PSP 2.1 script. So, we interpret this ANOVA results as improvements between PSP0 and PSP2, between PSP0 and PSP2.1, and also improvements between PSP1 and PSP2 and between PSP1 and PSP2.1.

Regarding PSP I/II we found that there is significant difference between Program 1 and Program 7, and also that there is significant difference between Program 2 and Program 4, 5, 6, 7. In PSP I/II Program 1 and 2 are completed following PSP0; Program 4 is completed following PSP1; Programs 5 is completed following PSP2; and Programs 6 and 7 are completed following PSP2.1. So, these results are consistent with improvements between PSP0 and PSP2.1, as well as improvements between PSP0 and PSP1, PSP0 and PSP2; and improvements between PSP0 and PSP2.1.

Separate course data showed a general downward trend in defect level with program number, irrespective of process level. This was as we expected based on the correlation between PSP level and program number. As a summary of this step, we can say that for each course we only found significant difference between assignments with different PSP level. According to the design and review techniques introduced in the corresponding PSP levels, these improvements were expected.

Figure 16 shows the estimated marginal means of defect density in unit testing vs. program number, for both courses. The graphic shows how the two courses perform differently. The declining defect level is more consistent and larger in PSP Fundamentals through the introduction of PSP 2.0. Defect levels appear to be more consistent by the end of the courses.

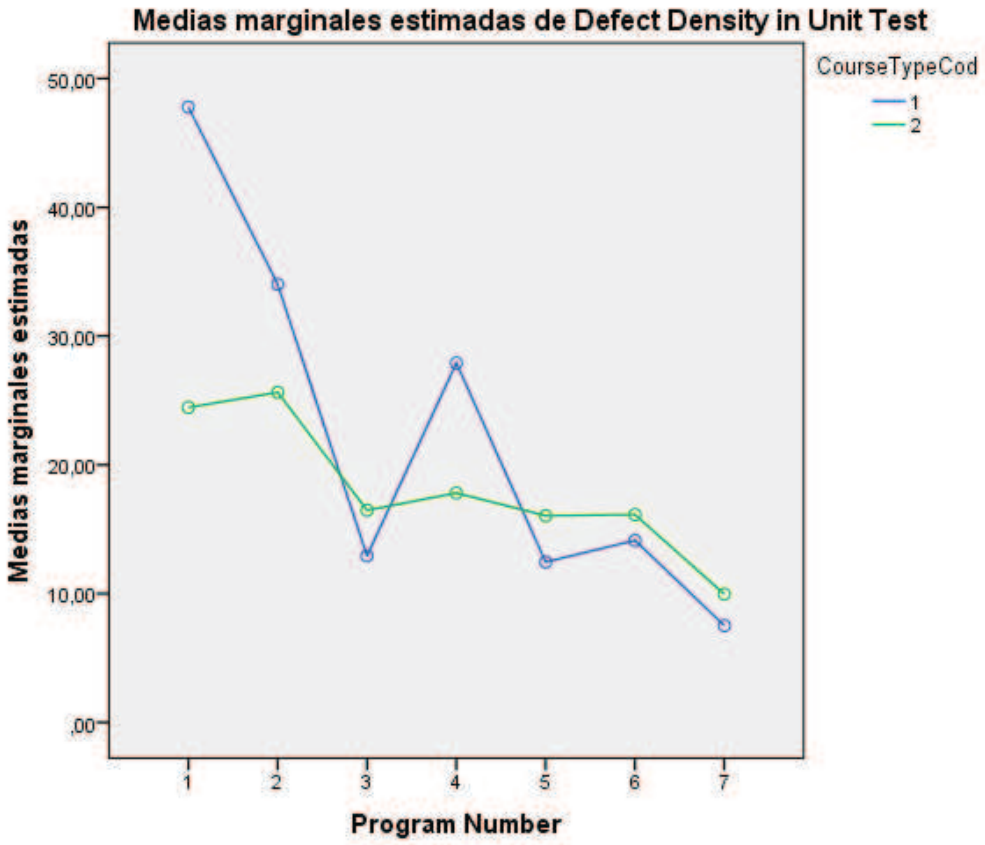


Figure 16: Estimated marginal means of DDUT vs. program number, for PSP Fund/Adv and PSP I/II

In the third and last step looks at each course separately again, and tries to find out if the defect density in unit testing differences between the PSP levels are happening when the design and code reviews are in fact introduced. If there are significant changes between PSP levels where the reviews are introduced, this will be showing that the introduced techniques are the factor affecting the engineers' performance and not the program repetition.

So, in the third step, a two-way ANOVA is applied, to find out if it is possible to state that LN(DDUT) for PSP level v in course version Z is different from LN(DDUT) for PSP level w in course version Z with statistical validity, for each PSP levels v, w where v<>w and v, w belongs to {PSP0, PSP1, PSP2, PSP2.1}; and Z belongs to {PSP Fund/Adv, PSP I/II}. LN(DDUT) is the dependent variable, PSP level and course type are the factors. The null hypothesis H0 states the means are the same; the alternative hypothesis states that they are different.

$H0 : \mu LN(DDUT)_{v, Z} = \mu LN(DDUT)_{w, Z}$

$H1 : \mu LN(DDUT)_{v, Z} <> \mu LN(DDUT)_{w, Z}$

Where v and w refers to all the PSP levels, PSP0, PSP1, PSP2 and PSP2.1; and Z refers to one of both of the courses version, PSP Fund/Adv and PSP I/II.

Looking at the two-way ANOVA results, we found that there is significant difference between PSP0 and PSP1, between PSP0 and PSP2, and also between PSP0 and PSP2. We also found that there is significant difference between PSP1 and PSP2, and between PSP1 and PSP2.1.

Figure 17 shows the 95% confidence intervals of defect density in unit testing for each PSP level, for both courses together.

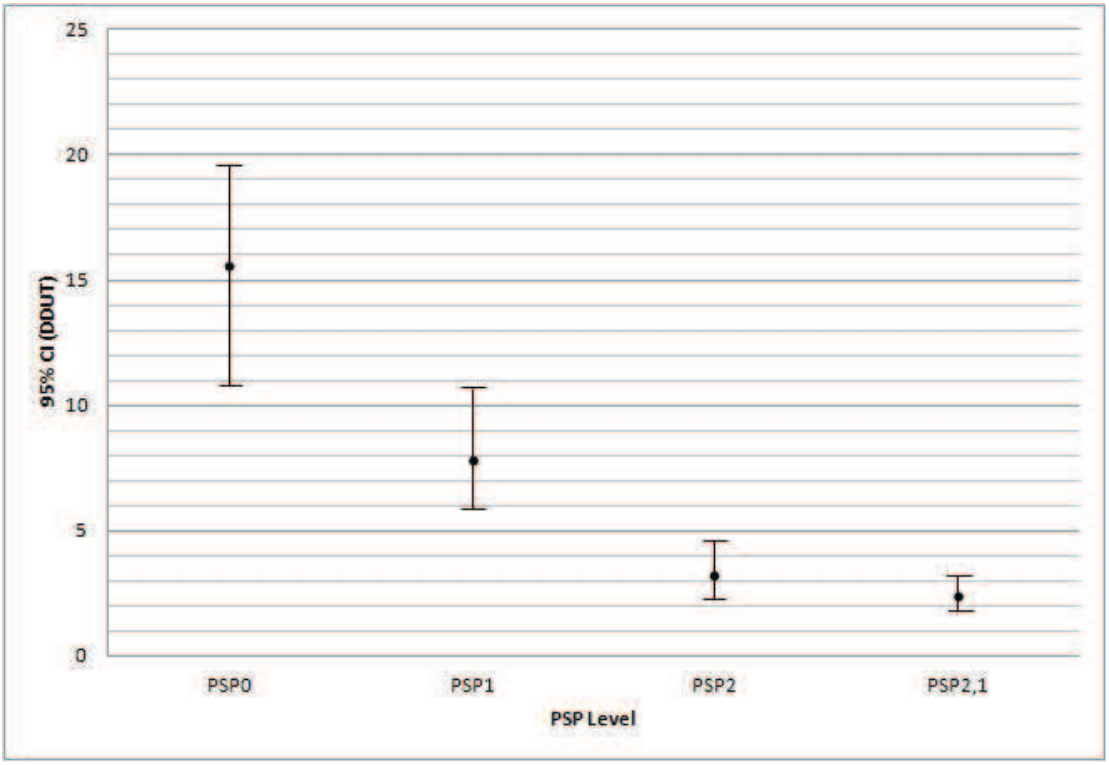


Figure 17: 95% Confidence interval of DDUT for each PSP level

Looking at both courses together in more detail, results show that:

- PSP1 was a factor of 1.52 more effective than PSP0 at an alpha level of 0.05 with a confidence range of the differences of [0.607, 2.70].
- PSP2 was a factor of 2.26 more effective than PSP0 at an alpha level of 0.05 with a confidence range of the differences of [1.965, 6.677].
- PSP2.1 was a factor of 2.28 more effective than PSP0 at an alpha level of 0.05 with a confidence range of the differences of [2.861, 8.220].
- PSP2 was a factor of 0.78 more effective than PSP1 at an alpha level of 0.05 with a confidence range of the differences of [0.809, 3.309].
- PSP2.1 was a factor of 0.89 more effective than PSP1 at an alpha level of 0.05 with a confidence range of the differences of [1.284, 4.130].

The term “more effective”, in this study means that the defect density in unit testing is reduced.

4.2 Threats to validity and limitations

By definition, defect density depends on the amount of defects removed. But the amount of defects found and removed in the test phase depends on the experience and on how good the student is in doing unit testing. Therefore, we have a threat related to testing, because the tests - that are coincident with the treatment- may influence the student behavior.

According to the history, these courses were taught largely, but not entirely at different times. Newer development environments and changes in the computer

language instruction may alter subject behavior or the defect injection profile. This could affect course differences.

Even after having transformed the data, the normality assumption for ANOVA could not be satisfied. The distributions of defect density tend to be positively skewed, with long tails extending to the right and a truncated range at zero. This type of non-normal distribution is to be expected given the source of the data. There can never be a negative count for defects, so the truncation at zero is expected. In addition, we would expect many small values of defect density and relatively fewer large values. This positively skewed distribution is particularly expected when engineers (as a group) reduce the defect density of the programs as they improve their quality during the course. This is the type of data where either a logarithmic or inverse transformation can be used to create a more nearly normal distribution [14]. Based on our examination of the effects of these two types of transformations on the distribution of residuals, the logarithmic transformation was used in the confirmatory analysis.

4.3 Conclusions

In this analysis we considered the work of 93 software engineers, who during PSP work, developed 7 or 8 programs, depending on the course version. Each subject took the complete PSP course, either PSP for Engineers I and II or PSP Fundamentals and Advance. We analyzed the data collected by each student to determine whether the design and reviews improve the quality of a product in test or if such improvement is only a consequence of gaining experience in the problem domain.

Both courses appear to be effective in demonstrating effective use of design and reviews and both show reduction in defect injections. Levels achieved at the end of the course are consistent with best in class practice. This cross course comparison allowed us to find out that a) “Hawthorne effect” is not as plausible as “gaining experience in the problem domain” or b) PSP techniques associated with PSP level as a causal explanation for the improvements. The strong association with PSP level suggests that improvement effects are most plausible regarding mastering PSP techniques rather than general domain knowledge. This might be further examined in a future study with an analysis of phase injection and removal.

Because PSP level changes so rapidly in the PSP Fund/Adv and PSP I/II program number and PSP process level are tightly correlated in a way that makes separating the effects difficult. These results cannot ensure that the observed improvements are exclusively due to mastering the process techniques introduced in the PSP. We propose future analysis in *Chapter 7* to obtain more generalizable results.

The study of this particular hypothesis was published in the Proceedings of the TSP Symposium 2012 and included in a SEI Special Report [39].

5. Yield

This section presents the analysis, results, threats to validity and conclusions related to the study of the performance of the engineers in pre-compiled defect yield

5.1 Analyses and Results

This subsection presents in detail each step of the descriptive and the statistical analysis, discussing the results of each step.

5.1.1 Descriptive Statistics

The objective is to demonstrate whether the introduction of design review and code review following PSP level 2 has a significant impact on the value of engineers' yield, or if such improvement is only a consequence of gaining experience in the problem domain.

We define Process Yield, our dependent variable, as follows:

Yield = (100 * Defects removed before the compile phase) / Defects injected before the compile phase

As we said earlier, after the data quality process, our data set was reduced to 217 subjects in total, 120 from the PSP Fund/Adv course and 97 from the PSPI/II course. The descriptive statistics for the dependent variable are displayed in Table 12.

	N	Min	Max	Media	St. dev.
Yield	1519	,00	100,00	37,4024	37,83361

Table 12: Descriptive statistics of Yield

The independent variables are:

- Course Type – It can be PSP Fund/Adv (labeled by "1" in plots) or PSPI/II (labeled by "2").
- Program Assignment – It can be 1, 2, 3, 4, 5, 6, 7
- PSP Level – It can be 0 (PSP0.1) , 1 (PSP1.0 and 1.1), 2 (PSP2) or 3 (PSP2.1)

In Table 12, the number of samples N is the sum of the assignments' samples considered for each student of both courses. N = 120 * 7 + 97 * 6 = 1866. The minimum and maximum values, the media and the standard deviation are also shown.

Figure 18 shows a box and whisker chart of Yield grouped by course type and PSP level. Figure 19 shows a box and whisker chart of Yield too, but in this case grouped by course type and program assignment.

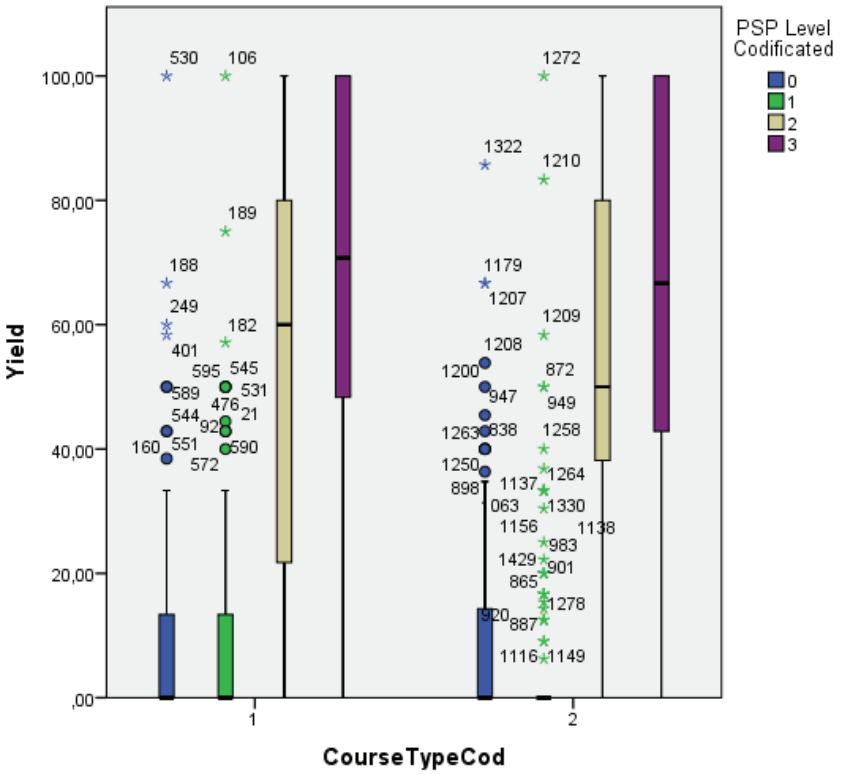


Figure 18: Box and whisker chart of Process Yield for each PSP level, for both courses PSP Fund/Adv and PSP I/II

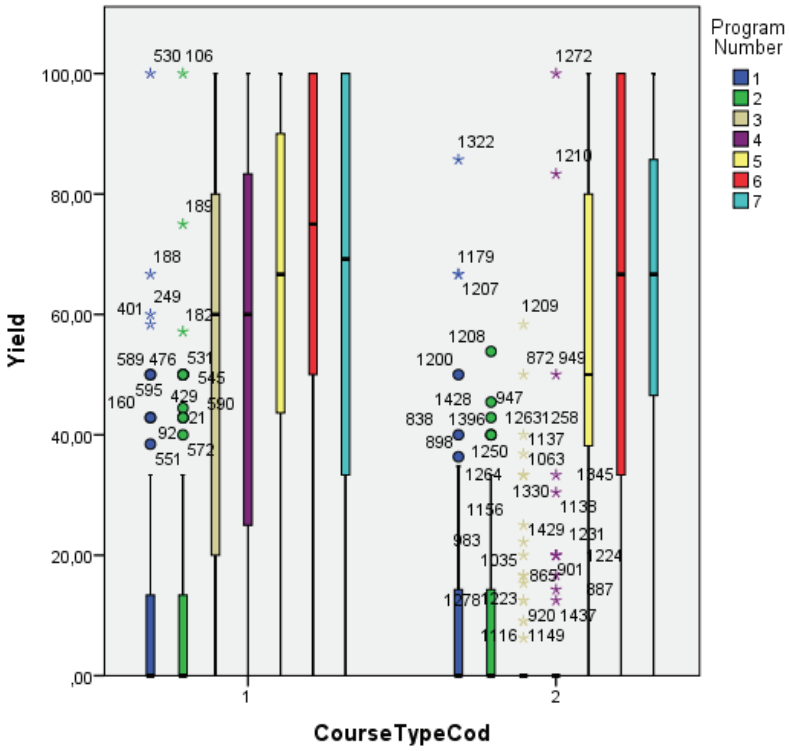


Figure 19: Box and whisker chart of Process Yield for each program assignment, for both courses PSP Fund/Adv and PSP I/II

5.1.2 Three Step Approach Analysis

To complete the statistical analysis, we must follow the three step analysis procedure that was explained in the Section 2.1.

The first step tries to find out whether there are differences between the two courses by comparing the process yield for each program assignment. If there is no statistically significant difference, that means that the yield in the same program assignment of both courses is not changing. However, in the courses each program assignment has different PSP level. So, it seems that the changes in the PSP levels are not affecting the process yield changes. In this case, if any changes in the process yield existed through the exercises, then the exercise repetition and domain learning would be the root causes of the changes. On the other hand, when we find differences, we should move forward to the second step in order to find if the PSP level could be the root cause of the changes.

So we applied a set of parametric tests of ANOVA, one for each program number, to determine if it is possible to state that the yield for program assignment x in PSP Fund/Adv course version is different from the yield for the same program assignment x in PSP I/II with statistical validity. Yield is the dependent variable, and course type is the factor. The null hypothesis H0 states the means are the same; the alternative hypothesis states that they are different.

$H0_x : \mu Yield_{x,\ PSP\ F/A} = \mu Yield_{x,\ PSP\ I/II}$

$H1_x : \mu Yield_{x,\ PSP\ F/A} <> \mu Yield_{x,\ PSP\ I/II}$

Where the x refers the program assignment numbers, which go from 1 to 7.

As at least one test where the PSP level of the program number is different in both courses rejects H0 with $\alpha \leq 0.05$, we proceed to the next step.

The second step looks at each course separately, and tries to find out if the yield differences between the course programs assignments are happening when the PSP level has changed or if the differences are happening even when the PSP level has no changed between two assignments. If there are significant changes between programs assignments with the same PSP level, this can lead us to think that the effects on the process yield are due to the repetition of exercises and not due to the design and code review introduction. Otherwise, if the significant changes are only between programs assignments with different PSP level, then we must study (in the third step) the behavior of the process yield through the PSP levels, when grouping the program assignments by PSP level.

So, for this second step a one-way ANOVA for repeated measures was applied for each course version, to find out if it is possible to state that Yield for program assignment x in course version Z is different from Yield for program assignment y in course version Z with statistical validity, for each program assignment x, y where x<y; and Z= belongs to {PSP Fund/Adv, PSP I/II}. Yield is the dependent variable, and program assignment is the factor. The null hypothesis H0 states the means are the same; the alternative hypothesis states that they are different.

$H0 : \mu Yield_{x,\ Z} = \mu Yield_{y,\ Z}$

$H1 : \mu Yield_{x,\ Z} <> \mu Yield_{y,\ Z}$

Where x and y refer to all the program assignment numbers, which go from 1 to 7, where x<y; and Z refers to one of both of the courses version, PSP Fund/Adv and PSP I/II.

Table 13 and Table 14 summarize the ANOVA discussed above for this step for the courses PSP Fund/Adv and PSP I/II respectively.

PSP Fund/Adv				
Program Assignment (I)	**Program Assignment (J)**	**PSP Level**	**Mean difference (I-J)**	**Sig.**
1	2	1	,354	1,000
	3	2	-42,488	,000
	4	2	-45,780	,000
	5	2.1	-50,867	,000
	6	2.1	-57,087	,000
	7	2.1	-52,325	,000
2	3	2	-42,841	,000
	4	2	-46,134	,000
	5	2.1	-51,221	,000
	6	2.1	-57,441	,000
	7	2.1	-52,679	,000
3	4	2	-3,293	1,000
	5	2.1	-8,380	,807
	6	2.1	-14,600	,008
	7	2.1	-9,838	,370
4	5	2.1	-5,087	1,000
	6	2.1	-11,307	,146
	7	2.1	-6,545	1,000
5	6	2.1	-6,220	1,000
	7	2.1	-1,458	1,000
6	7	2.1	4,762	1,000

Table 13: Yield ANOVA outputs for program assignment comparison in PSP Fund/Adv

PSP I/II				
Program Assignment (I)	**Program Assignment (J)**	**PSP Level**	**Mean difference (I-J)**	**Sig.**
1	2	0.1	2,009	1,000
	3	1	5,105	1,000
	4	1.1	5,367	1,000
	5	2	-46,287	,000
	6	2.1	-50,577	,000
	7	2.1	-50,972	,000
2	3	1	3,096	1,000
	4	1.1	3,359	1,000
	5	2	-48,296	,000
	6	2.1	-52,586	,000
	7	2.1	-52,980	,000
3	4	1.1	,263	1,000
	5	2	-51,392	,000
	6	2.1	-55,682	,000
	7	2.1	-56,076	,000
4	5	2	-51,654	,000
	6	2.1	-55,944	,000
	7	2.1	-56,339	,000
5	6	2.1	-4,290	1,000
	7	2.1	-4,685	1,000
6	7	2.1	-,394	1,000

Table 14: Yield ANOVA outputs for program assignment comparison in PSP I/II

When we analyze the results at Table 13 and Table 14, we should look at the values that are lower or equal than 0.05 in the significance column. In the PSP Fund/Adv course we found that there are significant differences between Program 1 and Programs 3, 4, 5, 6, 7. That means that H0 is rejected and the Yield means in the PSP Fund/Adv course are significantly different between Program 2 and Program 3,4,5,6 and 7. In PSP Fund/Adv Program 1 is completed following the PSP0 script, Program 3

and 4 are completed following the PSP 2 script, and Program 5 to 7 are completed following the PSP2.1 script. So, we interpret that this shows improvements between PSP0 and PSP 2 or PSP2.1 (depending on which assignment). Exactly the same happens between Program 2 and all the next assignments. In this course, Program 2 is completed following the PSP1 script, se we interpret that this shows improvements between PSP1 and PSP 2 or PSP2.1. We also found that there is significant difference between Program 3 and Programs 6, so we consider that this shows improvements between PSP2 and PSP2.1.

In the PSP I/II Adv course we found that there is significant difference between Program 1 and Program 5, 6, 7. Exactly the same happens between Program 2 and 5, 6, 7; between Program 3 and 5, 6, 7; and between Program 4 and all the next assignments. In PSP I/II course, Program 1 and 2 are completed following PSP0; Program 3 and 4 are completed following PSP1; Program 5 following PSP 2; Program 6 and 7 are completed following PSP2. We interpret all this significant differences as improvements between PSP0 and PSP 2, PSP0 and PSP2.1, PSP1 and PSP 2, PSP1 and PSP2.1.

As a summary of this step, we can say that for each course we only found significant difference between assignments with different PSP level, 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.

Figure 20 shows the estimated marginal means of Yield vs. 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 on the first PSP2 introduction.

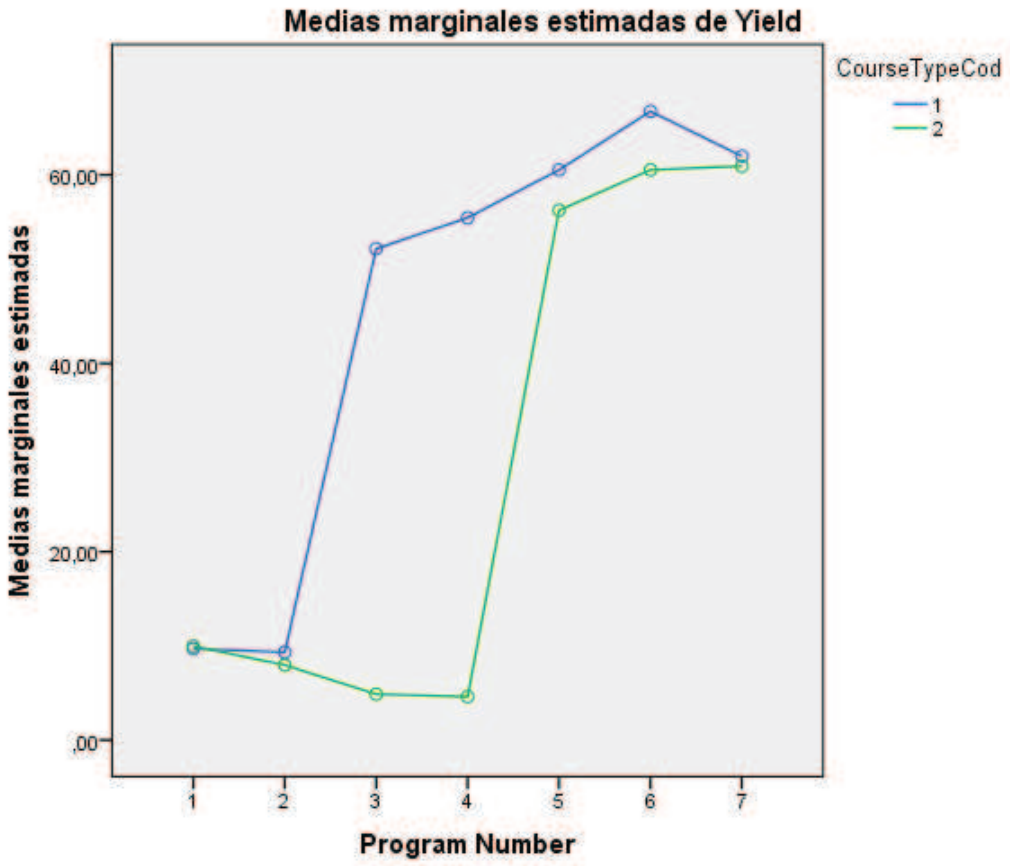


Figure 20: Estimated marginal means of Yield vs. program number, for PSP Fund/Adv and PSP I/II

In the third and last step looks at each course separately again, and tries to find out if the process yield differences between the PSP levels are happening when the design and code reviews are in fact introduced. If there are significant changes between PSP levels where these techniques are introduced, this will be showing that the design and code reviews are the factor affecting the engineers' performance and not the program repetition.

So, in the third step, a two-way ANOVA is applied, to find out if it is possible to state that Yield for PSP level v in course version Z is different from Yield for PSP level w in course version Z with statistical validity, for each PSP levels v, w where v<>w and v, w belongs to {PSP0, PSP1, PSP2, PSP2.1}; and Z belongs to {PSP Fund/Adv, PSP I/II}. Yield is the dependent variable and PSP level and course type are the factors. The null hypothesis H0 states the means are the same; the alternative hypothesis states that they are different.

H0 : $\mu Yield_{v,\,Z} = \mu Yield_{w,\,Z}$

H1 : $\mu Yield_{v,\,Z} <> \mu Yield_{w,\,Z}$

Where v and w refers to all the PSP levels, PSP0, PSP1, PSP2 and PSP2.1; and Z refers to one of both of the courses version, PSP Fund/Adv and PSP I/II.

Looking at the two-way ANOVA results, in both courses we find out that there is significant difference between PSP0 and PSP2, PSP2.1. We also found that there is significant difference between PSP1 and PSP2, PSP2.1.

Figure 21 shows the 95% confidence intervals of Yield for each PSP level, for both courses.

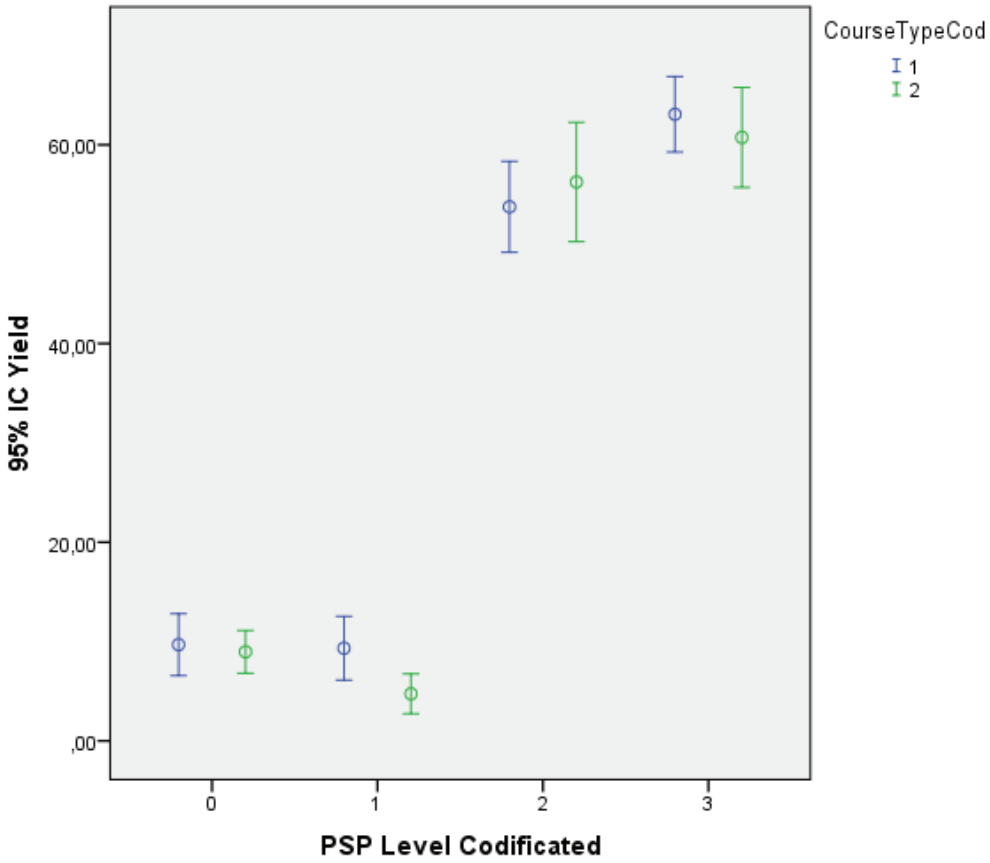


Figure 21: 95% Confidence interval of Yield for each PSP level in PSP Fund/Adv and PSP I/II

Looking at both courses together results show that:

- PSP2 was a factor of 1.72 more effective than PSP0 at an alpha level of 0.05 with a confidence range of the differences of [39.46, 51.90].
- PSP2 was a factor of 1.82 more effective than PSP1 at an alpha level of 0.05 with a confidence range of the differences of [41.71, 54.23].
- PSP2.1 was a factor of 1,83 more effective than PSP0 at an alpha level of 0.05 with a confidence range of the differences of [47.09, 58.07].
- PSP2.1 was a factor of 1.92 more effective than PSP1 at an alpha level of 0.05 with a confidence range of the differences of [49.34, 60.41].
- PSP2.1 was a factor of 0.23 more effective than PSP2 at an alpha level of 0.05 with a confidence range of the differences of [1.24, 12.57].

The term “more effective”, in this study means that the yield is improved.

5.2 Threats to validity and limitations

The normality assumption for ANOVA could not be satisfied with raw data. That is why we performed transformations on the original data and replicated the analyses with the transformed scores. When the statistical results derived from the transformed variables are consistent with those of the raw variables (and the plot of

residuals indicates a more nearly normal distribution), we can be confident that the lack of normality is not leading us astray.

The yield values for PSP levels 0 and 1 were consistently low, and only during PSP level 2 did they begin to more fully reflect the entire range of possible values. This presents us with two highly skewed distributions (PSP levels 0 and 1) and one fairly symmetrical distribution (PSP level 2 and 3). We are hard-pressed to devise a single transformation that can be applied to all of these distributions in order to make them all more normal simultaneously. Examination of the normal probability plots with the residuals from the ANOVA confirmed that the distribution for PSP level 2 and 3 is fairly normal in form, whereas the distributions for PSP levels 0 and 1 are grossly non-normal.

Given this condition, we rely on a nonparametric alternative to the repeated measures ANOVA to perform the confirmatory analysis. Results test indicate statistically significant ordered differences in yield from PSP level to PSP level. This analysis confirms the statistically significant increase in yield found in our study.

5.3 Conclusions

In this analysis we considered the work of 217 software engineers, who during PSP work, developed 7 or 8 programs, depending on the course version. Each subject took the complete PSP course, either PSP for Engineers I and II or PSP Fundamentals and Advance. We analyzed the data collected by each student to determine whether the introduction of design and code reviews improve the engineer's yield or if such improvement is only a consequence of gaining experience in the problem domain.

Both courses appear to be effective in demonstrating that the introduction and application of design and code reviews improve the engineer's yield value in a very significant way.

Early defect removal is one of the most economical ways to improve the quality of delivered software products. Preventing unplanned rework from occurring in the final stages of a software project allows more complete testing and assurance that the product will function as expected when it is delivered.

6. Production Rate

This section presents the analysis, results, threats to validity and conclusions related to the study of the performance of the engineers regarding production rate.

6.1 Analyses and Results

This subsection presents in detail each step of the descriptive and the statistical analysis, discussing the results of each step.

6.1.1 Descriptive Statistics

The objective is to demonstrate that 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 spent does not change with a higher PSP level.

But, if there is gain or loss in production rate, we want to demonstrate whether the changes are due to the introduced process or if such changes are only a consequence of gaining experience in the problem domain.

We define production rate, our dependent variable, as follows:

Production Rate = (Actual A&M LOC / Actual Minutes) * 60

As we said earlier, after the data quality process our data set was reduced to 160 subjects in total, 82 from the PSP Fund/Adv course and 78 from the PSPI/II course. The descriptive statistics for the dependent variable are displayed in Table 15.

	N	Min	Max	Media	St. dev.
Production rate	1120	,51	229,18	37,6795	26,97612

Table 15: Descriptive statistics of Production rate

The independent variables are:

- Course Type – It can be PSP Fund/Adv (labeled by "1" in plots) or PSPI/II (labeled by "2").
- Program Assignment – It can be 1, 2, 3, 4, 5, 6, 7
- PSP Level – It can be 0 (PSP0.1) , 1 (PSP1.0 and 1.1), 2 (PSP2) or 3 (PSP2.1)

In Table 15, the number of samples N is the sum of the assignments' samples considered for each student of both courses. N = 82 * 7 + 78 * 6 = 1866. The minimum and maximum values, the media and the standard deviation are also shown.

Figure 22 shows a box and whisker chart of Production rate grouped by course type and PSP level. Figure 23 shows a bar-whisker chart of Production rate too, but in this case grouped by course type and program assignment.

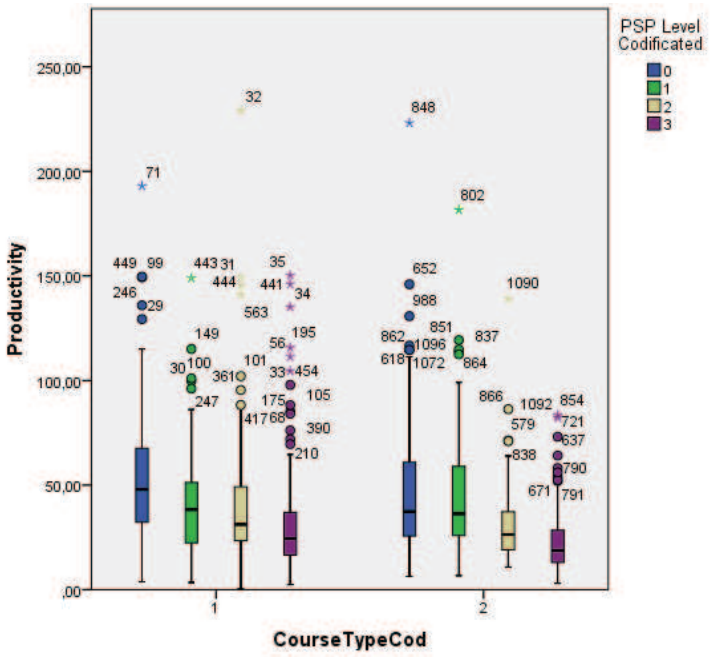


Figure 22: Bar-whisker chart of Production rate for each PSP level, for both courses PSP Fund/Adv and PSP I/II

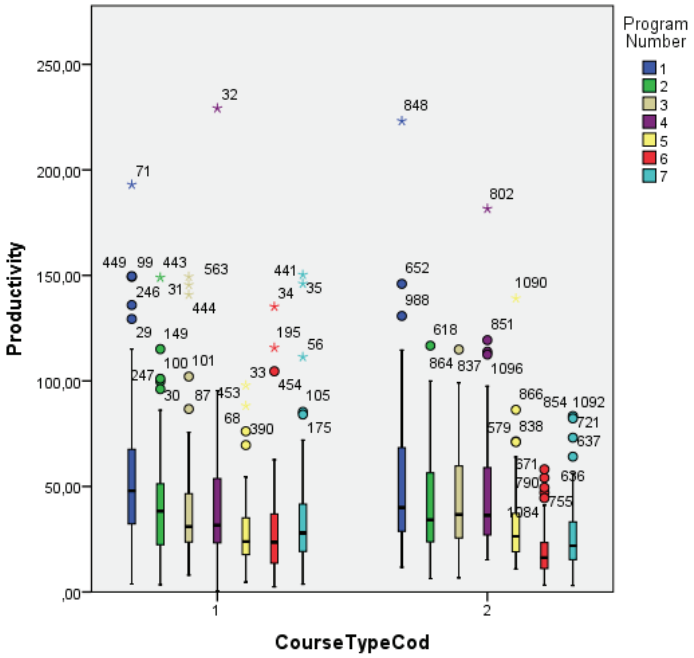


Figure 23: Bar-whisker chart of Production rate for each program assignment, for both courses PSP Fund/Adv and PSP I/II

6.1.2 Three Step Approach Analysis

To complete the statistical analysis, we must follow the three step analysis procedure that was explained in the Section 2.1.

The first step tries to find out whether there are differences between the two courses by comparing the production rate for each program assignment. And we made this comparison for all the program assignments. If there is no statistically significant difference, that means that the production rate in the same program assignment of both courses is not changing. However, in the courses each program assignment has different PSP level. So, it seems that the changes in the PSP levels are not affecting the production rate changes. In this case, if any changes in the production rate existed through the exercises, then the exercise repetition and domain learning would be the root causes of the changes. On the other hand, when we find differences, we should move forward to the second step in order to find if the PSP level could be the root cause of the changes.

So we applied a set of parametric tests of ANOVA, one for each program number, to find out if it is possible to state that Production rate for program assignment x in PSP Fund/Adv course version is different from Production rate for the same program assignment x in PSP I/II with statistical validity. ProductionRate is the dependent variable and course type is the factor. The null hypothesis H0 states the means are the same; the alternative hypothesis states that they are different.

$H0_x$: $\mu ProductionRate_{x,\ PSP\ F/A} = \mu ProductionRate_{x,\ PSP\ I/II}$

$H1_x$: $\mu ProductionRate_{x,\ PSP\ F/A} <> \mu ProductionRate_{x,\ PSP\ I/II}$

Where the x refers the program assignment numbers, which go from 1 to 7.

As at least one test where the PSP level of the program number is different in both courses rejects H0 with $\alpha \leq 0.05$, we proceed to the next step.

The second step look each course separately, and tries to find out if the production rate differences between the course programs assignments are happening when the PSP level has changed or if the differences are happening even when the PSP level has no changed between two assignments. If there are significant changes between programs assignments with the same PSP level, this can lead us to think that the effects on the production rate are due to the repetition of exercises and not due to the process changes. Otherwise, if the significant changes are only between programs assignments with different PSP level, then we must study (in the third step) the behavior of the production rate through the PSP levels, when grouping the program assignments by PSP level.

So, for this second step a one-way ANOVA for repeated measures was applied for each course version, to find out if it is possible to state that Production rate for program assignment x in course version Z is different from Production rate for program assignment y in course version Z with statistical validity, for each program assignment x, y where x<y; and Z= belongs to {PSP Fund/Adv, PSP I/II}. ProductionRate is the dependent variable, and program is the factor. The null hypothesis H0 states the means are the same; the alternative hypothesis states that they are different.

H0 : $\mu ProductionRate_{x, Z} = \mu ProductionRate_{y, Z}$

H1 : $\mu ProductionRate_{x, Z} <> \mu ProductionRate_{y, Z}$

Where x and y refer to all the program assignment numbers, which go from 1 to 7, where x<y; and Z refers to one of both of the courses version, PSP Fund/Adv and PSP I/II.

Table 16 and Table 17 summarize the ANOVA discussed above for this step for the courses PSP Fund/Adv and PSP I/II respectively.

PSP Fund/Adv				
Program Assignment (I)	**Program Assignment (J)**	**PSP Level**	**Mean difference (I-J)**	**Sig.**
1	2	1	13,069	,041
	3	2	14,573	,012
	4	2	13,645	,026
	5	2.1	26,457	,000
	6	2.1	25,336	,000
	7	2.1	20,185	,000
2	3	2	1,503	1,000
	4	2	,576	1,000
	5	2.1	13,388	,032
	6	2.1	12,266	,076

	7	2.1	7,116	1,000
3	4	2	-,927	1,000
	5	2.1	11,884	,101
	6	2.1	10,763	,222
	7	2.1	5,612	1,000
4	5	2.1	12,812	,050
	6	2.1	11,690	,116
	7	2.1	6,540	1,000
5	6	2.1	-1,122	1,000
	7	2.1	-6,272	1,000
6	7	2.1	-5,150	1,000

Table 16: Production rate ANOVA outputs for program assignment comparison in PSP Fund/Adv

PSP I/II				
Program Assignment (I)	**Program Assignment (J)**	**PSP Level**	**Mean difference (I-J)**	**Sig.**
1	2	0.1	10,534	,118
	3	1	9,635	,236
	4	1.1	7,396	1,000
	5	2	20,915	,000
	6	2.1	32,864	,000
	7	2.1	26,162	,000
2	3	1	-,899	1,000
	4	1.1	-3,138	1,000
	5	2	10,381	,133
	6	2.1	22,330	,000
	7	2.1	15,628	,001
3	4	1.1	-2,238	1,000
	5	2	11,280	,064

	6	2.1	23,229	,000
	7	2.1	16,527	,000
4	5	2	13,518	,008
	6	2.1	25,468	,000
	7	2.1	18,766	,000
5	6	2.1	11,949	,036
	7	2.1	5,247	1,000
6	7	2.1	-6,702	1,000

Table 17: Production rate ANOVA outputs for program assignment comparison in PSP I/II

When we analyze the results at Table 16 and Table 17, we should look at the values that are lower or equal than 0.05 in the significance column. In the PSP Fund/Adv course we found that there is significant difference between Program 1 and Programs 2, 3, 4, 5, 6, 7. According to the PSP levels followed to complete each assignment, we interpret that this shows loss of production rate between PSP0 and PSP1, PSP 2 or PSP2.1 (depending on which assignment). We also found that there is significant difference between Program 2 and Programs 5, and between Program 4 and Program 5. We interpret that this shows loss of production rate between PSP1 and PSP2.1, and between PSP2 and PSP2.1.

In the PSP I/II Adv course we found that there is significant difference between Program 1 and Program 5, 6, 7. According to the PSP levels followed to complete each assignment, these are consistent loss of production rate between PSP0 and PSP 2 or PSP2.1 (depending on which assignment). We found that there is significant difference between Program 2 and Program 6, 7. These are consistent with deterioration between PSP0 and PSP2.1. We also see same behavior between program 3 and Program 6, 7. These reflect deterioration between PSP1 and PSP2.1. We also found that there is significant difference between Program 4 and Program 5, 6, 7. These are consistent with deterioration between PSP1 and PSP2 or PSP2.1. Finally we found significance between Program 5 and Program 6. These are consistent with deterioration between PSP2 and PSP2.1.

As a summary of this step, we can say that for each course we only found significant difference between assignments with different PSP level. There is a deterioration of production rate as engineer's move forward in the PSP level.

Figure 24 shows the estimated marginal means of Production rate vs. program number, for both courses. The graphic shows how engineer's production rate evolves during the complete courses.

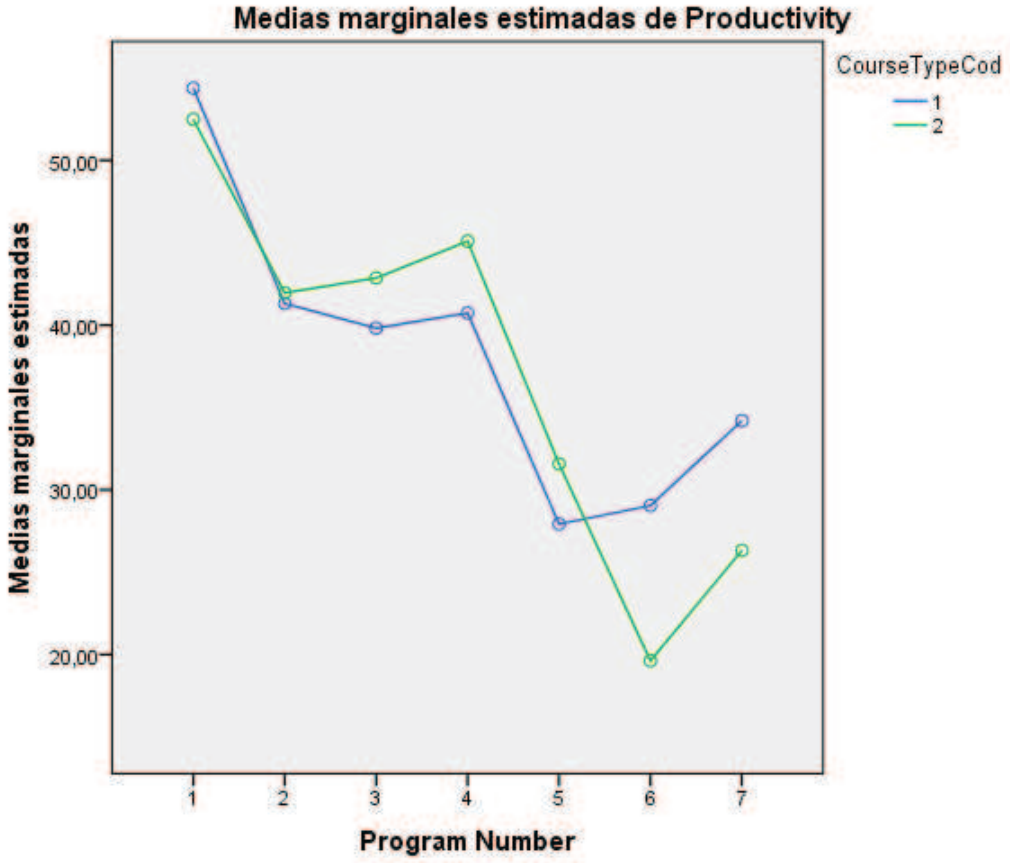


Figure 24: Estimated marginal means of Production Rate vs. program number, for PSP Fund/Adv and PSP I/II

In the third and last step looks at each course separately again, and tries to find out if the production rate differences between the PSP levels are happening when changes in the PSP levels are happening. If there are significant changes between different PSP levels, this will be showing that the process changes are the factor affecting the engineers' performance and not the program repetition.

So, in the third step, a two-way ANOVA is applied, to find out if it is possible to state that Production rate for PSP level v in course version Z is different from Production rate for PSP level w in course version Z with statistical validity, for each PSP levels v, w where v<>w and v, w belongs to {PSP0, PSP1, PSP2, PSP2.1}; and Z belongs to {PSP Fund/Adv, PSP I/II}. ProductionRate is the dependent variable, and PSP level and course type are the factors. The null hypothesis H0 states the means are the same; the alternative hypothesis states that they are different.

H0 : $\mu ProductionRate_{v, Z} = \mu ProductionRate_{w, Z}$

H1 : $\mu ProductionRate_{v, Z} <> \mu ProductionRate_{w, Z}$

Where v and w refers to all the PSP levels, PSP0, PSP1, PSP2 and PSP2.1; and Z refers to one of both of the courses version, PSP Fund/Adv and PSP I/II.

Looking at the two-way ANOVA results without course discrimination, we find out that there is significant difference between each PSP level compared in pairs.

Figure 25 shows the 95% confidence intervals of Production rate for each PSP level, considering both courses together.

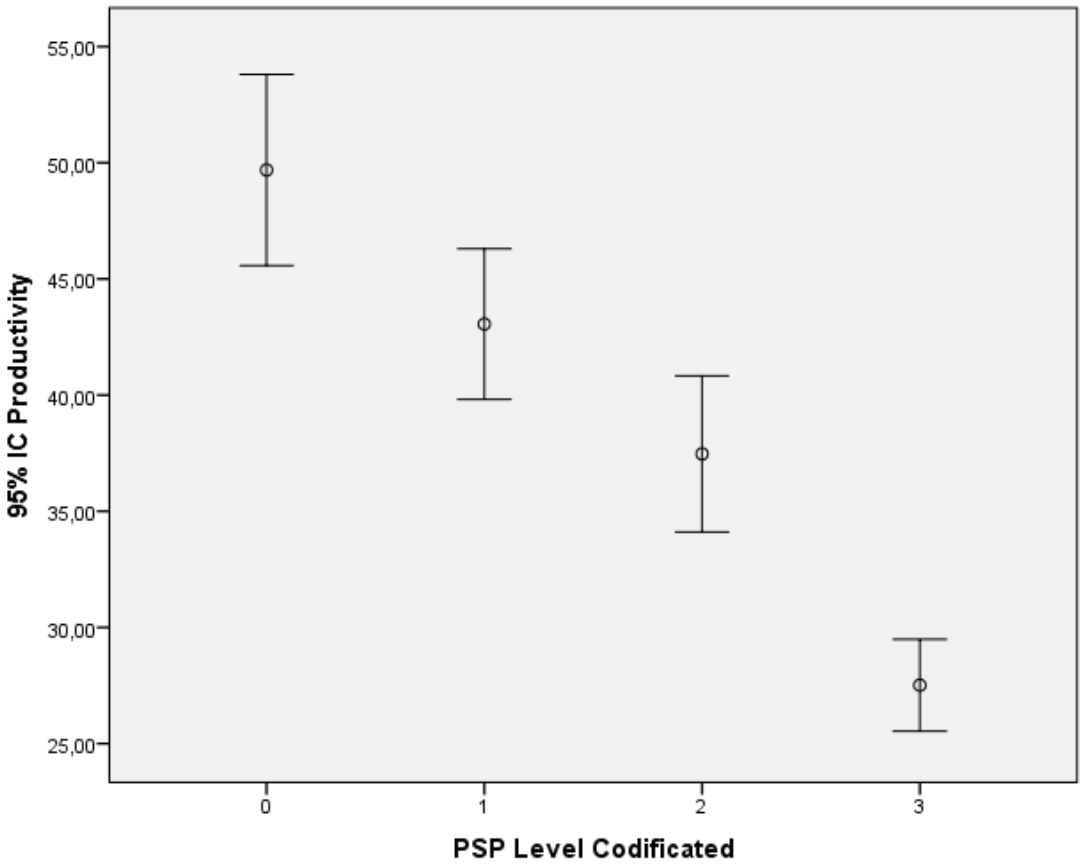


Figure 25: 95% Confidence interval of Production rate for each PSP level

Looking at both courses together results show that:

- PSP1 was a factor of 0.23 less effective than PSP0 at an alpha level of 0.05 with a confidence range of the differences of [1.68, 14.64].
- PSP2 was a factor of 0.42 less effective than PSP0 at an alpha level of 0.05 with a confidence range of the differences of [8.37, 21.39].
- PSP2.1 was a factor of 0.72 less effective than PSP0 at an alpha level of 0.05 with a confidence range of the differences of [18.39, 29.85].
- PSP2 was a factor of 0.22 less effective than PSP1 at an alpha level of 0.05 with a confidence range of the differences of [0.21, 13.23].
- PSP2.1 was a factor of 0.70 less effective than PSP1 at an alpha level of 0.05 with a confidence range of the differences of [10.23, 21.69].
- PSP2.1 was a factor of 0.44 less effective than PSP2 at an alpha level of 0.05 with a confidence range of the differences of [3.48, 15.00].

6.2 Threats to validity and limitations

The normality assumption for ANOVA could not be fully satisfied. The distributions of production rate for each PSP level show moderate positive skew. Again, this type of distribution is to be expected, given the source of the data. There can never be a production rate value of zero, since that would translate to zero lines of code per hour. While vast differences in individual production rate exist among engineers, the number of very high production rate values expected (versus very low) is relatively

small. Therefore, we see many engineers reporting moderate to low values of production rate, and a few engineers reporting relatively high values of production rate. This type of "non-normality" is often treated with a square-root transformation. The normal probability plots of residuals for the original analysis (using "raw" values of production rate) indicated a departure from normality, so the confirmatory analysis was carried out with square root-transformed production rate values. Examination of the normal probability plots (from the analysis of transformed data) confirmed that the transformation did, in fact, result in more nearly normal distributions of residuals.

6.3 Conclusions

In this analysis we considered the work of 160 software engineers, who during PSP work, developed 7 or 8 programs, depending on the course version. Each subject took the complete PSP course, either PSP for Engineers I and II or PSP Fundamentals and Advance. We analyzed the data collected by each student in order to see how production rate evolves during the courses.

Despite of what we expected, 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. This is an important result that suggests further study or design of the courses.

With these results, we can see the production rate deterioration up to unit testing phase. However, we still need to know what happens after that. That is, how PSP impacts production rate in integration and system test phases (stages that are not included in PSP). We think that this would be an interesting investigation line to be considered as a future work. That is, evaluate the whole PSP application effects in the next stages of the software development process.

7. Size Estimation Accuracy

This section presents the analysis, results, threats to validity and conclusions related to the study of the performance of the engineers regarding the accuracy of the size estimation.

7.1 Analyses and Results

This subsection presents in detail each step of the descriptive and the statistical analysis, discussing the results of each step.

7.1.1 Descriptive Statistics

The objective is to demonstrate whether the introduction of a formal estimation technique for size in PSP level 1 improves the accuracy of engineers' size estimates, or if such improvement is only a consequence of gaining experience in the problem domain.

We define size estimation accuracy, our dependent variable, as follows:

Size Estimation Accuracy = (Estimated LOC - Actual LOC)/Estimated LOC

As we said earlier, after the data quality process our data set was reduced to 311 subjects in total, 148 from the PSP Fund/Adv course and 163 from the PSPI/II course. The descriptive statistics for the dependent variable are displayed in Table 18.

	N	Min	Max	Media	St. dev.
Size Estimation Accuracy	1866	,000	3,000	0,4,6684	,495140

Table 18: Descriptive statistics of Size Estimation Accuracy

As in PSP0, no size estimation value is requested to the student, the first assignment data of each course could not be included in this study. So, the independent variables are:

- Course Type – It can be PSP Fund/Adv (labeled by "1" in plots) or PSPI/II (labeled by "2").
- Program Assignment – It can be 2, 3, 4, 5, 6, 7
- PSP Level – It can be 0 (PSP0.1) , 1 (PSP1.0 and 1.1), 2 (PSP2) or 3 (PSP2.1)

In Table 18, the number of samples N is the sum of the assignments' samples considered for each student of both courses. N = 148 * 6 + 163 * 6 = 1866. The minimum and maximum values, the media and the standard deviation are also shown.

The first inconvenient that we found, is 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 analyze the evolution of the rest of the course, but not specifically the PSP1 introduction.

In order to get a clearer idea of the group trends we can take a look to some histograms. The distributions shown in Figure 26 illustrate the performance of engineers' size estimation for the three PSP levels, for both courses. The values plotted were derived by summing the data for the assignments in each PSP level and computing the estimation accuracy value by calculating (Estimated LOC - Actual LOC) / Estimated LOC. The distributions shown in Figure 27 illustrate the performance of engineers' size estimation for each program assignment, from 2 to 7, for both courses.

With respect to the performance of the group by PSP level, the distributions in Figure 26 show a general reduction in the distance from zero for the values of size estimation accuracy. The long 'tail' of the distribution PSP1 panel is considerably shortened by the lasts panel, which represents PSP level 2 and 2.1. In addition, the distribution appears to be more symmetrically spread around zero in the lasts panels. These observations are not so clear when viewing distributions when program number increases.

Improvement in accuracy for any given engineer (as opposite to group trends as shown in the figures mentioned above) could be seen in two different ways. First, the absolute distance of the estimation accuracy metric from zero would be reduced. Second, over the course of several assignments, both overestimates and underestimates would be seen. This is in contrast to a pattern of consistent underestimating (or overestimating).

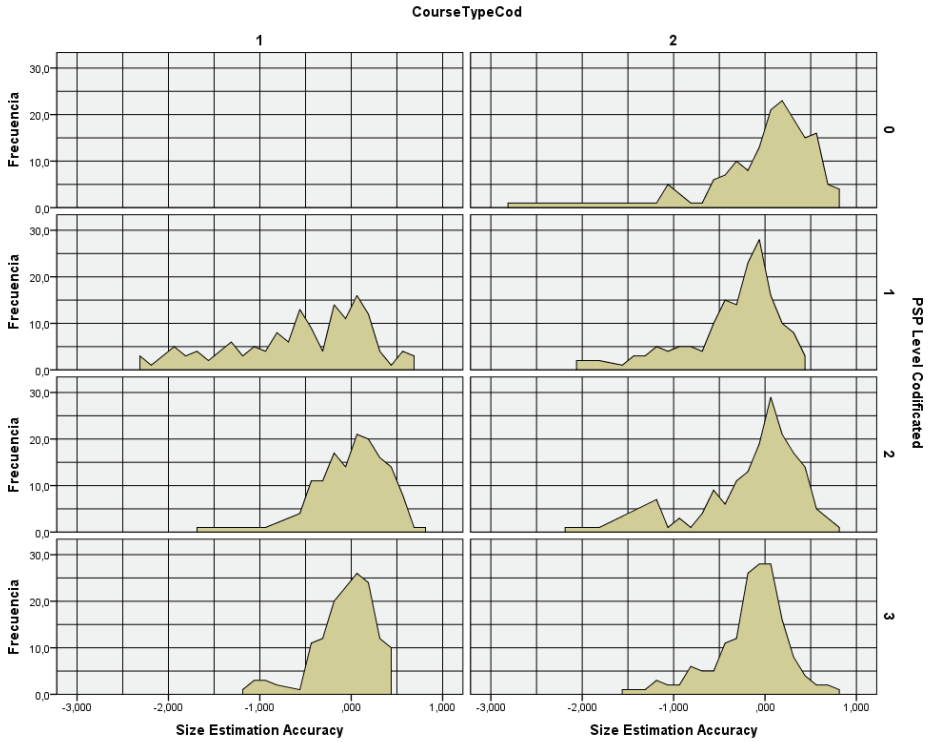


Figure 26: Distributions grouped by PSP level, for each course

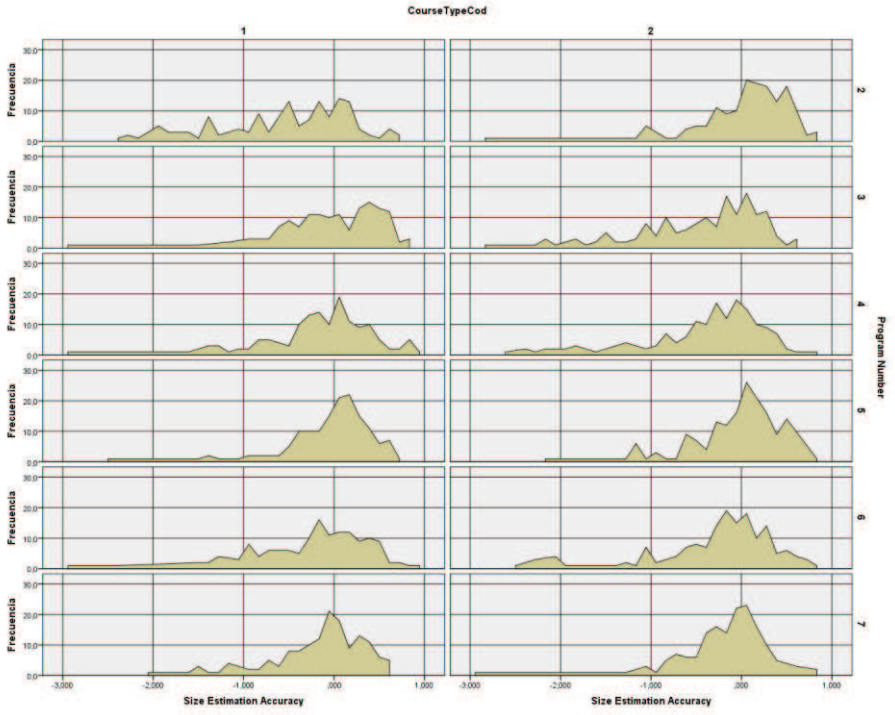


Figure 27: Distributions grouped by Program Number, for each course

Regarding to this group trends descriptive observations, we think that we have an important threat to validity, as when we summarized the data, we do not have the same amount of assignments considered on each level. For PSP Fund/Adv, there is only one assignment in PSP 1, two assignments in PSP 2 and three assignments in PSP 2.1. And for PSP I/II, there is one assignment in PSP 0, two assignments in PSP 1, one assignment in PSP 2, and three assignments in PSP2.1. So, we do not give an

opportunity to PSP0 and PSP1 to compensate their mistake among various programs. The PROBE method works by doing compensation by itself. So, it is expected that the individual will be ovestimating and underestimating. But, as there only one sample in one level, that level is not having the opportunity of the compensation.

Now that we have a general idea of the group trends, we move forward to the individual analysis of the size estimation accuracy. What we expect is that in the individual behavior, each student became to oscillate nearer to 0 after proxy technique is introduced, something similar to Figure 28.

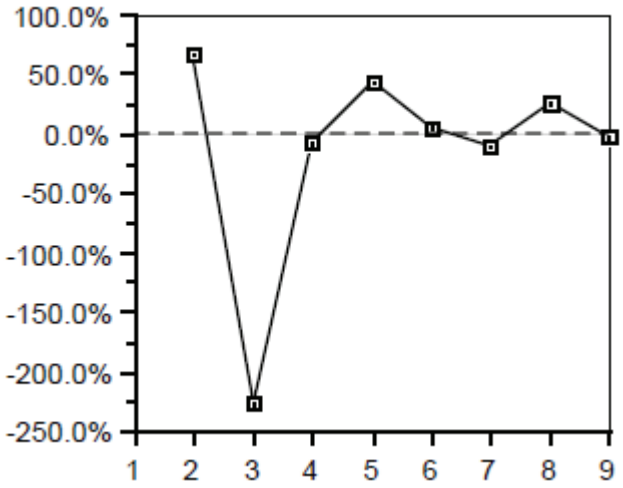


Figure 28: Example of size estimation accuracy evolution for one engineer

The ANOVA works as one would want and 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: ABS ((Estimated LOC - Actual LOC)/Estimated LOC)

As that variable is not normal, we tried to normalize with a log-transform. Even the normality assumption could not be fully satisfied; this new variable is clearly near to a normal variable.

It is important to clarify that when Estimated LOC=Actual LOC, this variable is not defined on this point. That is the reason for adding a constant value of 0.5 before the log-transformation.

So, our new dependent variable is:

LN(ABS(SEA)) = LN(ABS (Estimated LOC - Actual LOC +0,5)/Estimated LOC)

Basically, we did the following:

- Prior to transformation, added a constant of 0.5 to everything. This will not alter the shapes or the slopes.
- Log-transformed the data. Now have no infinities at zero.
- Performed the fits
- Untransformed the data
- Subtracted the constant from the mean to calculate the effect size and the confidence intervals.

Figure 29 shows a box and whisker chart of LN(ABS(SEA)) grouped by course type and PSP level. Figure 30 shows a box and whisker chart of LN(ABS(SEA)) too, but in this case grouped by course type and program assignment.

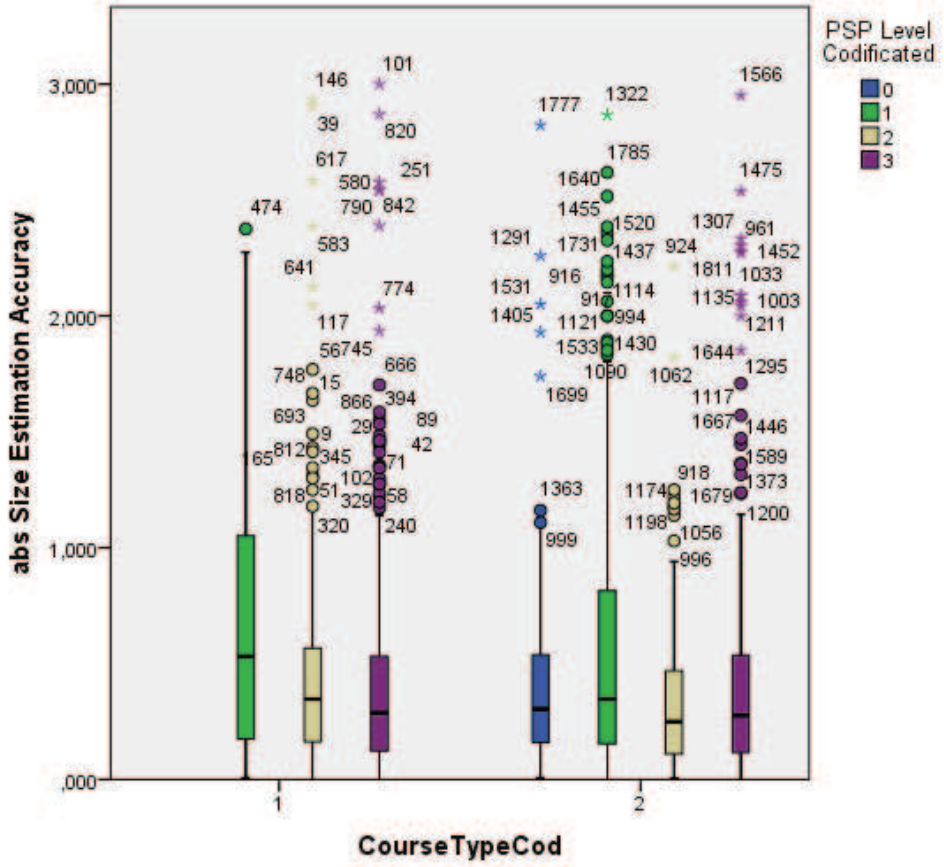


Figure 29: Box and whisker chart of ABS(SEA) for each PSP level, for both courses PSP Fund/Adv and PSP I/II

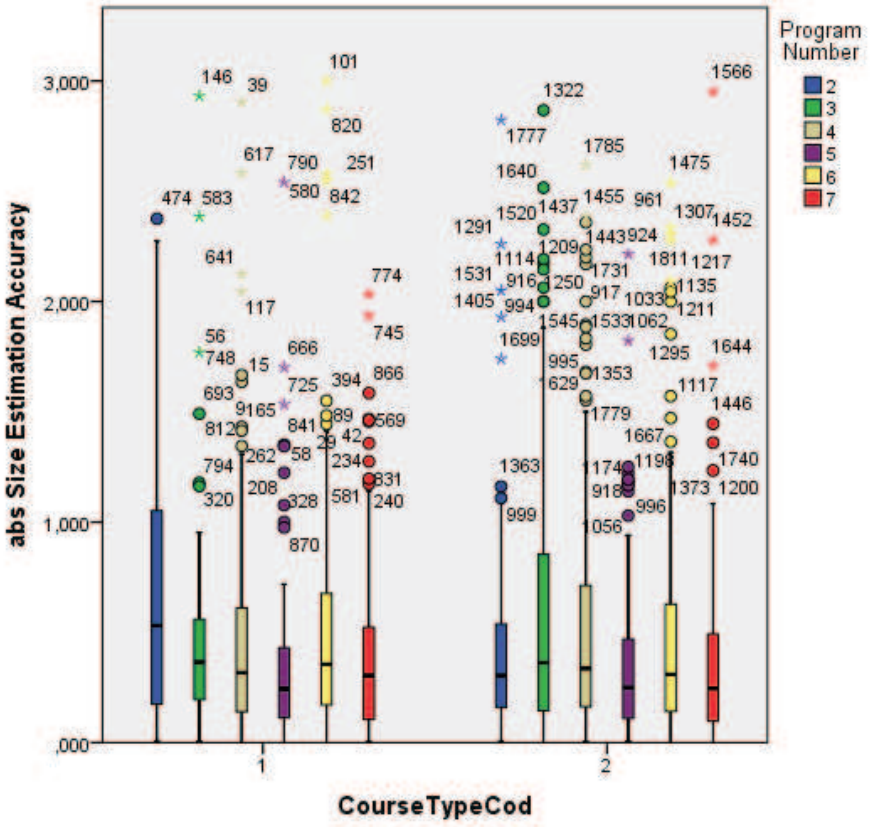


Figure 30: Box and whisker chart of LN(ABS(SEA)) for each program assignment, for both courses PSP Fund/Adv and PSP I/II

7.1.2 Three Step Approach Analysis

To complete the statistical analysis, we must follow the three step analysis procedure that was explained in the Section 2.1.

The first step tries to find out whether there are differences between the two courses by comparing the LN(ABS(SEA)) for each program assignment. If there is no statistically significant difference, that means that the size estimation accuracy in the same program assignment of both courses is not changing. However, in the courses each program assignment has different PSP level. So, it seems that the changes in the PSP levels are not affecting the dependent variable changes. In this case, if any changes in the size estimation accuracy existed through the exercises, then the exercise repetition and domain learning would be the root causes of the changes. On the other hand, when we find differences, we should move forward to the second step in order to find if the PSP level could be the root cause of the changes.

So we applied a set of parametric tests of ANOVA, one for each program number, to find out if it is possible to state that LN(ABS(SEA)) for program assignment x in PSP Fund/Adv course version is different from LN(ABS(SEA)) for the same program assignment x in PSP I/II with statistical validity. LN(ABS(SEA)) is the dependent variable and course type is the factor. The null hypothesis H0 states the means are the same; the alternative hypothesis states that they are different.

$H0_x : \mu LN(ABS(SEA))_{x,\ PSP\ F/A} = \mu LN(ABS(SEA))_{x,\ PSP\ I/II}$

$H1_x : \mu LN(ABS(SEA))_{x,\ PSP\ F/A} <> \mu LN(ABS(SEA))_{x,\ PSP\ I/II}$

Where the x refers the program assignment numbers, which go from 2 to 7.

As at least one test where the PSP level of the program number is different in both courses rejects H0 with $\alpha \leq 0.05$, we proceed to the next step.

The second step look each course separately, and tries to find out if the size estimation accuracy differences between the course programs assignments are happening when the PSP level has changed or if the differences are happening even when the PSP level has no changed between two assignments. If there are significant changes between programs assignments with the same PSP level, this can lead us to think that the effects on the LN(ABS(SEA)) are due to the repetition of exercises and not due to the estimation technique introduction. Otherwise, if the significant changes are only between programs assignments with different PSP level, then we must study (in the third step) the behavior of the size estimation accuracy through the PSP levels, when grouping the program assignments by PSP level.

So, for this second step a one-way ANOVA for repeated measures was applied for each course version, to find out if it is possible to state that LN(ABS(SEA)) for program assignment x in course version Z is different from LN(ABS(SEA)) for program assignment y in course version Z with statistical validity, for each program assignment x, y where x<y; and Z= belongs to {PSP Fund/Adv, PSP I/II}. LN(ABS(SEA)) is the dependent variable, and program assignment is the factor. The null hypothesis H0 states the means are the same; the alternative hypothesis states that they are different.

H0 : $\mu LN(ABS(SEA))_{x, Z} = \mu LN(ABS(SEA))_{y, Z}$

H1 : $\mu LN(ABS(SEA))_{x, Z} <> \mu LN(ABS(SEA))_{y, Z}$

Where x and y refer to all the program assignment numbers, which go from 2 to 7, where x<y; and Z refers to one of both of the courses version, PSP Fund/Adv and PSP I/II.

Table 19 and Table 20 summarize the ANOVA discussed above for this step for the courses PSP Fund/Adv and PSP I/II respectively.

PSP Fund/Adv				
Program Assignment (I)	**Program Assignment (J)**	**PSP Level**	**Mean difference (I-J)**	**Sig.**
2	3	2	,233	1,000
	4	2	,311	,352
	5	2.1	,606	,000
	6	2.1	,216	1,000
	7	2.1	,505	,003
3	4	2	,078	1,000
	5	2.1	,373	,102
	6	2.1	-,017	1,000

	7	2.1	,271	,718
4	5	2.1	,295	,480
	6	2.1	-,094	1,000
	7	2.1	,194	1,000
5	6	2.1	-,390	,069
	7	2.1	-,102	1,000
6	7	2.1	,288	,532

Table 19: LN(ABS(SEA)) ANOVA outputs for program assignment comparison in PSP Fund/Adv

PSP I/II				
Program Assignment (I)	**Program Assignment (J)**	**PSP Level**	**Mean difference (I-J)**	**Sig.**
2	3	1	-,156	,853
	4	1.1	-,096	,980
	5	2	,299	,225
	6	2.1	,007	1,000
	7	2.1	,322	,157
3	4	1.1	,059	,998
	5	2	,456	,009
	6	2.1	,164	,827
	7	2.1	,478	,005
4	5	2	,396	,039
	6	2.1	,104	,972
	7	2.1	,419	,023
5	6	2.1	-,291	,251
	7	2.1	,022	1,000
6	7	2.1	,314	,178

Table 20: LN(ABS(SEA)) ANOVA outputs for program assignment comparison in PSP I/II

When we analyze the results at Table 19 and Table 20, we should look at the values that are lower or equal than 0.05 in the significance column. In the PSP Fund/Adv course we found that there is only significance between Program 2 and Programs 5, 7. That means that H0 is rejected and the LN(ABS(SEA)) means in the PSP Fund/Adv course are significantly different between Program 2 and Program 5, and also are significantly different between Program 2 and Program 7. In PSP Fund/Adv Program 2 is completed following the PSP1 script and Program 5 and 7 are completed following the PSP 2.1 script. So, we interpret that this changes in the means shows improvements between PSP1 and PSP2.1.

In the PSP I/II Adv course we found that there is significant difference between Program 3 and Program 5, 7. And we also found that there is significant difference between Program 4 and Program 5, 7. That means that H0 is rejected and the LN(ABS(SEA)) means in the PSP I/II course are significantly different between Program 3 and Program 5, between Program 3 and Program 7, between Program 4 and Program 5 and also between Program 4 and Program 7. In PSP I/II Program 3 and 4 are completed following the PSP1 script, Program 5 is completed following the PSP2 script and Program 7 is completed following the PSP 2.1 script. So, these significant changes are consistent with improvements between PSP1 and PSP 2, and between PSP1 and PSP2.1 (depending on which assignment).

As a summary of this step, we can say that for each course we only found significant difference between assignments with different PSP level. According to the PROBE technique introduced, which is based on engineer historical data, these improvements were expected.

Figure 31 shows the estimated marginal means of ABS(SEA) vs. program number, for both courses. The graphic shows how the two courses perform differently, even 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.

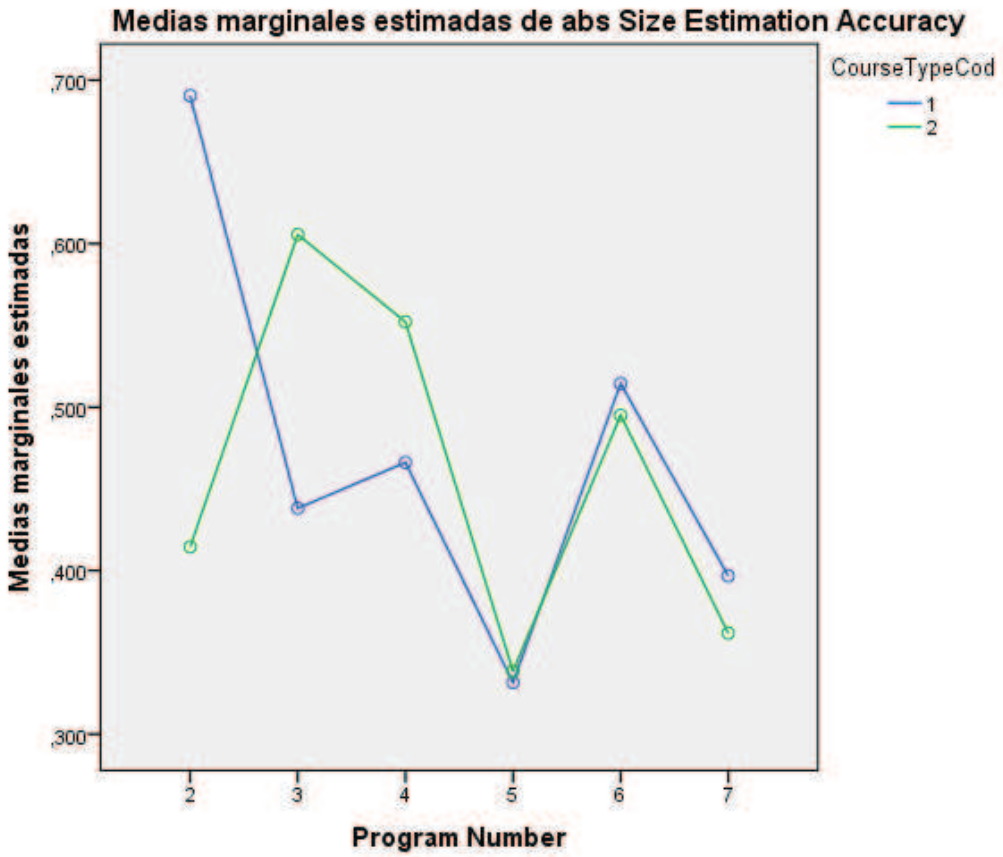


Figure 31: Estimated marginal means of ABS(SEA) vs. program number, for PSP Fund/Adv and PSP I/II

In the third and last step looks at each course separately again, and tries to find out if the size estimation accuracy differences between the PSP levels are happening when the PROBE method is in fact introduced. If there are significant changes between PSP levels where the PROBE is introduced, this will be showing that the introduced estimation technique is the factor affecting the engineers' performance and not the program repetition.

So, in the third step, a two-way ANOVA is applied, to find out if it is possible to state that LN(ABS(SEA)) for PSP level v in course version Z is different from LN(ABS(SEA)) for PSP level w in course version Z with statistical validity, for each PSP levels v, w where v<>w and v, w belongs to {PSP0, PSP1, PSP2, PSP2.1}; and Z belongs to {PSP Fund/Adv, PSP I/II}. LN(ABS(SEA)) is the dependent variable, and PSP level and course type are the factors. The null hypothesis H0 states the means are the same; the alternative hypothesis states that they are different.

$H0 : \mu LN(ABS(SEA))_{v,Z} = \mu LN(ABS(SEA))_{w,Z}$

$H1 : \mu LN(ABS(SEA))_{v,Z} <> \mu LN(ABS(SEA))_{w,Z}$

Where v and w refers to all the PSP levels, PSP0, PSP1, PSP2 and PSP2.1; and Z refers to one of both of the courses version, PSP Fund/Adv and PSP I/II.

Looking at the two-way ANOVA results, 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 introduction of PSP1.

Regarding to the two-way ANOVA results for the PSP I/II course, we found that there is significant difference between PSP1 and PSP2, PSP2.1.

Figure 32 shows the 95% confidence intervals of absolute value of size estimation accuracy for each PSP level, for both courses.

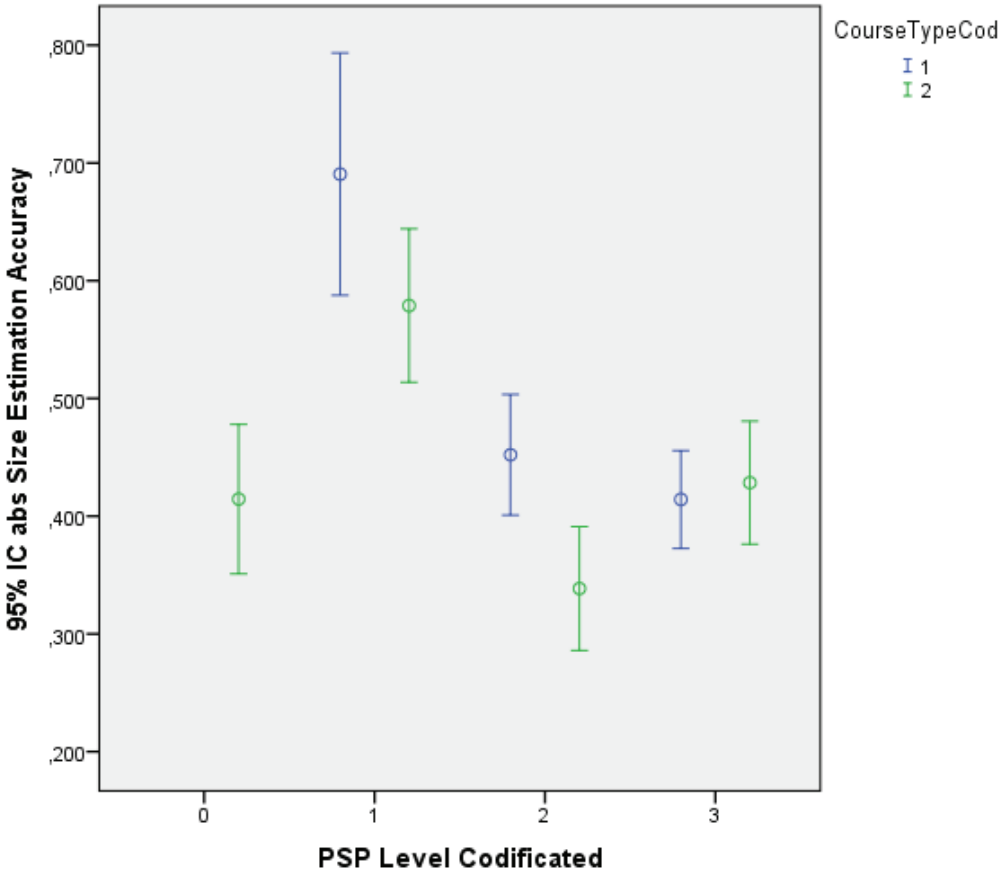


Figure 32: 95% Confidence interval of ABS(SEA) for each PSP level in PSP Fund/Adv and PSP I/II

Looking at both courses together results show that:

- PSP2 was a factor of 2.29 more effective than PSP1 at an alpha level of 0.05 with a confidence range of the differences of [1.07, 1.60].
- PSP2.1 was a factor of 2.65 more effective than PSP1 at an alpha level of 0.05 with a confidence range of the differences of [1.16, 1.66].

The term “more effective”, in this study means that the size estimation accuracy is improved.

As a summary of this step, we can say that there is significant difference between PSP1 and PSP2, and also between PSP1 and PSP2.1.

The lack of significance between PSP0 and all others PSP levels in PSP I/II may be because:

- There is only one exercise on PSP0 that can be analyzed, the difference is too small to resolve (power of the sample)
- The log-transformation hides the results (transforming those zeros will reduce the effect)

- There is no difference

With these results, we do not really see directly that the introduction of the estimation technique improves the size estimation accuracy, because in PSP2 and PSP2.1 are introduced the design and code reviews and design templates, not the estimation' techniques. Here we can say that introduction of reviews and the use of design templates is correlated with the size estimation accuracy improvement. But we are not able to see if there is a relationship between the introduction of the estimation technique and the accuracy of engineers' size estimates. Here we should remember that the introduction of the templates occurs after the estimate was made.

An interesting point here is that PSP 2 coincides with the stage at which we have enough historic data to use PROBE A. For size, this occurs on exercise 4 in PSP Fund/Adv and on exercise 5 in PSP I/II. Just estimating smaller parts does not seem to help much immediately, but there does seem to be a correlation across courses with sufficient data to use PROBE A. This could be experience, or it could be the PROBE A estimator.

It is important to remember that the estimation technique introduced in the PSP courses is based on historical data and needs repetition, so it is really hard to see with the available data that the improvements are due to the technique or due to the exercise repetition.

7.1.3 Complementary Analysis by PROBE method

In order 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. Not looking at the PSP level, but looking 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). So, we group the students by PROBE method by calculating the average of the LN(ABS(SEA)) for each student and for each PROBE method. Then we run the two-way repeated measures ANOVA grouping by PROBE method (instead of PSP level) and course type. Figure 33 shows the 95% confidence intervals of the absolute value of size estimation accuracy for each PROBE method, for both courses together.

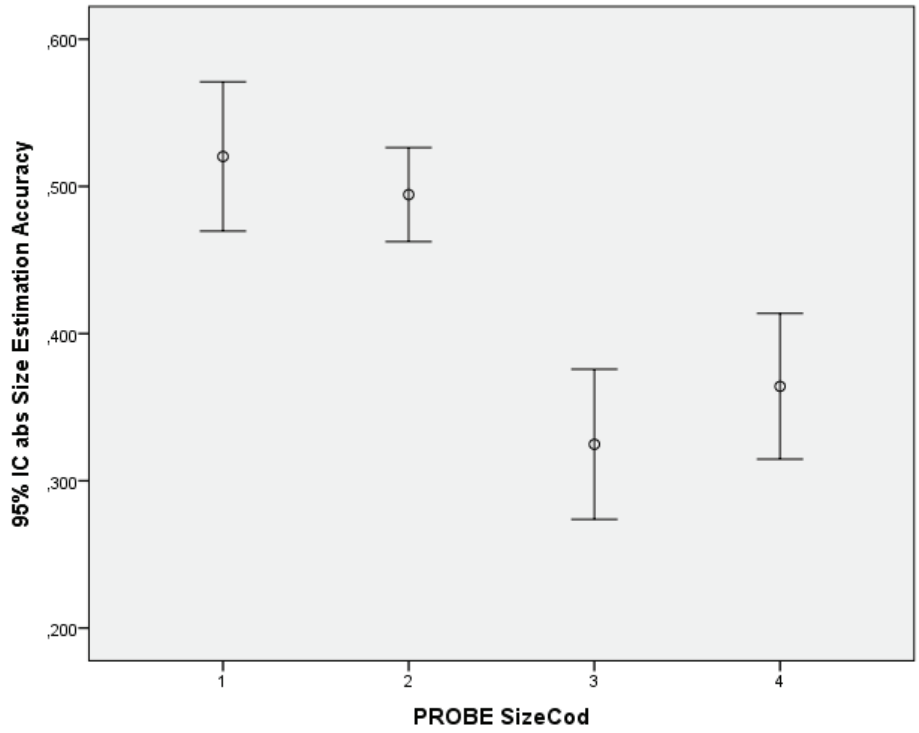


Figure 33: 95% Confidence interval of ABS(SEA) for each PROBE method. In the x-axis (PROBE SizeCode label) values 1, 2, 3 and 4 represent PROBE D, C, B and A respectively.

If we look in detail of all the students together, without doing discrimination by the course they took, we got that:

- PROBE B was a factor of 2.84 more effective than PROBE D at an alpha level of 0.05 with a confidence range of the differences of [1.04, 1.94].
- PROBE A was a factor of 2.32 more effective than PROBE D at an alpha level of 0.05 with a confidence range of the differences of [1.06, 1.75].
- PROBE B was a factor of 2.49 more effective than PROBE C at an alpha level of 0.05 with a confidence range of the differences of [1.00, 1.79].
- PROBE A was a factor of 1.95 more effective than PROBE C at an alpha level of 0.05 with a confidence range of the differences of [1.03, 1.60].

Again, the term "more effective" in this study means that the size estimation accuracy is improved. Results are different in each course and they are also different when we see them globally, without course discrimination. This can be explained by the power of the sample.

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 support 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:

- Regression (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 take multiple repetitions to stabilize
- The estimation technique is not just one technique. In fact, it is a package of three different methods, and student varies it application during the course
- The PSP level introduction on the last two courses is not the optimal to study this hypothesis

Again, repetition is necessary, and with the available data and results, we are not really able to conclude about the main reason of the improvements.

7.1.4 Another Descriptive Approach Proposal

In order to do a more complete analysis of this particular hypothesis, we decided to complement this study by showing the same data but in a different descriptive way.

This new descriptive approach consists on aggregating the data for each student and for each PROBE method to make a pair by pair comparison between each PROBE method. Having these results for each pair comparison by student, we will be able to see for example the percentage of students that produce better estimation using PROBE X than using PROBE Y, the percentage that produce worse estimations, and the percentage that suffers no changes at all. We will also be able to see if the subjects that improved are having a big improvement or not, and if the subjects that have not improved have worsened slightly or not.

For this approach, we must define a few variables. From now on, we are going to call SEA_X to the size estimation accuracy of a student when using the PROBE method X.

So, given any two PROBE methods, X and Y that we want to compare, we call division of the accuracy of the size estimation ($divE_{XY}$) to:

- SEA_X / SEA_Y , when SEA_X is greater than SEA_Y
- $- SEA_Y / SEA_X$, otherwise

This term, division of the accuracy of the size estimation, calculates the percentage of reduction in the size estimation error from one PROBE method to another (or the percentage of increase, depending on the sign).

In addition, as a rule of thumb we define that:

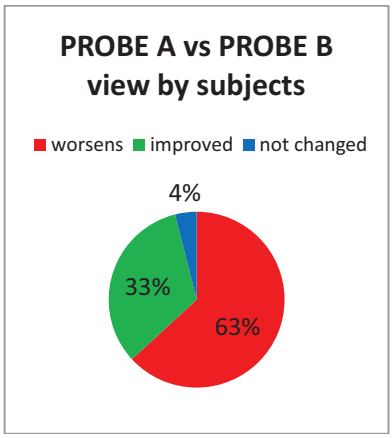
- the subject “improved” when there is a reduction of 10% or more ($divE_{XY} < -1.1$)
- the subject “worsens” when there is an increase of 10% or more ($divE_{XY} > -1.1$)
- otherwise, the subject “does not changed” ($-1.1 <= divE_{XY} <= 1.1$)

With these terms, we did all the calculus between each pair of PROBE methods:

- PROBE A vs. PROBE B
- PROBE A vs. PROBE C
- PROBE A vs. PROBE D

- PROBE B vs. PROBE C
- PROBE B vs. PROBE D
- PROBE C vs. PROBE D

Pie charts, histograms and tables summarizing the results for each comparison are shown in Figure 34 to Figure 45.



	# Subjects	DivE$_{AB}$ Average
worsens	48	5,467405802
improved	25	-10,92950108
not changed	3	0,365858664
	76	

Figure 34: PROBE A vs. PROBE B comparison - distribution and averages

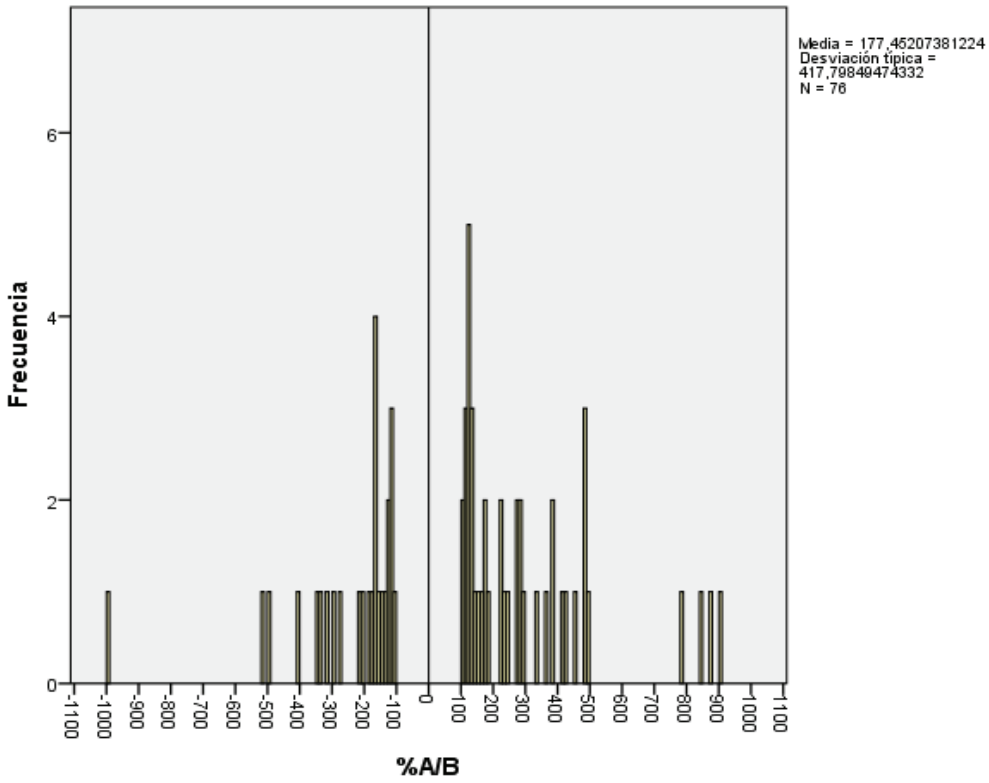
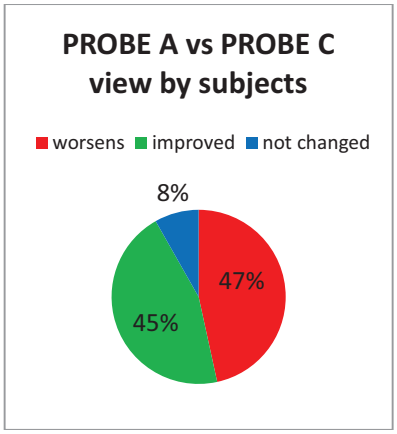


*Figure 35: PROBE A vs. PROBE B comparison – (100 * divE$_{AB}$) histogram*



	# Subjects	DivE$_{AC}$ Average
worsens	34	4,549361628
improved	33	-2,806773957
not changed	6	-0,011104197
	73	

Figure 36: PROBE A vs. PROBE C comparison - distribution and averages

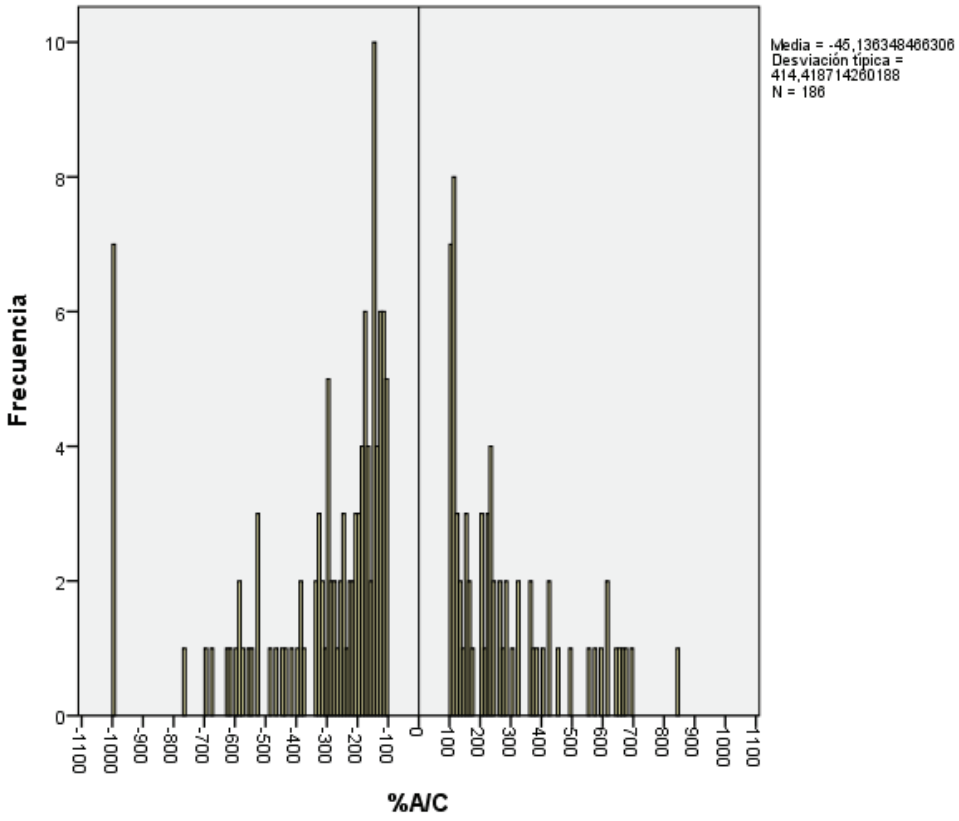
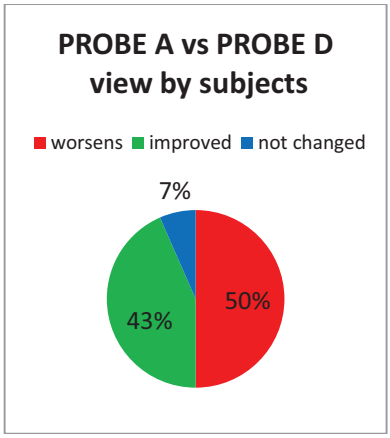


*Figure 37: PROBE A vs. PROBE C comparison – (100 * divE$_{AC}$) histogram*



	# Subjects	DivE$_{AD}$ Average
worsens	38	5,068644572
improved	33	-3,435743871
not changed	5	0,644623627
	76	

Figure 38: PROBE A vs. PROBE D comparison - distribution and averages

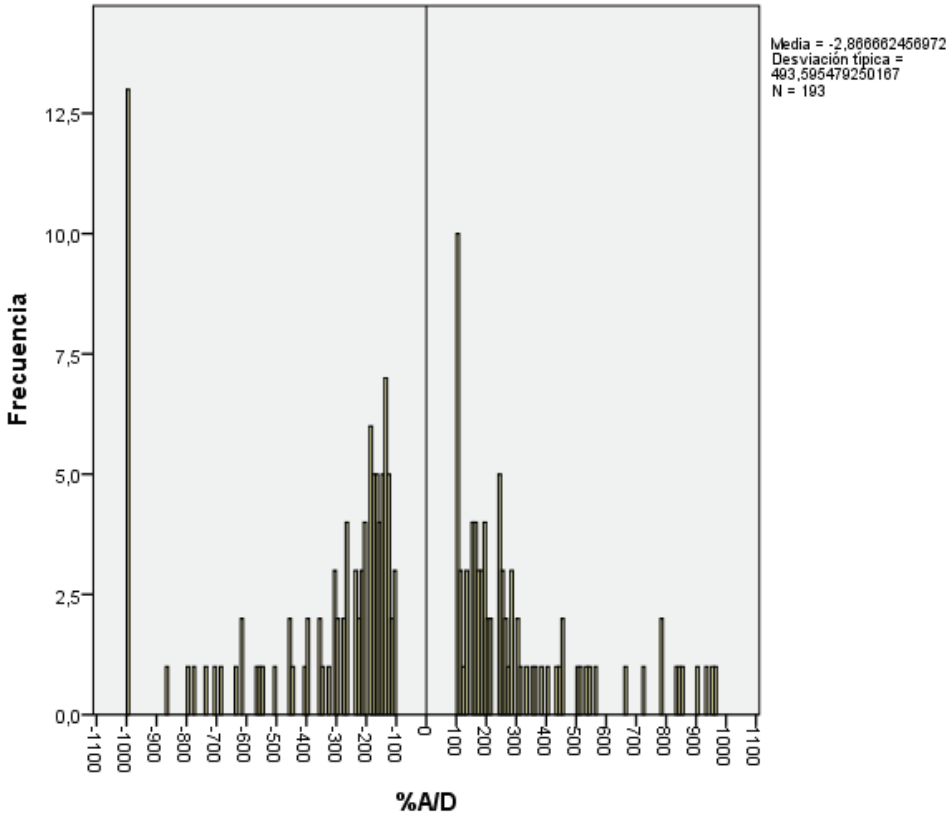
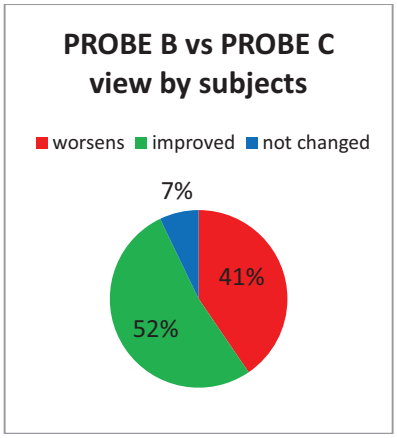


*Figure 39: PROBE A vs. PROBE D comparison – (100 * divE$_{AD}$) histogram*



	# Subjects	DivE$_{BC}$ Average
worsens	51	4,443403873
improved	66	-8,730598792
not changed	9	-0,344108207
	126	

Figure 40: PROBE B vs. PROBE C comparison - distribution and averages

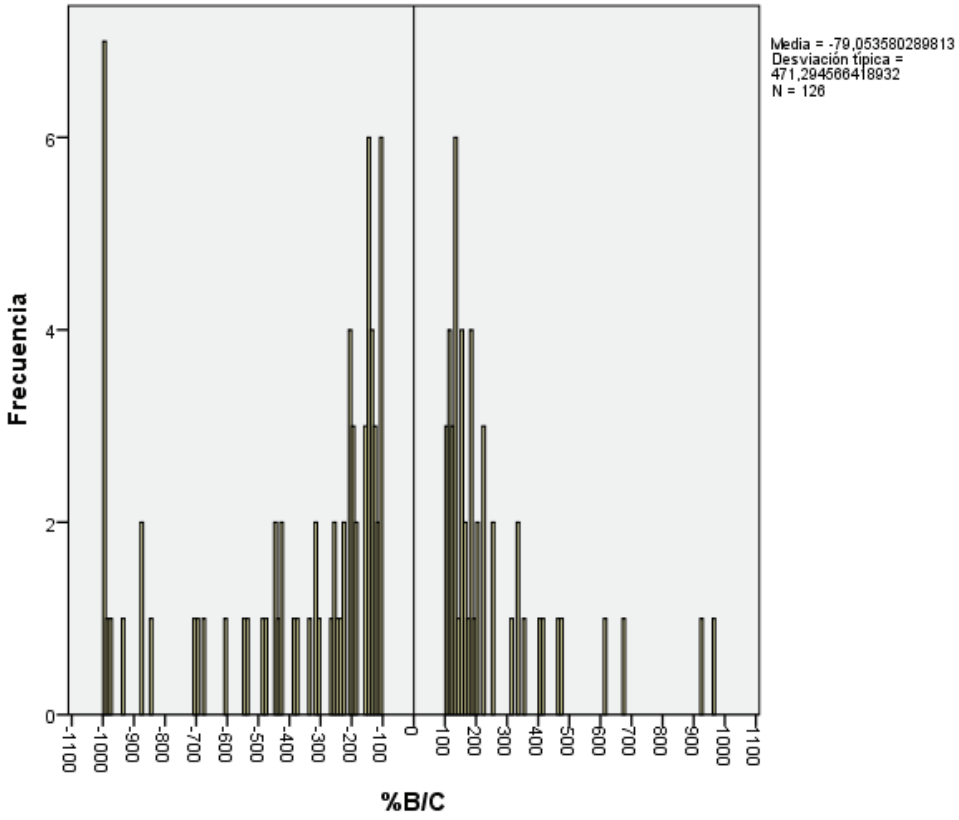
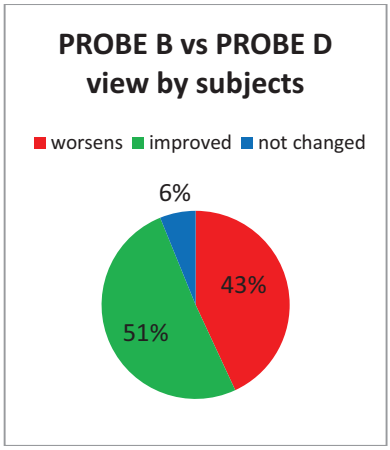


*Figure 41: PROBE B vs. PROBE C comparison – (100 * divE$_{BC}$) histogram*



	# Subjects	DivE$_{BD}$ Average
worsens	56	5,338059177
improved	66	-6,151261868
not changed	8	-0,004253836
	130	

Figure 42: PROBE B vs. PROBE D comparison - distribution and averages

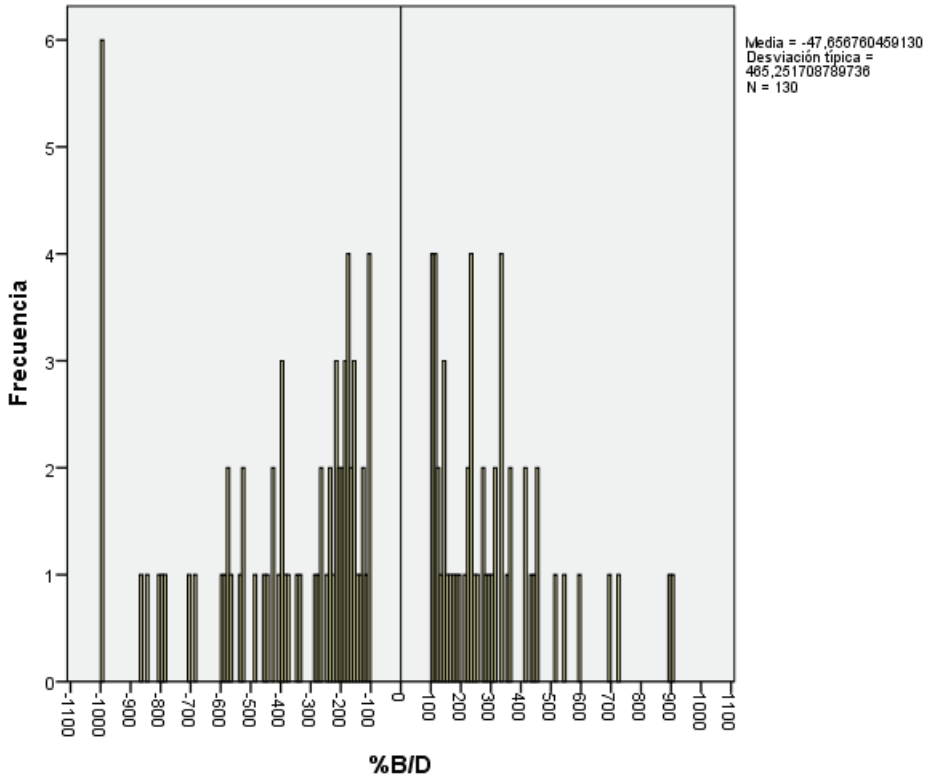
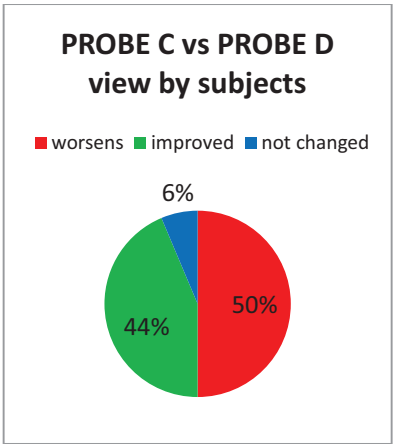


*Figure 43: PROBE B vs. PROBE D comparison – (100 * divE$_{BD}$) histogram*



	# Subjects	DivE$_{CD}$ Average
worsens	63	4,709465445
improved	55	-5,696616916
not changed	8	-0,012602948
	126	

Figure 44: PROBE C vs. PROBE D comparison - distribution and averages

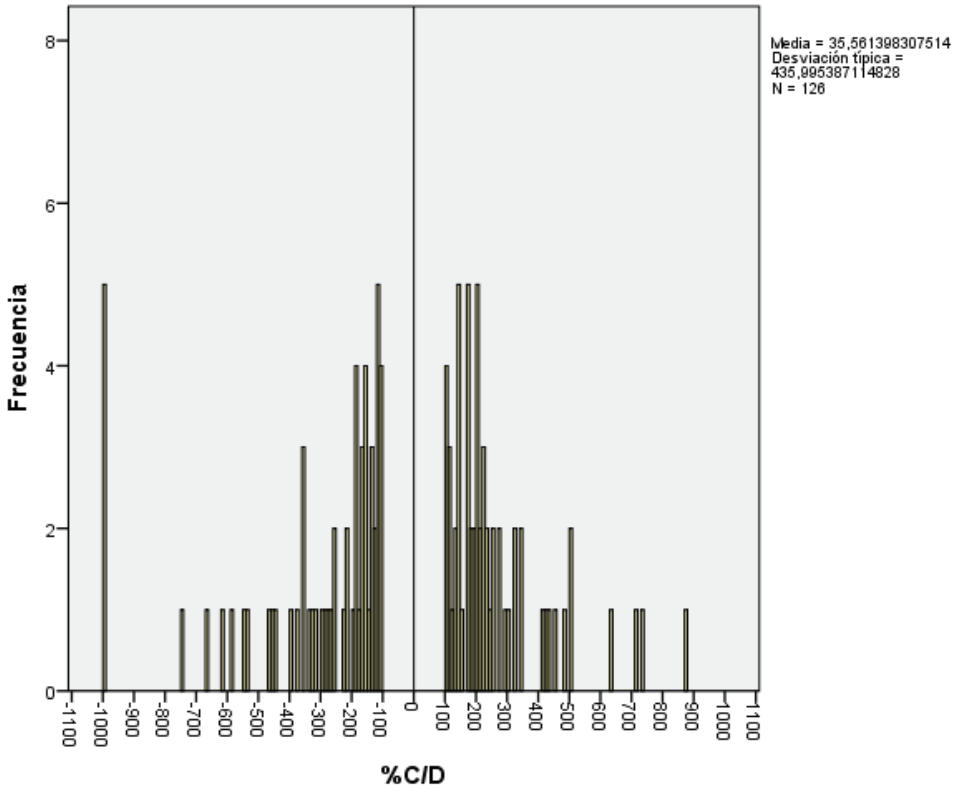


*Figure 45: PROBE C vs. PROBE D comparison – (100 * divE$_{CD}$) histogram*

With the pie charts, we can see in all the comparisons that the percentages of subjects that improved and that worsened are not very distant. In fact, the proportions are all very similar. With our definition, we can see in all comparisons that generally near half of the subjects improved and near half of the subjects worsened between any two PROBE methods.

What is interesting to see is the size of those changes. If we take a look at the PROBE A comparisons with PROBE C and with PROBE D, we can see that there are many subjects that have a big improvement (that estimates between a 500% and 1000% better with PROBE A than with the other method). We can also see that the majority of the students that have not improved, have in fact worsened their estimations by a much lower percentage. When combined with Figure 26 this seems to show that the "fat tail",

that are the worst estimates, were removed. A very similar behavior can be seen in the PROBE B vs. PROBE C histograms and PROBE B vs. PROBE D histograms.

Scatter plots for each PROBE method are presented in Figure 46 to Figure 49. There are presented the estimated size versus the actual size, were each point of the plot represents an engineer. The neaerer a point is to the y=x line, the better the estimation is. When a point is inside the line, the estimation is perfect.

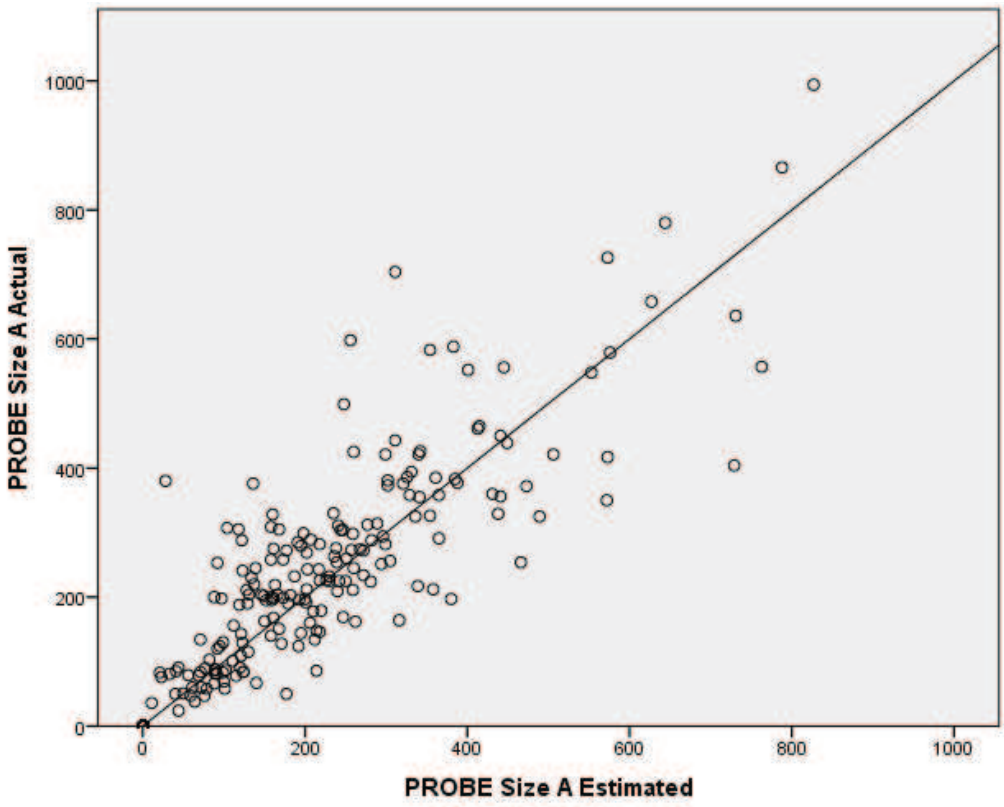


Figure 46: PROBE A - Estimated size vs Actual size scatter plot

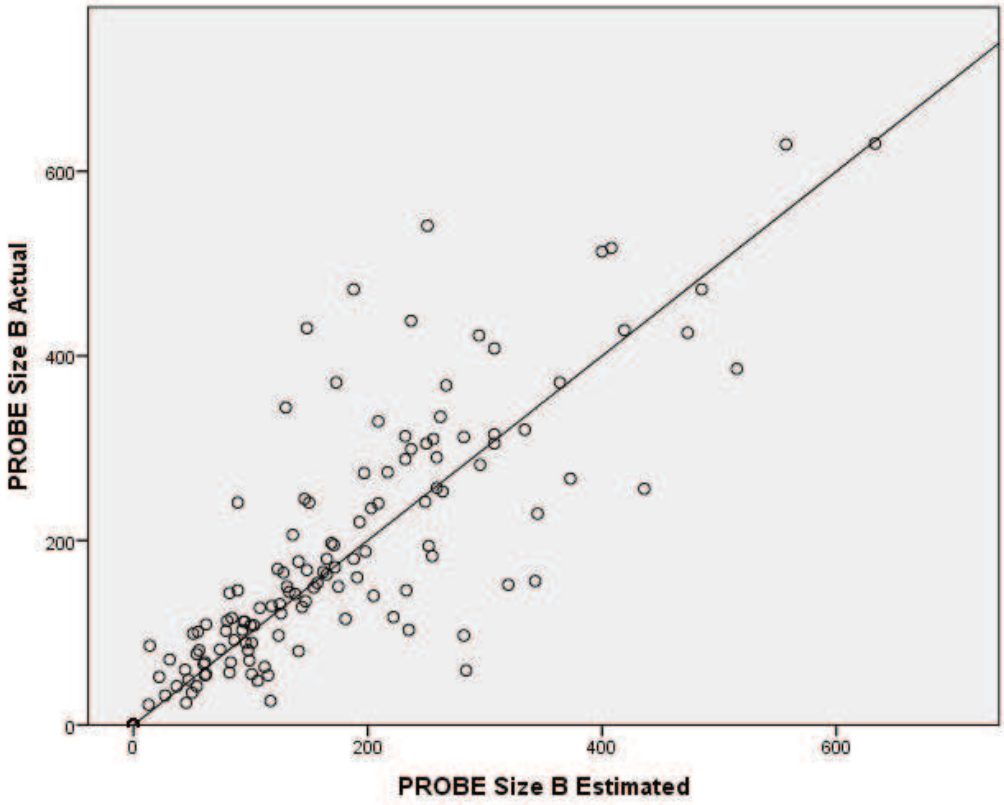


Figure 47: PROBE B - Estimated size vs Actual size scatter plot

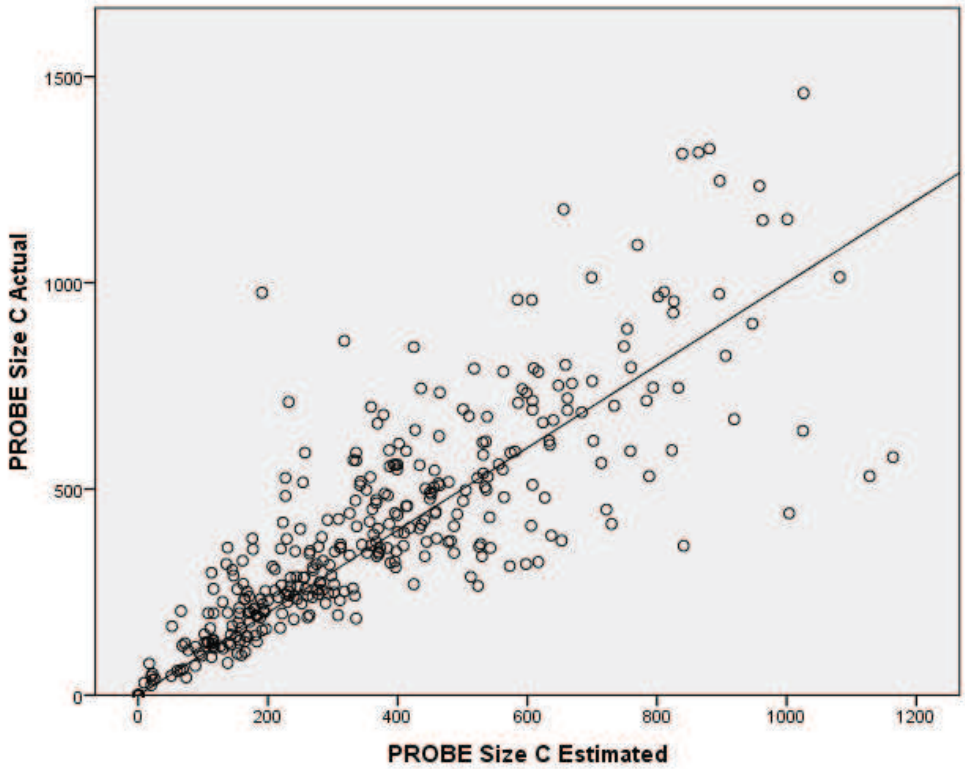


Figure 48: PROBE C - Estimated size vs Actual size scatter plot

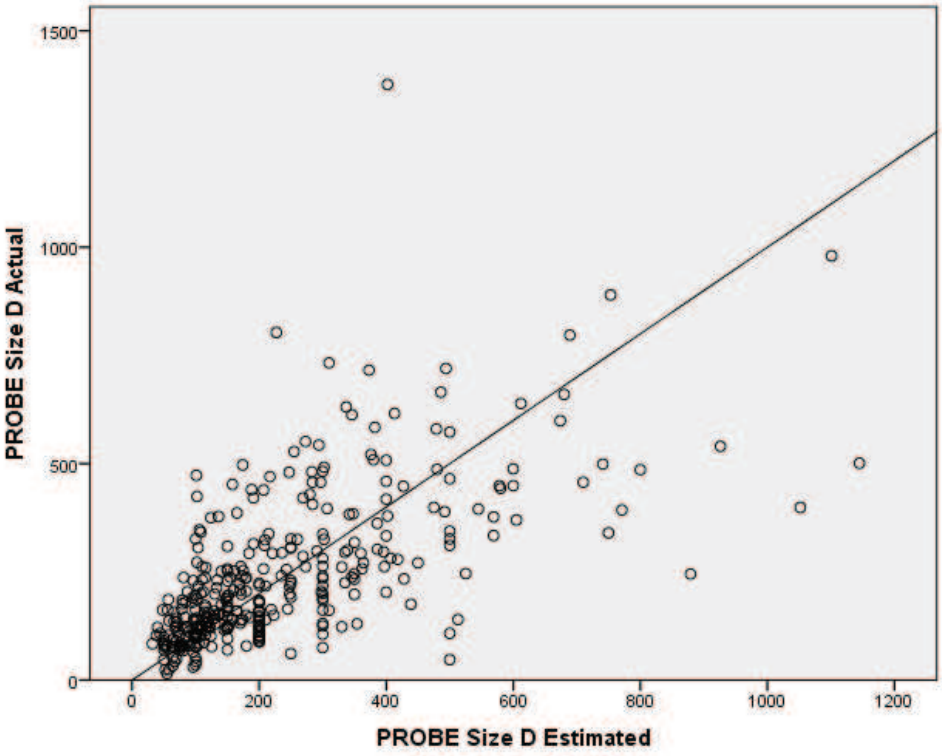


Figure 49: PROBE D - Estimated size vs Actual size scatter plot

In the scatter plots we can see that for PROBE C and D, the points are more diffuse than the points in the PROBE A and B scatter plots.

We know that this is just a descriptive analysis and that we cannot draw any conclusion about these observations. But it allows us to visualize the data in a different way, and we get a better idea of how subjects evolve during the application of the different estimation techniques during the courses.

7.2 Threats to validity and limitations

Estimating is a skill. The PROBE method consists on a lineal regression using engineer historical data that provides a framework for gathering estimating data. But if the engineer is not able to make a good conceptual design, with a good understanding of the product requirements and with a correct definition of the product elements that will produce the desired functions, it will not work. So, this could be a limitation to visualize the introduction and application of the PROBE estimation technique.

The PSP provides a proxy-based estimation method (introduced during PSP level 1) to help engineers decompose the program and estimate the size of each element, based on historical data. During the courses, the PSP tool used by the students, provide a C++ Class Size Ranges matrix that is used as baseline for all the students, despite the programming language they use. Maybe that size ranges are not appropriate to other programming languages and could be distorting the real engineer estimation. Also, class could not be optimal proxies for different programming languages, and database elements, screens, reports, etc., could be better proxies for other languages. Teachers should control these and motivate engineers to generate and use their own proxies.

While there will always be a subjective element to estimation no matter how much data it is based on, the PSP training strives to teach engineers how to make the best use of their own past experience. When the size estimation method is introduced at

the start of PSP level 1, in the first version of the PSP course (the one based on 10 program assignments) the engineers have data from the three previous assignments as a basis for estimating the fourth. But in the PSP Fund/Adv course, there is only one data point at PSP0. Maybe the introduction strategy is not the optimal, as it does not have a good basis to apply the PROBE method in PSP1.

Even having transformed the data, the normality assumption for ANOVA could not be fully satisfied. The distributions of ABS(SEA) tend to be positively skewed, with long tails extending to the right and a truncated range at zero. This type of non-normal distribution is to be expected given the source of the data. There can never be a negative absolute value, so the truncation at zero is expected. This positively skewed distribution is particularly expected when engineers (as a group) improve the size estimations of the programs as the linear regression helps them to estimate better during the course. This is the type of data where either a logarithmic or inverse transformation can be used to create a more nearly normal distribution [14]. Based on our examination of the effects of these two types of transformations on the distribution of residuals, the logarithmic transformation was used in the confirmatory analysis.

7.3 Conclusions

In this analysis we considered the work of 311 software engineers, who during PSP work, developed 7 or 8 programs, depending on the course version. Each subject took the complete PSP course, either PSP for Engineers I and II or PSP Fundamentals and Advance. We analyzed the data collected by each student to determine whether the introduction of a formal estimation technique for size improve the accuracy of engineers' size estimates or if such improvement is only a consequence of gaining experience in the problem domain.

Both courses appear to be successful in demonstrating effective use of the PROBE estimating method, and in showing improvement on the accuracy of engineers' size estimations.

The followings are candidate reasons for improvement that are not mutually exclusive:

- Simple repetition with feedback of actual results
- Introduction of a structured approach (PROBE D, C) including estimation by parts
- PROBE A, B corrections

Data do not clearly support our hypothesis, as we cannot analyze the PSP1 introduction on both courses. Perhaps our hypothesis is not satisfied, but we cannot be sure yet because the PSP level introductions in the studied courses format are not the optimal to analyze this behavior. Since PSP level changes so rapidly in the PSP Fundamentals and PSP I course, the program number and the PSP process level are tightly correlated in a way that makes separating the effects difficult. We also have very few points with PSP0 and PSP1 to analyze. That is why the second analysis approach, visualizing the data using the PROBE method, was necessary.

The results support some of our ideas and we are getting closer to see if PSP techniques associated with PSP level are the real causal explanation for the improvements, but this is not proved yet. Probably a larger study, examining the results by PROBE type and by PSP level together could be useful to reinforce these results.

These results cannot ensure that the observed improvements are exclusively due to mastering the process techniques introduced in the PSP. In Chapter 7, we propose some controlled experiments to permit the different PSP levels process changes to stabilize so that we could more directly examine improvements between programs with and without process changes. In this way, we can obtain more generalizable results.

8. General Conclusions of the Data Analysis

Previous studies of Personal Software Process have examined the effect of the PSP on the performance of software engineer and it was found that the improvements were statistically significant, and it was considered that the observed results could be generalized beyond the involved participants [7] [8] [9] [10]. Those studies only considered students of the first version of the PSP course which uses 10 program assignments and as there is a strong correlation between the program assignment and PSP level. Those studies may have some threats to external validity. In this work we tried to face the generalization threat and consider the latest two course versions to analyze and evaluate the effectiveness of the PSP and the impact of the domain experience (programming repetition).

To reach our goal we evaluated engineers' performance differences with respect to four dimensions: defect density in unit testing, yield, production rate and size estimation accuracy.

Regarding defect density in unit testing, we find out significant improvement with a mean reduction of a factor of 2.3. Our results suggest that improvements in defect density in unit testing are most plausible regarding mastering PSP techniques rather than programming repetition.

When analyzing yield, we find out significant improvement with a mean increase of a factor of 1.9. The results support that design and code reviews techniques are the main reasons of the improvements rather than the learning effect.

On the other hand, regarding production rate we find out a mean reduction of a factor of 0.7. Despite of what we expected, 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.

Finally, looking at the size estimation accuracy results, we find out significant improvement with a mean reduction of a factor of 2.6. For this hypothesis we were not able to discard the domain learning effect as the root causes of the improvements, as the estimation technique introduced in the PSP courses is based on historical data and needs repetition.

In conclusion, the analyses reported here substantiate that trends in personal performance observed during PSP application are significant, and that the observed improvements or deteriorations represent real change in individual performance, not a change in the average performance of the group.

Since PSP level changes so rapidly in the PSP Fund/Adv and PSP I/II course, the program number and the PSP process level are tightly correlated in a way that makes separating the effects difficult. This issue affected specially the size estimation accuracy study; however the results lead us to think that the process phases are probably one of the main reasons of the changes. In *Chapter 7*, we propose several controlled experiments, which will allow evaluating the programming repetition effects in depth.

Chapter 6

Related Work

Several empirical studies on the effects of the PSP training were published during the last 15 years. Those studies, which try to demonstrate the benefits of disciplined software development on the individual level, were developed among others by Hayes et al. in 1997 [7], Rombach et al. in 2007 [8], Paulk in 2010 [9] and by Nichols et al. in 2013 [10]. In this chapter the findings of these studies are presented, as well as their results are compared against the findings that were obtained in our work.

1. Hayes et al. – 1997

The Personal Software Process (PSP):
An Empirical Study of the Impact of PSP on Individual Engineers

Authors:	Hayes, Will; Over, James W.
Published:	Software Engineering Institute, Carnegie Mellon University Technical Report CMU/SEI-97-TR-001
Year:	1997

The objectives of this study were to test key assertions about the benefits of the PSP and to consider whether the observed results can be generalized beyond the study participants. Since the PSP was developed to improve individual performance, the study examined changes in individual performance as new practices were introduced.

The goal of the study was the following:

"Analyze the data collected at the PSP levels (0, 1, 2)
for the purpose of evaluating performance differences of engineers
with respect to size estimation accuracy / effort estimation accuracy /
yield / defect density / productivity[6]
from the viewpoint of a researcher
in the context of the PSP training of 298 engineers
that performed the PSP I/II original course."

The findings of this study are:

- The median individual improvement in size estimation accuracy is a factor of 2.5.

[6] Hayes et al. defined Productivity in the same way as we defined Production Rate.

- The median improvement in effort estimation accuracy is a factor of 1.75.
- The median reduction in total defect density is a factor of 1.5.
- 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
- The median improvement in yield was an increase of 50% in the number of defects removed before compile.
- Although significant fluctuation in productivity occurred (statistically), no real substantive gain or loss in productivity was observed.

This is an important work because it was the first to examine PSP results using recognized statistical techniques. It clearly states what PSP should accomplish, and it documents early results.

Comparing this study against our work, there are different experimentation aspects. They analyzed 298 engineers of the PSP I/II original course version while we analyzed 347 engineers of the other two course versions, the PSP I/II revised and the PSP Fundamentals and Advance.

Hayes et al. looked at the improvements during the course, without analyzing if the main reasons of the improvements are in fact the PSP introduced techniques and phases. We cannot compare our findings related to impact of the programming repetition against this study. But the observations about the effect size of the improvements can be compared against our results.

They detect that median reduction in defect density for the test phase is a factor of 2.5. For the same variable, we find out that the mean reduction is a factor of 2.3. Our results also support that product quality improvements are due to the PSP practices introduced rather than the domain effect learning.

Regarding the process yield, they detect that the median improvement was as an increase of 50% in the number of defects removed before compile. For the same variable we detect that the mean improvement is a factor of 1.9. This corresponds to a mean improvement in yield of a 55%. We computed this value by subtracting the yield for PSP level 1 from the yield for PSP level 2 for each engineer, then computing the mean of that distribution.

On the other hand, they find no real substantive gain or loss in production rate. For the same variable we find out a loss in the rate, with a mean reduction of a factor of 0.7.

Moreover, they realize that the median individual improvement in size estimation accuracy is a factor of 2.5. For the same variable we realize that the mean improvement is a factor of 2.6. In this case, we were not able to fully discard the programming repetition as the root causes of the improvements, as the PROBE method for size estimation is based on historical data and needs repetition.

As we can see, results regarding defect density in unit testing, yield and size estimation accuracy follow the same line in both studies. However, production rate results are quite different. Further research related to the last two course versions should be performed in order to know which could be the reasons of this production rate deterioration.

2. Rombach et al. – 2007

Teaching disciplined software development

Authors:	Rombach, Dieter; Münch, Jürgen; Ocampo, Alexis; Humphrey, Watts; Burton Dan
Published:	Journal of Systems and Software; Vol. 81, No. 5
Pages:	747-763
Year:	2008

This study is a replication and extension of the Hayes et al. study. It addresses the meaning of disciplined software development, its benefits, and the challenges of teaching it. It presents a quantitative study that demonstrates the benefits of disciplined software development on the individual level.

The goal of this study was the following:

"Analyze the data collected at the PSP levels (0, 1, 2, 3[7])
for the purpose of evaluating performance differences of engineers
with respect to size estimation accuracy / effort estimation accuracy /
defect estimation accuracy / yield / defect density / productivity
from the viewpoint of a researcher
in the context of the PSP training of 3090 engineers
that performed the PSP I/II original course".

The findings of this study are:

- Neither accept nor reject the hypothesis about size estimation accuracy improvement
- The effort estimation accuracy improves from an average of -25% to -10%
- At compile test phase, defect density improves from an average of 42 Defects/KLOC removed to 12 Defects/KLOC. Similar differences are shown at the compile and test phase together, and for defect density in the overall.
- Productivity[8] improves from an average of 30 LOC per hour to 39 LOC per hour.
- Yield improves from an average of 5% to 55%
- Neither accept nor reject the hypothesis about defect estimation accuracy improvement

[7]PSP level 3 is a level that was used in the first version of the PSP course. After that it has not been used anymore.

[8] Rombach et al. also defined Productivity in the same way as we defined Production Rate.

Comparing this study against our work, there are different experimentation aspects. We analyze a data set rather smaller than theirs: 347 engineers of the last two course version and they analyze 3090 engineers of the PSP I/II original course.

Referring the defect density, we find out that at test phase, it improves from an average of 15 Defect/KLOC to 2.5 Defect/KLOC. For the same variable, Rombach et al. find out that it improves from an average of 42 Defects/KLOC removed to 12 Defects/KLOC.

Regarding process yield we discover that it improves from an average of 7% to 62%. For the same variable, Rombach et al. discover that it improves from an average of 5% to 55%.

Moreover, we find out that production rate deteriorates from an average of 32 LOC per hour to 20 LOC per hour. For the same variable, Rombach et al. find out that it improves from 30 LOC per hour to 39 LOC per hour.

On the other hand, looking at the size estimation accuracy results, we find out significant improvement with a mean reduction of a factor of 2.6. However, Rombach et al. neither accept nor reject the hypothesis about size estimation accuracy improvement.

As we can see, yield results follow the same line in both studies. Although both studies show that defect density in unit testing improves, the beginning defect density averages are quite different on both studies (15 versus 42 defects per KLOC). In Rombach et al. work it is specified that the PSP courses can be a two weeks training for engineers from industry or a semester course in an academic environment. Even though they say that most of the classes were taught in industry to practicing software developers and less than 4% of the data is from students in a university setting, we should not discard this experimentation difference as one of the reasons for this distinct behavior of defect density average at the beginning of both studies. The students' previous experience and profiles can be affecting this variable, as well as the differences in the controlled environment between a massive undergraduate course and a SEI (or SEI partner) course for professional developers. On the other hand, regarding production rate, the studies show opposite results. Size estimation accuracy results are different as well, since they do not find significant improvements.

3. Paulk – 2010

The Impact of Process Discipline on Software Quality and Productivity

Author:	Paulk, Mark
Published:	ASQ Software Quality Professional; Vol. 12, No. 2
Pages:	15-19
Year:	2010

This work consists of a study of the impact of process discipline on personal software quality and productivity[9]. He considered a data set of 2435 programs developed by engineers who performed the PSP I/II original course. The article does not

[9] Paulk also defined Productivity in the same way as we defined Production Rate.

clarify how many engineers were involved in those developments. Only programs written in C were considered, in order to remove a possible confounding factor.

The impacts found on quality are:

- Defect density (defects/KLOC) in testing: Quality improved by 79% and variability decreased by 81%
- The number of defects generally decrease across the PSP assignments despite the observation that the number of lines of code is increasing at the same time.
- Programmer ability also affects software quality. The top-quarter students improved their software quality by a factor more than two, and the bottom-quarter students improved theirs by a factor more than four.

The impacts found on productivity are:

- Productivity (LOC/hour) increases but it is not practical significant.
- Programmer ability also affects productivity.

Comparing this study against our work, there are different experimentation aspects. He analyzed 2435 programs developed by engineers that performed the PSP I/II original version. The author does not clarify how many engineers are involved in those programs. We analyzed 347 engineers of the other two course versions, the PSP I/II revised and the PSP Fundamentals and Advance.

Regarding defect density in unit testing, we find out that quality improved by 84% and variability decreased by 80%. For the same variable he finds out that quality improved by 79% and variability decreased by 81%.

On the other hand, regarding production rate we discover a mean reduction of a factor of 0.7. For the same variable he discovers not practical significant changes.

As we can see, defect density in unit testing results follow the same line in both studies, while the product rate results are quite different.

4. Nichols et al. – 2013

The Personal Software Process (PSP) Revisited: Empirical Benefit Analysis

Author:	Nichols, William; Küpper Steffen; Andelfinger Urs
Published:	Technical Report - draft version provided by the authors
Year:	2013

This technical report aims at deepening the understanding of original hypotheses of Hayes et al. and Rombach et al. They analyzed the data of 3111 engineers that performed the PSP I/II original course. The data set is a superset of the data used by Hayes et al. and by Rombach et al. They are considering graduate and undergraduate students, as well as subjects that performed a two weeks training for engineers from industry or a semester course in an academic environment.

The main findings in this study are:

- Neither accept nor reject the hypothesis about size estimation accuracy improvement
- Neither accept nor reject the hypothesis about size effort accuracy improvement
- Regarding estimations, they find out different behavior between engineers that overestimate and the engineers that underestimate.
- The defects injected per thousand lines of code does not decreases
- The number of defects removed per thousand lines of code decrease
- With the introduction of design and code reviews, the defect densities of programs entering the compile and test phases decrease significantly
- Engineers do not lose on productivity[10]
- The introduction of design review and code review has a significant impact on the value of engineers' yield.
- Reject the hypothesis about defect estimation accuracy improvement

Comparing this study against our work, there are different experimentation aspects. We analyze a data set rather smaller than theirs: 347 engineers of the last two course version versus 3111 engineers of the PSP I/II original course.

Both studies find out improvements in size estimation accuracy, defect density in unit testing and yield, while the impact on production rate is quite different on both studies.

5. Comparison summary

Four related works were presented in the previous sections. These research works study the effects of the PSP training and try to demonstrate the benefits of disciplined software development on the individual level.

One of the main differences between these studies and our work is that all of them analyzed data from students who performed the PSP I/II original course while we analyzed data from the PSP I/II revised and PSP Fundamentals and Advance courses. The amount of subjects considered on each study is also a relevant difference, which impacts on the generalization of the observations.

The data quality assessment and cleaning is another difference between these studies. Hayes et al. only considered for their analysis the students that reported complete data (records available for the 10 assignments involved in the course) for each variable under study. On the other hand, Rombach et al. reported that they assessed the data quality and that they only used data from those students whose data for all 10 exercises were correct, complete, and consistent. Paulk does not include any specification about the quality of the analyzed data set. Besides, Nichols et al. reported that they considered those engineers who at least provided complete data for all the assignments, also reporting that further data cut-offs were made based in the PSP guidelines, in order to base their analysis in complete, correct and consistent data. In our

[10] Nichols et al. also defined Productivity in the same way as we defined Production Rate.

work, we performed the data quality assessment and cleaning based on the Data Quality theory. We performed that by defining and measuring 10 data quality problems (associated with the accuracy, consistency, completeness and uniqueness data quality dimensions) through 91 specific metrics.

Another important difference is that they looked at the improvements based on how the engineers' performance evolved during the course. That approach implies a threat to external validity, which is the confounding of process phases and techniques insertions with the gaining of domain experience as related programs are developed. With our approach, we faced that threat.

Table 21 presents a comparison table of the findings from all the studies discussed in the previous sections.

Hypothesis	Hayes et al. 1997	Rombach et al. 2007	Paulk 2010	Nichols et al. 2013	Our study 2013
Size Estimation Accuracy	Improved a factor of 2.5	Neither accept nor reject the hypothesis	Not analyzed	Neither accept nor reject the hypothesis	Improved a factor of 2.6
Effort Estimation Accuracy	Improved a factor of 1.75	Improved from -25% to -10%	Not analyzed	Neither accept nor reject the hypothesis	Not analyzed
Total Defect Density	Reduced a factor of 1.5	Reduced from 103 to 50 defects per KLOC	Not analyzed	Reduced factor not specified	Not analyzed
Defect Density in Compile phase	Reduced a factor of 3.7	Reduced from 57 to 12 defects per KLOC	Not analyzed	Reduced factor not specified	Not analyzed
Defect Density in Unit Testing	Reduced a factor of 2.5	Reduced from 42 to 12 defects per KLOC	Improved by 79% and variability decreased by 81%	Reduced factor not specified	Reduced a factor of 2.3. Reduced from 15 to 2.5 defects per KLOC. Improved by 84% and variability decrease by 80%
Process Yield	Increase of 50%	Improves from 5% to 50%	Not analyzed	Improved factor not specified	Increase of 55%. Improves from 7% to 62%
Production Rate	No gain or loss	Improves from 30 to 39 LOC per hour	No gain or loss	No gain or loss	Reduced a factor of 0.7. Deteriorates from 32 to 20 LOC per hour
Defect Estimation Accuracy	Not analyzed	Neither accept nor reject the hypothesis	Not analyzed	Rejected the hypothesis	Not analyzed

Table 21: Findings comparison[11]

[11] The table has different units for the effects because it is not possible to calculate a standardized effect size from the available data of the articles. To calculate effect size is necessary to have the mean of the groups, the standard deviation and the size of the samples.

All studies present an improvement in defect density in unit testing. Although not all the works analyzed the evolution of the total defect density and the defect density in the compile phase, we can see a reduction trend. Additionally, improvements are observed regarding process yield in all the studies that analyzed that variable. These observations support the PSP course benefits regarding product quality, regardless of whether these are achieved by the process itself or by the domain learning effects.

However, regarding size, effort and defect estimations, some studies report improvements while others show not significant improvements.

Production rate seems to be the most variable of all the analyzed hypotheses, as some studies find improvements, others find no real gain or loss, and we find a significant loss of productivity.

Besides showing an improvement in defect density in unit testing, process yield and size estimation accuracy, our work includes the elimination of the threat to validity related to the learning by programming repetition in two of those variables: process yield and defect density in unit testing. This is one of the thesis' contributions since it had not been previously studied.

Chapter 7

Conclusions and Future Work

This chapter includes three sections: conclusion, contributions of the research and the future work.

1. Conclusions

Almost every new product or system that we use in our daily life 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 needs improved software quality, better cost and schedule management as well as reduced software development cycle time [40].

The Team Software Process (TSP) is a software development process for teams that satisfies these needs and which uses the Personal Software Process (PSP) for each team member [4] [5]. The PSP is a defined and measured software process designed to be used by an individual software engineer to addresses the software businesses needs by improving the technical practices and individual abilities of software engineers, and by providing a quantitative basis for managing the development process [41].

Given that the TSP is a process successfully used and it is qualified as the best software development process for medium and large scale projects [6], it is important to know whether the processes and the applied techniques of the PSP lead to develop high quality products. Therefore, the general goal of this thesis is to know if the different techniques and phases of the PSP (and therefore, the PSP itself) produce positive changes in the aforementioned aspects of the 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. An earlier version of the course has several published studies demonstrating improvement in developer performance with process insertion [7] [8] [9] [10] [11] [12], 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 thesis 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 Hayes [7] and Rombach [8], we decided to evaluate the effects of the last two PSP course versions through four 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 thesis as:

- 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 ***defect density in unit testing / yield / production rate / size estimation accuracy*** from the viewpoint of a researcher in the context of the PSP training of engineers in "PSP for Engineers I/II revised" course and the training of engineers in "PSP Fundamentals and Advance" 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.

The quality of the PSP collected data can have a relevant impact in the results. We considered that it was important to find a way to ensure that the statistical analyses were based on quality data. That is why, based on the Data Quality theory [13], we thoroughly identified and defined possible quality problems that the data under study might contain. We implemented the algorithms required for measuring, cleaning and collecting the metadata and we executed those algorithms afterwards.

Ten quality problems were identified, and a total of 91 metrics were defined in order to measure these problems applied to objects of the database. After executing all measuring, we observed that a 1.34% of the total of the measured objects has some error or possible error. Our result is different from the findings of a prior study on the quality of the PSP data, in which it is reported a 4.8% of errors within data [33]. This difference is probably due to the previous study which was based on a version of the course for which there was neither computer nor tool support for the process and the calculations involved. After the measurement process, we executed a data cleaning procedure, obtaining a data set with the necessary quality for our statistical analysis for each hypothesis.

The quality of the data collected during a software development project is relevant due to the problems that poor data quality can cause: deviated estimations, wrong predictions, bad project monitoring, among others. Our analysis on the quality of the data collected during a disciplined development process (as the PSP) shows few quality defects (about 1%). However, the cleaning of these few low quality data is necessary when performing hypothesis tests.

After cleaning the data, we analyzed whether performance improvements are due to the programming repetition or due to the introduction of phases and techniques. To carry this out we defined and applied an indirect statistical method of analysis that consists of three steps in which the relationships between program number, PSP level, course version and engineers' performance are examined by applying analysis of variance statistical methods. We followed this approach for each of the hypothesis.

Regarding defect density in unit testing, we found significant improvement with a mean reduction of a factor of 2.3. This result is consistent with Hayes and Rombach findings. Our results not only show the effect size, but suggest that improvements in

defect density in unit testing are most plausible regarding mastering PSP techniques rather than programming repetition.

Our results show significant improvement in the process yield with a mean increase of a factor of 1.9. This result is also consistent with Hayes and Rombach findings. Our results also support that design and code reviews techniques are the main reason of the improvements rather than the learning effect.

Regarding production rate we found a mean reduction of a factor of 0.7. This result differs from the Hayes findings, which did not find gain or loss in the production rate. Our result also differs from Rombach findings, who found an improvement of the production rate. 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.

Looking at the size estimation accuracy results, we found significant improvement with a mean reduction of a factor of 2.6. This result is consistent with Hayes findings, but differs from Rombach findings, as he neither accepts nor rejects this hypothesis. For this particular dimension we were not able to discard the domain learning effect as the root causes of the improvements, as the estimation technique introduced in the PSP courses (the PROBE method) is based on historical data and needs repetition.

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 the followed approach, we are able to suggest that the PSP is the root cause of the improvements rather than the domain learning effect in two of the four studied hypothesis: defect density in unit testing and process yield. Since PSP level changes so rapidly in the PSP Fundamentals and Advance 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 reject 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 of the changes, so further research and experimentation is 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 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.

Perform controlled experiments related to the application of a software development process is not a trivial task. The PSP, since it is a highly instrumented process, allows having enough data for Software Engineering experiments. In this work we were able to see that from a simple recollection of direct measures like defects, times per phases and size, effort and quality estimations, arise many software metrics that are really important and which allow to effectively evaluate the process execution.

Given that Software Engineering cares about quality, costs and schedule [42], it is necessary to perform more empirical studies which show how techniques, methods and processes help to improve each of those issues [43]. Unfortunately, the available empirical evidence in this area is still not quite enough [44]. Not many software development processes have been as much studied as the PSP and the TSP.

2. Contributions

The main contributions of this work are the following:

- We give insight into the engineers' performance changes in the last two PSP course versions with respect to four dimensions: defect density in unit testing, yield, production rate and size estimation accuracy. The behavior on the engineers' performance on these two course versions, PSP Fundamentals and Advance and PSP for engineers I/II revised, was not studied before.
- We address the threat to external validity related to the learning by programming repetition. Results suggest that improvement effects are most plausible regarding mastering PSP techniques rather than general domain knowledge in two of the studied dimensions: process yield and defect density in unit testing.
- We study the quality of the data collected during the execution of the last two PSP course formats. We did not find in the literature other data quality analysis like this one, at least not using a formal approach based on the Data Quality discipline. Our approach can be used, with adaptations, for other software development processes.

3. Future Work

As future work we propose several controlled experiments to continue facing the domain learning threat in order to obtain more generalizable results.

One approach consists of a controlled experiment where the students must perform, for example, the same eight assignments defined for the PSP I/II revised course. But, in this experiment, the students perform the first program with PSP0 and all the other assignments with PSP0.1. That is, in this experiment proposal, the student repeats the same baseline process, without the introduction of new techniques or new phases. Here, the researcher will be able to see precisely how the domain learning effect is affecting any dimension of the engineers' performance, blocking the effect of the PSP level. If there are improvements during the experiment, there is no doubt that they are due to programming repetition. Part of this work has already begun and an extended abstract has been recently sent (on April 2nd, 2013) to present an article in the TSP Symposium 2013. This extended abstract can be found in Appendix 5.

A second approach could be a controlled experiment with 3 groups of students: X, Y and Z. The students of the group X are introduced to PSP0.1, the students of the group Y to PSP1.1, and the students of the group Z to PSP2.1. Then, each student of each group performs, for example, 10 assignments applying the higher PSP level that they were introduced to. In this way, each PSP level will be stable at the end of the experimentation process for each group of students, and the effects of programming repetition and of the techniques applied on each PSP level could be examined.

A third approach would be an extended PSP course with at least three exercises at each PSP level. We judge that this would permit the process changes to stabilize allowing us to examine improvements between programs with and without process change more directly.

Any of these controlled experiments proposals are useful not only to face the programming repetition threat for the variables studied in this thesis work, but for any variable defined to analyze the changes of an aspect of the engineers' performance in the PSP. Although more approaches are possible, we just present some of them.

Regarding the production rate results that we found, which were quite different from the previous studies related to the PSP I/II original course, we propose to perform further research related to the last two course versions in order to know which could be the reasons of this production rate deterioration. Related with this issue, we also propose as future work a comparison between the techniques introduction in the different course versions, in order to determine the optimal way to introduce the process phases and techniques to teach the PSP.

We think that it is important to analyze the PSP I/II original course in order to evaluate the learning effect on that course. That is, we propose to do a replication of our study with the old course version data as a future work. There are three main reasons for this replication: the programming repetition effect has never been studied on that PSP course version, there is more available data (more than 3000 engineers, that is, more than ten times the data that we analyzed), and each level of PSP is used in at least three assignments. The last two reasons can help generalize the results of our study, as well as gaining more knowledge about the size estimation accuracy.

We consider necessary to analyze the other variables related to engineers' performance that have been studied previously (for the PSP I/II original course) in the related works and which were not studied in this opportunity: effort estimation accuracy, defect estimation accuracy, total defect density and defect density per phase. This would allow reaching a fully complete analysis of the performance impact of the PSP's phases and techniques.

In regards to the quality of the data collected during the PSP, it is of interest to understand effect of data cleaning upon the hypotheses. An empirical approach would be analyze the same hypotheses but considering all the students' data without the data quality cleaning and cut-offs applied. This result could be compared to the analysis with cuts applied. A substantial change would strongly suggest care should be taken when using that data to make decisions.

It seems reasonable to analyze the particular data quality problems that we found in order to improve the PSP support tool (which is used by the students during the courses to recollect the process data) and prevent data errors. An example of an improvement could be that the tool automatically notify of any quality problem before saving the data. The grading checklist used by the PSP instructors could also be improved based on the data quality problems found, in order to improve the teaching of the PSP.

Something interesting would be to conduct this kind of studies for the data collected during the use of the Team Software Process. The data to be used would be data generated by the use of this process in the industry. There, the impact of the process phases and techniques on the engineers' performance can be evaluated in industrial scale projects, during the whole software development process.

Appendix 1

Software Quality Models and Processes

This appendix presents software quality models and processes created by the SEI, as the CMMI and the TSP. Also a complete description of the PSP Quality Process and Product Measures is presented

1. CMM and CMMI

This section presents a summary of the main characteristics of the Capability Maturity Model (CMM), created by the Software Engineering Institute (SEI) of Carnegie Mellon University. This description of CMM is based on the SEI technical reports *Key Practices of the Capability Maturity Model* [45] and *Capability Maturity Model for Software* [46].

CMM is created to guide software organizations in the process improvement strategy's selection. This is done through the determination of the maturity of the development process and the identification of the more critical elements.

The following definitions are useful to understand CMM and CMMI:

Software process capability describes the range of expected results that can be achieved by following a software process. The software process capability of an organization provides one means of predicting the most likely outcomes to be expected from the next software project the organization undertakes.

Software process performance represents the actual results achieved by following a software process. It focuses on the achieved results, while software process capability focuses on expected results.

Software process maturity is the extent to which a specific process is explicitly defined, managed, measured, controlled, and effective.

The hypothesis that is handled both by CMM and CMMI is that a mature process is a process with high capability.

CMM has five maturity levels. Each level is composed by Key Process Areas. For an organization to reach a certain maturity level, it must meet the Key Areas of that level and all the previous levels. In the Figure 50 are shown all levels, the software process types associated to each level and the key areas of each level.

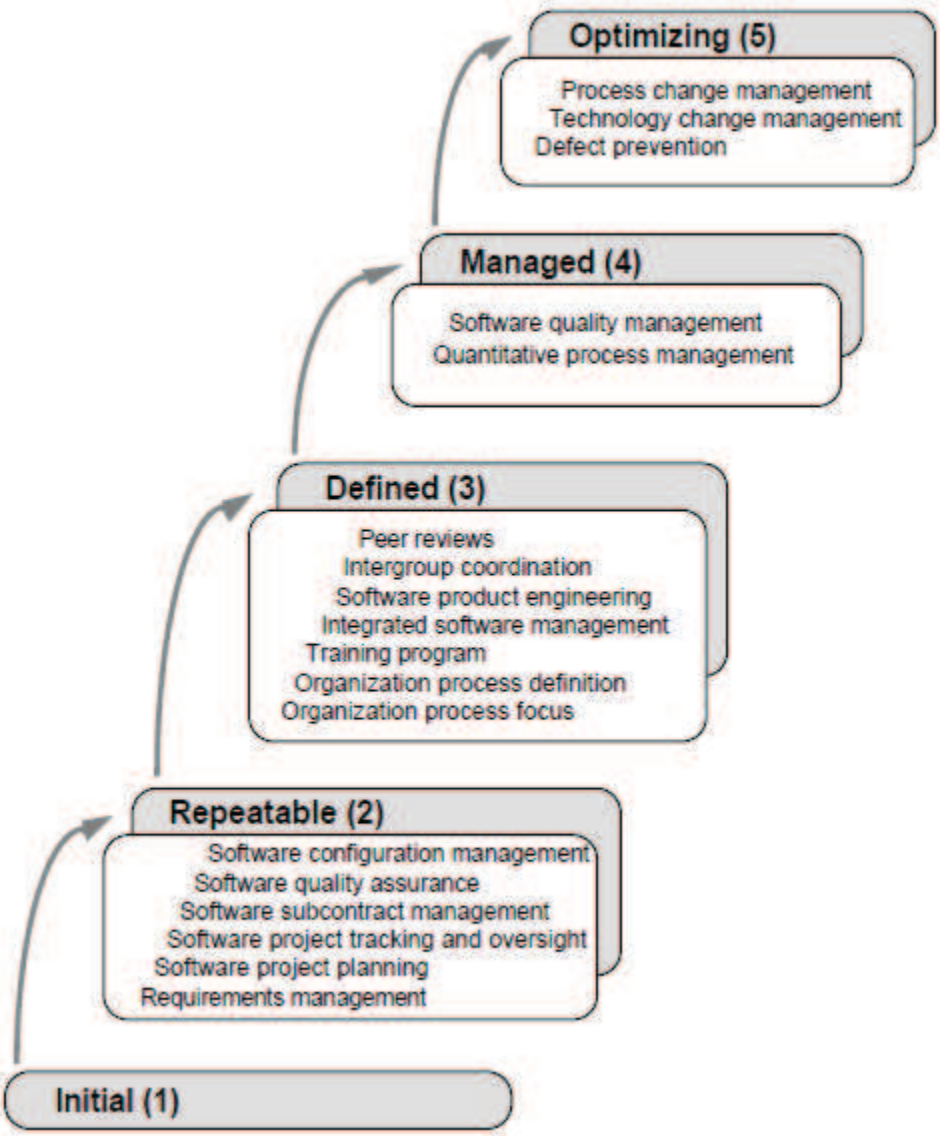


Figure 50: CMM Levels and Key Areas

As our focus is software quality, only the key areas related to quality are explained.

At level 2, the purpose of Software Quality Assurance is to provide management with appropriate visibility into the process being used by the software project and of the products being built. Software Quality Assurance is an integral part of most software engineering and management processes.

The goals of the Software Quality Assurance are:

- To plan software quality assurance activities.
- To objectively verify the adherence of software products and activities to the applicable standards, procedures, and requirements.
- To inform affected groups and individuals about software quality assurance activities and results.
- To inform about noncompliance issues that cannot be resolved within the software project, and to address them to senior management for it to intervene.

At level 4, the purpose of Software Quality Management is to develop a quantitative understanding of the quality of the project's software products and achieve specific quality goals.

The goals of the Software Quality Management are:

- To plan the project's software quality management activities.
- To define measurable goals for software product quality and their priorities.
- To manage and quantify the actual progress toward achieving the quality goals for the software products.

CMMI (Capability Maturity Model Integration) is the successor of the CMM. The CMM was developed from 1987 until 1997. In 2002, CMMI Version 1.1 was released, Version 1.2 followed in August 2006, and Version 1.3 in November 2010.

Although Capability Maturity Model Integration (CMMI) has differences with CMM, these have no influence in the presented work. Therefore, neither a presentation nor a discussion on CMMI will be done. It has been considered that it would not add to this work, and that it would repeat many of what has already been stated about CMM.

2. TSP

This section presents a summary of the main characteristics of the Team Software Process (TSP), developed by Watts Humphrey in 1996 at the SEI. This description of the TSP is based on the SEI technical report *The Team Software Process (TSP)* [23].

As it was stated before, the TSP provides a disciplined context for engineering work. The principal motivator for the development of the TSP was the conviction that engineering teams can do extraordinary work, but only if they are properly formed, suitably trained, staffed with skilled members, and effectively led. The objective of the TSP is to build and guide such teams.

Early experience with the TSP shows that its use improves the quality and productivity of engineering teams while helping them to meet cost and schedule commitments more accurately.

The teambuilding principles used in the TSP to establish the conditions that characterize effective teams are as follows:

- The team members establish common goals and defined roles.
- The team develops an agreed-upon strategy.
- The team members define a common process for their work.
- All team members participate in producing the plan, and each member knows his or her personal role within such plan.
- The team negotiates the plan with management.
- Management reviews and accepts the negotiated plan.
- The team members do the job in the way that they have planned to do it.
- The team members communicate freely and often.

- The team forms a cohesive group: the members cooperate, and they are all committed to meeting the goal.
- The engineers know their status, get feedback on their work, and have leadership that sustains their motivation.

The principal elements of the TSP process are shown in Figure 51. Before the members can participate on a TSP team, they must know how to do disciplined work. Training in the PSP is required to provide engineers with the knowledge and skills to use the TSP.

While there are many ways to build teams, they all require that the individuals work together to accomplish some demanding task. In the TSP, this demanding team-building task is a four-day planning process that is called the team launch. In a launch, all the team members develop the strategy, process, and plan for doing their project. After completing the launch, the team follows its own defined process to do the job.

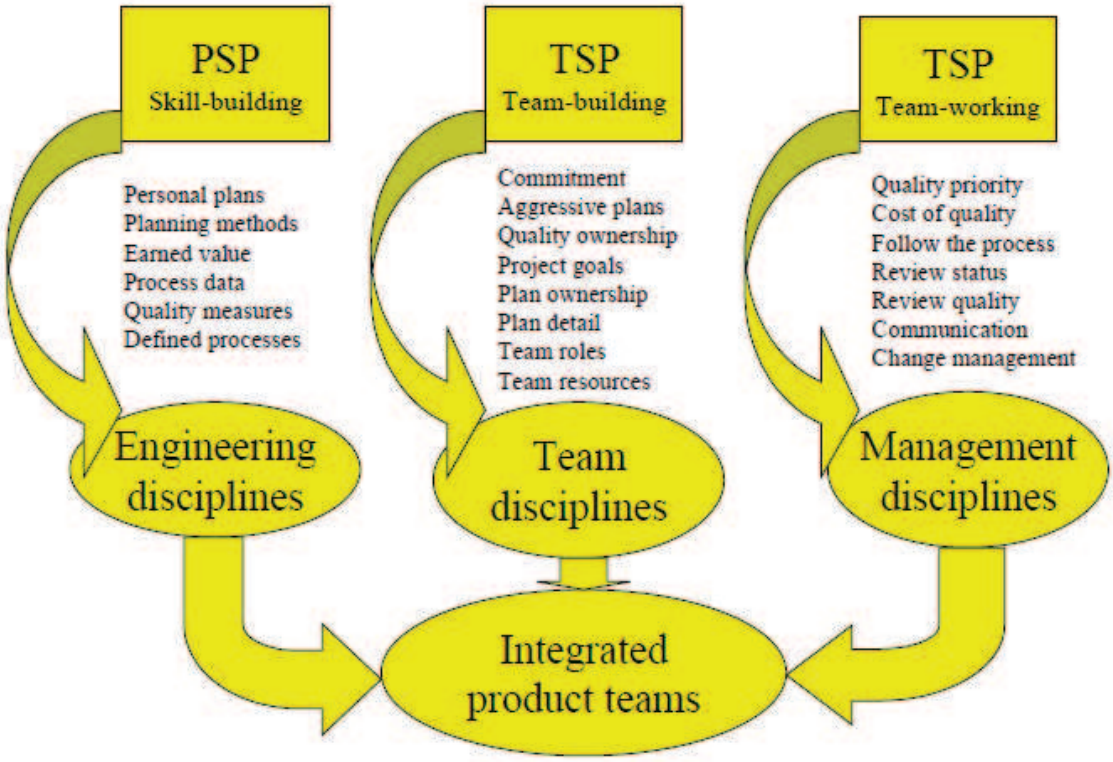


Figure 51: TSP Team-Building

In the TSP, the principal quality emphasis is on defect management. To manage quality, teams must establish quality measures, set quality goals, establish plans to meet these goals, measure progress against the plans, and take remedial action when the goals are not achieved. The TSP shows teams how to do this. The elements of TSP quality management are making a quality plan, identifying quality problems, and finding and preventing quality problems.

TSP introduces a series of quality measures that help to identify quality problems. These measures are:

- Percent defect free (PDF)
- Defect-removal profile
- Quality profile

- Process quality index (PQI)

3. PSP Quality Process and Product Measures

With size, time, and defect data, there are many ways to measure, evaluate, and manage the quality of a program. The PSP provides a set of quality measures that helps engineers examine the quality of their process and programs from several perspectives. While no single measure can adequately indicate the overall quality of a process or a program, the aggregate picture provided by the full set of PSP measures is generally a reliable quality indicator. The principal PSP quality measures are:

- Defect density
- Review rate
- Development time ratios
- Defect ratios
- Yield
- Defect removal leverage
- Appraisal to failure ratio (A/FR)

Each of these measures is described in the following paragraphs.

Defect Density. Defect density refers to the defects per Added and Modified KLOC found in a program. Thus, if a 150 LOC program had 18 defects, the defect density would be 1000*18/150 = 120 defects/KLOC

Defect density is measured for the entire development process and for specific process phases. Since testing only removes a fraction of the defects in a product, when there are more defects that enter a test phase, there will be more remaining after the test phase is completed. Therefore, the number of defects found in a test phase is a good indicator of the number that remains in the product after that test phase is completed.

Review Rate. In the PSP design and code reviews, engineers personally review their programs. The PSP data show that when engineers review designs or code faster than about 150 to 200 added and modified LOC per hour, they miss many defects. With the PSP, engineers gather data on their reviews and determine how fast they should personally review programs to find all or most of the defects.

Development Time Ratios. Development time ratios refer to the ratio of the time spent by an engineer in any two development phases. In the PSP, the three development time ratios used in process evaluation are design time to coding time, design review time to design time, and code review time to coding time.

Defect Ratios. The PSP defect ratios compare the defects found in one phase to those found in another. The principal defect ratios are defects found in code review divided by defects found in compile, and defects found in design review divided by defects found in unit test. A reasonable rule of thumb is that engineers should find at least twice as many defects when reviewing the code as they find in compiling it. The number of defects found while compiling is an objective measure of code quality. When engineers find more than twice as many defects in the code review as in compiling, it generally means that they have done a competent code review or that they did not record all the compile defects. The PSP data also suggest that the design review to unit

test defect ratio should be two or greater. If engineers find twice as many defects during design review as in unit test, they have probably done acceptable design reviews.

Yield. In the PSP, yield is measured in two ways. *Phase yield* measures the percentage of the total defects that are found and removed in a phase. For example, if a program entered unit test with 20 defects and unit testing found 9, the unit test phase yield would be 45%. Similarly, if a program entered code review with 50 defects and the review found 28, the code review phase yield would be 56%. *Process yield* refers to the percentage of defects removed before the first compile and unit test. Since the PSP objective is to produce high quality programs, practiced reviewers can find 70% or more of the defects before compiling or testing.

Defect Removal Leverage (DRL). Defect removal leverage measures the relative effectiveness of two defect removal phases. For instance, if the defect removal leverage for design reviews over unit test is 3.06/1.71 = 1.79, it means that the engineer will be 1.79 times more effective at finding defects in design reviews as in unit testing. The DRL measure helps engineers design the most effective defect removal plan.

A/FR. The appraisal to failure ratio (A/FR) measures the quality of the engineering process, using cost-of-quality parameters [16]. The A stands for the appraisal quality cost, or the percentage of development time spent in quality appraisal activities. In PSP, the appraisal cost is the time spent in design and code reviews, including the time spent repairing the defects found in those reviews.

The F in A/FR stands for the failure quality cost, which is the time spent in failure recovery and repair. The failure cost is the time spent in compile and unit test, including the time spent finding, fixing, recompiling, and retesting the defects found in compiling and testing.

The A/FR measure provides a useful way to assess quality, both for individual programs and to compare the quality of the development processes used for several programs. It also indicates the degree to which the engineer attempted to find and fix defects early in the development process. In the PSP course, engineers are told to plan for A/FR values of 2.0 or higher. This ensures that they plan adequate time for design and code review

Appendix 2

Concepts of Empirical Software Engineering

This appendix consists of a technical report about Concepts of Empirical Software Engineering.

PEDECIBA Informática
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Reporte Técnico RT 13-01

Conceptos de ingeniería de Software empírica. Versión 2.0

Cecilia Apa, Stephanie de León,

Silvana. Moreno,Rosana Robaina,

Diego Vallespir

2013

Conceptos de ingeniería de software empírica v. 2.0
C. Apa, S.de Leóin, S. Moreno, R. Robaina, D. Vallespir
ISSN 0797-6410
Reporte Técnico RT 13-01
PEDECIBA
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay, 2013

Conceptos de Ingeniería de Software Empírica

Versión 2.0

Cecilia Apa
Stephanie de León
Silvana Moreno
Rosana Robaina
Diego Vallespir

Reporte Técnico InCo/Pedeciba-2013 TR:13-01
Febrero, 2013

Resumen

En este artículo se presentan conceptos teóricos básicos de la Ingeniería de Software Empírica, así como también técnicas y herramientas de experimentación y de estudio de casos. La experimentación es un método que se usa para corresponder ideas o teorías con la realidad, proporcionando evidencia que soporte las hipótesis o suposiciones que se creen válidas. La experimentación en la Ingeniería de Software no ha alcanzado aún la madurez que tiene la experimentación en otras disciplinas (por ejemplo, biología, química, sociología). Sin embargo, en los últimos años ésta área en la Ingeniería de Software ha cobrado gran importancia y su actividad ha sido creciente.

Un experimento controlado intenta validar ciertas hipótesis mediante un *ambiente* creado para tal fin. Para esto se controlan ciertas variables (independientes) y se observa el resultado que arrojan ciertas otras variables (dependientes). Luego, estadísticamente, se rechazan o aceptan las hipótesis propuestas.

Un estudio de casos es un método de aprendizaje acerca de un fenómeno; se basa en el entendimiento de dicho fenómeno, el cual se obtiene a través de la descripción y análisis del mismo dentro de su contexto. Los estudios de casos no generan resultados sobre las relaciones causales como lo hacen los experimentos controlados, sino que proporcionan una comprensión más profunda de los fenómenos bajo estudio.

Aquí se presentan dos procesos, uno para realizar experimentos controlados y otro para conducir estudios de casos. Estos procesos son utilizados por Grupo de Ingeniería de Software de esta Facultad para realizar sus estudios empíricos.

Control de Versiones del Documento

Versión	# Reporte	Fecha	Cambios
Versión 1	RT 10-02, 2010	Febrero, 2010	Versión inicial
Versión 2	RT 13-01, 2013	Febrero 2013	▪ Se agrega el método de investigación Estudio de casos. ▪ Se agregan los trabajos de investigación del Grupo de Ingeniería de Software, UdelaR ▪ Se ajustan la introducción, resumen y conclusiones.

Índice general

Índice de figuras

Índice de cuadros

1. Introducción

Este reporte tiene como objetivo presentar los fundamentos de la Ingeniería de Software Empírica (ISE), se presentan los conceptos de experimentos formales (controlados) y estudios de casos. Se pretende que este documento sea utilizado por Proyectos de Grado de la carrera Ingeniería en Computación, Maestrías, Doctorados, etc. que se encuentran realizando trabajos de ISE con el Grupo de Ingeniería de Software (GrIS). De esta manera los estudiantes de grado y posgrado pueden usar este documento como punto de partida para comprender la ISE. Además, pueden incluir este documento como parte de su informe de proyecto o su Tesis evitando tener un enfoque distinto de la ISE en cada Proyecto de Grado y/o Tesis.

Este reporte se basa casi completamente en los libros *Experimentation in Software Engineering: An Introduction* (18), *Basics of Software Engineering Experimentation* (4), *Software Metrics - A Rigorous And Practical Approach* (3) y *Case Study Research in Software Engineering* (7).

En la sección 2 se presentan los distintos enfoques y estrategias de la ISE. Una de estas estrategias es la de experimentos formales, estos se describen en la sección 3. En la sección 4 se describe un proceso para llevar adelante un experimento formal. Otra estrategia es la de Estudios de Casos, que se describe en la sección 5. Por último en la sección 6 se describen los trabajos de investigación que ha llevado adelante el GrIS en los ultimos años.

2. Enfoques y Estrategias

La ISE utiliza métodos y técnicas experimentales como instrumentos para la investigación. La evidencia empírica proporciona un soporte para la evaluación y validación de atributos (p.e. costo, eficiencia, calidad) en varios tipos de elementos de Ingeniería de Software (p.e. productos, procesos, técnicas, etc.). Se basa en la experimentación como método para corresponder ideas o teorías con la realidad, la cual refiere a mostrar con hechos las especulaciones, suposiciones y creencias sobre la construcción de software.

Se pueden distinguir dos enfoques diferentes al realizar una investigación empírica: el enfoque cualitativo y el cuantitativo. El enfoque **cualitativo** se basa en estudiar la naturaleza del objeto y en interpretar un fenómeno a partir de la concepción que las personas tienen del mismo. Los datos que se obtienen de estas investigaciones están principalmente compuestos por texto, gráficas e imágenes, entre otros.

El enfoque **cuantitativo** se corresponde con encontrar una relación numérica entre dos o más grupos. Se basa en cuantificar una relación o comparar variables o alternativas bajo estudio. Los datos que se obtienen en este tipo de estudios son siempre valores numéricos, lo que permite realizar comparaciones y análisis estadístico.

Es posible utilizar los enfoques cualitativos y cuantitativos para investigar el mismo tema, pero cada enfoque responde a diferentes interro-

gantes. Se puede considerar que estos enfoques son complementarios más que competitivos, ya que el enfoque cualitativo puede ser usado como base para definir la hipótesis que luego puede ser correspondida cuantitativamente con la realidad. Cabe destacar que las investigaciones cuantitativas pueden obtener resultados más justificables y formales que los cualitativos.

Hay 3 tipos principales de técnicas o estrategias para la investigación empírica: las encuestas, los estudios de casos y los experimentos.

Las **encuestas** se utilizan o bien cuando una técnica o herramienta ya ha sido usada o antes de comenzar a hacerlo. Son estudios retrospectivos de las relaciones y los resultados de una situación. Se puede realizar este tipo de investigación cuando una técnica, o herramienta ya ha sido utilizada o antes de que ésta sea introducida. Las encuestas son realizadas sobre una muestra representativa de la población, y luego los resultados son generalizados al resto de la población. El ámbito donde son más usadas es en ciencias sociales, por ejemplo, para determinar cómo la población va a votar en la siguiente elección.

En la Ingeniería de Software Empírica las encuestas se utilizan de forma similar, se obtiene un conjunto de datos de un evento que ha ocurrido para determinar cómo reacciona la población frente a una técnica, herramienta o método particular, o para determinar relaciones o tendencias. En un estudio es fundamental seleccionar correctamente las variables a estudiar, pues de ellas dependen los resultados que se pueden obtener. Si los resultados no permiten concluir sobre los objetivos del estudio se han elegido mal las variables.

Una de las características más relevantes de las encuestas es que proveen un gran número de variables para estudiar. Esto hace posible construir una variedad de modelos y luego seleccionar el que mejor se ajusta a los propósitos de la investigación, evitando tener que especular cuáles son las variables más relevantes. Dependiendo del diseño de la investigación (cuestionario) las encuestas pueden ser clasificadas como cualitativas o cuantitativas.

Los **estudios de casos** son métodos observacionales, se basan en la observación de una actividad o proyecto durante su curso. Son utilizados para monitorear proyectos, o actividades y para investigar entidades o fenómenos en un período específico.

El nivel de control de la ejecución es menor en los estudios de casos que en los experimentos. Esto se debe principalmente a que en los estudios de casos no se controla, sólo se observa, contrario a lo que ocurre en los experimentos.

Los estudios de casos son muy útiles en el área de Ingeniería de Software, se usan en la evaluación industrial de métodos y herramientas. Además, son fáciles de planificar aunque los resultados son difíciles de generalizar y comprender. Los estudios de casos no manipulan las variables, sino que éstas son determinadas por la situación que se está investigando.

Al igual que las encuestas, los estudios de casos pueden ser clasificados como cualitativos o cuantitativos dependiendo de lo que se quiera investigar del proyecto en curso.

Los **experimentos** son generalmente ejecutados en un ambiente de laboratorio, el cual brinda un alto grado de control. El objetivo en un experimento es manipular una o más variables y controlar el resto. Un experimento es una técnica formal, rigurosa y controlada de llevar a cabo una investigación.

3. Experimentos Formales

Como se mencionó anteriormente, los experimentos son una técnica de investigación en la cual se quiere tener un mejor control del estudio y del entorno en el que éste se lleva a cabo.

Los experimentos son apropiados para investigar distintos aspectos de la IS, como ser: confirmar teorías, explorar relaciones, evaluar la exactitud de los modelos y validar medidas. Tienen un alto costo respecto de las otras técnicas de investigación, pero a cambio ofrecen un control total de la ejecución y son de fácil replicación.

3.1. Terminología

En esta sección se presentan los términos más comunmente usados en diseño experimental. Se usan dos ejemplos de experimentos a lo largo de esta sección para introducir dichos términos.

En el primer ejemplo se tiene un experimento en el campo de la medicina, mediante el cual se quiere conocer la efectividad de los analgésicos en las personas entre 20 y 40 años de edad, llamado «Efec-Analgésicos».

En el segundo ejemplo, se quiere conocer la efectividad de 5 técnicas de verificación sobre un conjunto de programas, llamado «Efec-Técnicas».

Los objetos sobre los cuales se ejecuta el experimento son llamados **Unidades Experimentales** u objetos experimentales. La unidad experimental en un experimento de Ingeniería de Software podría llegar a ser el proyecto de software como un todo o cualquier producto intermedio durante el proceso.

Para *Efec-Analgésicos* se tiene que la unidad experimental es un grupo de personas entre 20 y 40 años de edad, en ese grupo de personas es en donde se observa el efecto de los analgésicos. En el ejemplo de *Efec-Técnicas*, se tiene que la unidad experimental es el conjunto de programas sobre los cuales se aplican las técnicas de verificación.

Aquellas personas que aplican los métodos o técnicas a las unidades experimentales se les llama **Sujetos Experimentales**. A diferencia de otras disciplinas, en la IS los sujetos experimentales tienen un importante efecto en los resultados del experimento, por lo tanto es una variable que debe ser cuidadosamente considerada.

En *Efec-Analgésicos* los sujetos son aquellas personas que administran los analgésicos a ser consumidos por los pacientes (enfermeros por ejemplo). Cómo los enfermeros administran los analgésicos a los pacientes no es algo que se espere vaya a afectar el experimento. La forma en que un enfermero administra un analgésico a un paciente es

poco probable que sea diferente a la de otro, y aunque lo fuera, no se espera que afecte los resultados del experimento.

En *Efec-Técnicas* los sujetos pueden ser ingenieros que aplican la técnica en un conjunto particular de programas (unidad experimental). En este caso, los resultados del experimento podrían diferir mucho de acuerdo a la formación y experiencia de los ingenieros, así como también la forma en que las técnicas son aplicadas, incluso el estado de ánimo del verificador podría influir en los resultados.

El resultado de un experimento es llamado **Variable de Respuesta**. Este resultado debe ser cuantitativo. Una variable de respuesta puede ser cualquier característica de un proyecto, fase, producto o recurso que es medida para verificar los efectos de las variaciones que se provocan de una aplicación a otra. En ocasiones, a una variable de respuesta se le llama también variable dependiente.

En *Efec-Analgésicos* la efectividad podría ser medida en el grado de alivio del dolor en un determinado lapso de tiempo, o bien qué tan rápido el analgésico alivia el dolor. En ambos casos, la variable debe ser expresada cuantitativamente. En el primer caso se podría tener una escala, en la cual cada valor signifique un grado de alivio del dolor, en el segundo caso, el lapso de tiempo en que el analgésico es efectivo, se podría medir en minutos.

Para *Efec-Técnicas* la efectividad podría ser medida de acuerdo a la cantidad de defectos que encuentra la técnica sobre la cantidad de defectos totales del software verificado.

Un **Parámetro** es cualquier característica que permanezca invariable a lo largo del experimento. Son características que no influyen o que no se desea que influyan en el resultado del experimento o en la variable de respuesta. Los resultados del experimento serán particulares a las condiciones definidas por los parámetros. El conocimiento resultante podrá ser generalizado solamente considerando los parámetros como variables en sucesivos experimentos y estudiando su impacto en las variables de respuesta.

En el ejemplo de *Efec-Analgésicos* se tiene que el rango de edades (entre 20 y 40 años de edad) es un parámetro del experimento, los resultados serán particulares para el rango establecido.

En *Efec-Técnicas* un parámetro posible es el tamaño del software a ser verificado (por ejemplo: que tenga entre 200 y 500 LOCs). Otro parámetro para este experimento podría ser la experiencia de los verificadores, en este caso se podría fijar la experiencia en un determinado nivel.

Cada característica del desarrollo de software a ser estudiada que afecta a las variables de respuesta se denomina **Factor**. Cada factor tiene varias alternativas posibles. Lo que se estudia, es la influencia de las alternativas en los valores de las variables de respuesta. Los factores de un experimento son cualquier característica que es intencionalmente modificada durante el experimento y que afecta su resultado.

El factor en *Efec-Analgésicos* es «los analgésicos», en *Efec-Técnicas* tenemos que el factor es «las técnicas de verificación». Para ambos casos el factor se varía intencionalmente (se varía el tipo de analgé-

sico o tipo de técnica de verificación) para ver cómo afecta en la efectividad.

Los posibles valores de los factores en cada unidad experimental son llamados **Alternativas** o niveles. En algunos casos también se les llama tratamientos.

Las alternativas de *Efec-Analgésicos* son los distintos tipos de analgésicos que se estudian en el experimento (p.e. Aspirina, Zolben, etc). De igual forma, para *Efec-Técnicas* las distintas alternativas son los 5 tipos distintos de técnicas que se estudian.

El intento de ajustar determinadas características de un experimento a un valor constante no es siempre posible. Es inevitable y a veces indeseable tener variaciones de un experimento a otro. Éstas variaciones son conocidas como **Bloqueo de Variables** y dan lugar a un determinado tipo de diseño experimental, llamado *block design*.

Una variable indeseada para *Efec-Analgésicos* podría ser el «umbral del dolor». Si se aplica una alternativa de analgésico a personas con umbral del dolor alto y otra alternativa a personas con umbral del dolor bajo, se tendría una variación indeseada, ya que la efectividad que se mida de los distintos tipos de analgésico va a variar no solamente por el tipo de analgésico administrado sino por el nivel de umbral del dolor del paciente al cual se lo administra.

En el caso de *Efec-Técnicas*, podría resultar que la experiencia de los verificadores resultase una variación indeseada si no se la tiene en cuenta previamente. Una forma de bloquear la experiencia en verificación podría ser dividir a los participantes en dos grupos: uno de verificadores experientes y otro sin experiencia.

Cada ejecución del experimento que se realiza en una unidad experimental es llamada **experimento unitario** o experimento elemental. Lo que significa que cada aplicación de una combinación de alternativas de factores por un sujeto experimental en una unidad experimental es un experimento elemental.

Un experimento elemental es cada terna <$analgsico_i$, $enfermero_j$, $paciente_k$> para el ejemplo de *Efec-Analgésicos*. Para el ejemplo de *Efec-Técnicas* sería la terna <$tecnica_i$, $verificador_j$, $software_k$>.

La figura 1 ilustra la interacción entre los distintos tipos de componentes de un experimento.

3.2. Principios generales de diseño

Muchos aspectos deben ser tenidos en cuenta cuando se diseña un experimento. Los principios generales de diseño son: aleatoriedad, bloqueo y balance. A continuación se describe en qué consiste cada principio.

Aleatoriedad: el principio de aleatoriedad es uno de los principios de diseño más importantes. Todos los métodos de análisis estadístico requieren que las observaciones sean de variables independientes aleatorias. Por consiguiente, tanto las alternativas de los factores como los sujetos tienen que ser elegidos de forma aleatoria, ya que los sujetos tienen un impacto crítico en el valor de las variables de respuesta.

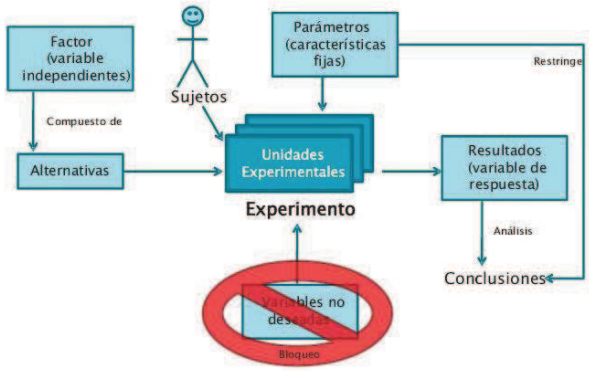


Figura 1: Componentes en un experimento de Ingeniería de Software

La aleatoriedad que se puede aplicar a un experimento también depende del tipo de diseño que se haya elegido. Por ejemplo, si se tienen dos factores A y B, cada uno con dos posibles alternativas (a1, a2, b1 y b2), las alternativas deben ser combinadas de la siguiente forma: a1b1, a1b2, a2b1, a2b2, ya que cuando se tienen dos factores se quiere observar el efecto de cada alternativa por separado y de la interacción entre ambas.

Esta combinación de alternativas es especificada por el tipo de diseño experimental que se eligió. Sin embargo, las cuatro combinaciones deben ser asignadas de forma aleatoria a los proyectos y sujetos, y es ahí en donde la aleatoriedad se aplica.

Bloqueo: la técnica de bloqueo se usa cuando se tienen factores que probablemente tengan efectos indeseados en las variables de respuesta y éstos efectos son conocidos y controlables.

Como se mencionaba en el ejemplo de *Efec-Técnicas* en la sección anterior, algunos verificadores podrían tener experiencia en el uso de las técnicas de verificación y otros no. Entonces, para minimizar el efecto de la experiencia, se agrupan a los participantes en dos grupos, uno con verificadores experientes y otro sin experiencia.

Balance: el balance es deseable ya que simplifica y fortalece el análisis estadístico de los datos, aunque no es necesario. Tomando como ejemplo el experimento de *Efec-Analgésicos* nuevamente, sería deseable que la cantidad de personas a las cuales se les administra Zolben sea igual a la cantidad de personas que se les administra Aspirina.

3.3. Tipos de Diseño

En el proceso del diseño experimental, primero se debe decidir (basándose en los objetivos del experimento) a qué factores y alternati-

vas estarán sujetas las unidades experimentales y qué parámetros deben ser establecidos. Luego, se debe examinar si existe la posibilidad de que algunos de los parámetros no pueda mantenerse en un valor constante, en ese caso se debe tener en cuenta cualquier variación indeseable. Finalmente, se debe elegir qué variables de respuesta serán medidas y cuáles serán los objetos y sujetos experimentales.

Teniendo establecidos los parámetros, factores, variables de bloqueo y variables de respuesta, se debe elegir el tipo de diseño experimental, en el cual se establece cuántas combinaciones de experimentos unitarios y alternativas deben haber.

Los distintos tipos de diseño experimental dependen del objetivo del experimento, del número de factores, de las alternativas de los factores y de la cantidad de variaciones indeseadas, entre otros.

Los tipos de diseño experimental se dividen en diseños de *un solo factor* y diseños de *múltiples factores*. A continuación se profundiza en los experimentos de un solo factor.

3.3.1. Diseño de un solo factor (*One-Factor Design*)

Para experimentos con un solo factor existen distintos tipos de diseños estándar, los principales son: los completamente aleatorios y los aleatorios con comparación por pares.

Los diseños **completamente aleatorios** son los tipos de diseño más simples, en los cuales se intenta comparar dos o más alternativas aplicadas a un determinado número de unidades experimentales, en donde cada unidad experimental se ve afectada una única vez, y por ende, por una sola alternativa. La asignación de las alternativas a los experimentos debe ser de forma aleatoria para asegurar la validez del análisis de datos.

Tomando como ejemplo *Efec-Técnicas* y suponiendo que el conjunto de programas sobre el cual se quiere conocer la efectividad de las técnicas lo componen diez programas distintos, se tendría que asignar las técnicas y los ingenieros de forma aleatoria a los programas que se vayan a verificar.

Una posible asignación aleatoria sería tener en una bolsa los nombres de todas las técnicas de verificación a aplicar, en donde la primera que se extraiga se aplique al programa P_1, la segunda a P_2 y así hasta el programa P_{10}. Luego de tener las duplas Programa-Técnica, efectuar la misma asignación aleatoria con los participantes: el primer participante extraído se lo asigna la dupla (P_1, T_x), el segundo a la dupla (P_2, T_y), y así sucesivamente.

El análisis estadístico que se puede hacer a este tipo de experimentos varía según si se aplican 2 o más alternativas para el factor.

Los diseños **aleatorios con comparación por pares** tienen como objetivo encontrar cuál es la mejor alternativa respecto de una determinada variable de respuesta. Estos tipos de diseño tienen la particularidad de que las alternativas se aplican al mismo experimento, instanciado en más de una unidad experimental.

Para el experimento de *Efec-Técnicas* no sería una buena decisión que cada ingeniero verificara 2 veces el mismo programa. En la segun-

da instancia de verificación, el ingeniero posee conocimiento tanto de los defectos del programa como de la tarea de verificar propiamente dicha (aunque sea con una técnica distinta). Por esto, para comparar las dos técnicas, ambas tienen que ser aplicadas por primera vez por ingenieros distintos, pero con similares características (ya que encontrar uno igual es imposible). La alternativa que debe aplicar cada ingeniero al programa debe ser asignada de forma aleatoria y no debe verificar un mismo programa más de una vez.

En este tipo de diseños se bloquean cierto tipo de variables que representan restricciones en la aleatoriedad que se le puede dar. Tomando como ejemplo nuevamente a *Efec-Técnicas*, si un verificador sin experiencia aplica más de una técnica durante el experimento, no sería deseable asignar al azar la técnica que cada verificador aplica en cada verificación.

Existe un efecto de aprendizaje en el cual, luego de que un verificador ejecutó una verificación, éste generó conocimiento sobre la verificación en sí, independientemente de la técnica que haya aplicado, y éste conocimiento influye significativamente en la segunda instancia de verificación que vaya a aplicar. Por tanto, la aleatoriedad en el orden de la asignación de técnicas en este ejemplo no es del todo deseable.

4. Proceso Experimental

Como se mencionó anteriormente, los experimentos son una técnica de investigación en la cual se quiere tener un mejor control del estudio y del entorno en el que éste se lleva a cabo.

Los experimentos son apropiados para investigar distintos aspectos de la IS, como ser: confirmar teorías, explorar relaciones, evaluar la exactitud de los modelos y validar medidas. Tienen un alto costo respecto de las otras técnicas de investigación, pero a cambio ofrecen un control total de la ejecución y son de fácil replicación.

El proceso para llevar a cabo un experimento está formado por varias fases: definición, planificación, operación, análisis e interpretación y presentación.

La primer fase es la de **definición**, en donde se define el experimento en términos del problema, objetivos y metas. La siguiente fase es la **planificación**, en la cual se determina el diseño del experimento. En la fase de **operación** se ejecuta el diseño del experimento, en donde se recolectan los datos que serán analizados posteriormente en la fase de **análisis e interpretación**. En esta última fase, conceptos estadísticos son aplicados para analizar los datos. Por último, se muestran los resultados obtenidos en la fase de **presentación**.

En la figura 2 se muestra una visión general de todo el proceso. Cada una de las fases que lo componen se detallan a continuación.

4.1. Definición

En la fase de Definición se determinan las bases del experimento, que se ilustra en la figura 3. Para ello se debe definir **el problema que**

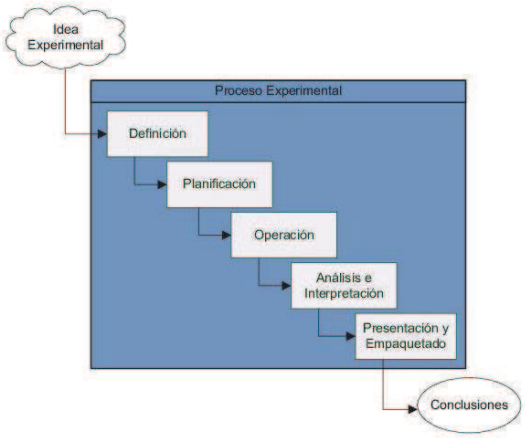


Figura 2: Visión general del Proceso Experimental

se quiere resolver, **propósito del experimento** y **los objetivos y metas** del mismo.

Figura 3: Fase de Definición del Experimento

Para el planteo del objetivo del experimento se debe definir *el objeto de estudio*, que es la entidad que va a ser estudiada en el experimento. Puede ser un producto, proceso, recurso u otro. También se debe establecer el *propósito*: la intención del experimento. Por ejemplo, evaluar diferentes técnicas de verificación.

Se debe definir además el *foco de calidad*, que refiere al efecto primario que esta bajo estudio, ejemplos son la efectividad y el costo de las técnicas de verificación. El propósito y el foco de calidad son las bases para las hipótesis del experimento.

Otro aspecto que debe estar presente es la *perspectiva*, que refiere al punto de vista con que los resultados obtenidos son interpretados. Por ejemplo, los resultados de la comparación de técnicas de verificación pueden verse desde la perspectiva de un experimentador, de un investigador o de un profesional. Un experimentador verá el estudio como una demostración de como una técnica de verificación puede ser

evaluada. Un investigador puede ver el estudio como una base empírica para refinar teorías sobre la verificación de software, enfocándose en los datos que apoyan o refutan estas teorías. Un profesional puede ver el estudio como una fuente de información sobre qué técnicas de verificación deberían aplicarse en la práctica.

Junto con los aspectos mencionados se debe definir el *contexto*, que es el ambiente en el que se ejecuta el experimento. En este punto se deben definir los *sujetos* que van a llevar a cabo el experimento y los *artefactos* que son utilizados en la ejecución del mismo. Se puede caracterizar el contexto de un experimento según el número de sujetos y objetos definidos en él: un solo objeto y un solo sujeto, un solo sujeto a través de muchos objetos, un solo objeto a través de un conjunto de sujetos, o un conjunto de sujetos y un conjunto de objetos.

4.2. Planificación

La planificación es la fase en la que se define como se va a llevar a cabo el experimento. Esta fase consta de las etapas: selección del contexto, formulación de las hipótesis, elección de las variables, selección de los sujetos, diseño del experimento, instrumentación y evaluación de la validez, que se muestran en la figura 4.

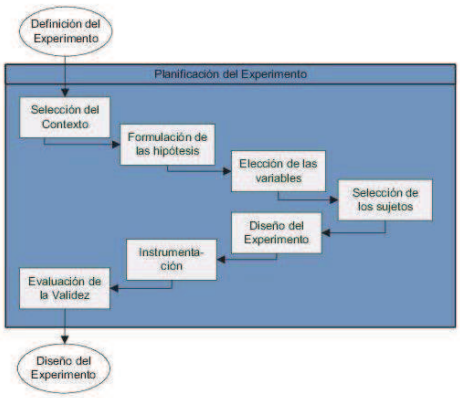


Figura 4: Fase de Planificación del Experimento

La etapa de **selección del contexto** es la etapa inicial de la planificación. En esta etapa se amplía el contexto definido en la etapa de Definición, especificando claramente las características del ambiente donde ejecuta el experimento. Se define si el experimento se va a realizar en un proyecto real (en línea, *on-line*) o en un laboratorio (fuera de línea, *off-line*), características de los sujetos y si el problema es «real» (problema existente en la industria) o «de juguete». También se debe

definir si el experimento es válido para un contexto específico o para un dominio general de la Ingeniería de Software.

Una vez que los objetivos están claramente definidos se pueden transformar en una hipótesis formal. La **formulación de las hipótesis** es una fase muy importante dentro de la etapa de planificación, ya que la verificación de la misma es la base para el análisis estadístico. En esta fase se formaliza la definición del experimento en la hipótesis.

Usualmente se definen dos hipótesis, la hipótesis nula y la hipótesis alternativa. La hipótesis nula, denotada H_0, asume que no hay una diferencia significativa entre las alternativas, con respecto a las variables dependientes que se están midiendo. Establece que si hay diferencias entre las observaciones realizadas, éstas son por casualidad, no producto de la alternativa aplicada. Esta hipótesis se asume verdadera hasta que los datos demuestren lo contrario, por lo que el foco del experimento está puesto en rechazarla. Un ejemplo de hipótesis nula es: «No hay diferencia en la cantidad de defectos encontrados por las técnicas de verificación».

En cambio la hipótesis alternativa, denotada H_1, afirma que existe una diferencia significativa entre las alternativas con respecto a las variables dependientes. Establece que las diferencias encontradas son producto de la aplicación de las alternativas. Ésta es la hipótesis a probar, para esto se debe determinar que los datos obtenidos son lo suficientemente convincentes para desechar la hipótesis nula y aceptar la hipótesis alternativa. Un ejemplo de hipótesis alternativa es, si se están comparando dos técnicas de verificación, decir que una encuentra más defectos que la otra. En caso de haber más de una hipótesis alternativa se denotan secuencialmente: $H_1, H_2, H_3, \ldots, H_n$.

Una vez definida la hipótesis, se debe identificar qué variables afectan a la/s alternativa/s. Luego de identificadas las variables se debe decidir el control a ejercer sobre las mismas.

La **selección de las variables** dependientes como la de las independientes están relacionadas, por lo que en muchos casos se realizan en simultáneo. Seleccionar estas variables es una tarea muy compleja, que en ocasiones implica conocer muy bien el dominio. Es importante definir las variables independientes y analizar sus características, para así investigar y controlar los efectos que ejercen sobre las variables dependientes. Se deben identificar las variables independientes que se pueden controlar y las que no. Además, se deben identificar las variables dependientes, mediante las cuales se mide el efecto de las alternativas. Generalmente hay sólo una variable dependiente y se deriva de la hipótesis.

Otro aspecto importante al llevar a cabo un experimento es la **selección de los sujetos**. Para poder generalizar los resultados al resto de la población, la selección debe ser una muestra representativa de la misma. Cuanto más grande es la muestra, menor es el error al generalizar los datos. Existen dos tipos de muestras que se pueden seleccionar: la probabilística, donde se conoce la probabilidad de seleccionar cada sujeto; y la no-probabilística, donde esta probabilidad es desconocida.

Luego de definir el contexto, formalizar las hipótesis, y seleccionar las variables y los sujetos, se debe **diseñar el experimento**. Es muy importante planear y diseñar cuidadosamente el experimento, para que los datos obtenidos puedan ser interpretados mediante la aplicación de métodos de análisis estadísticos.

Para comenzar a diseñar un experimento se debe elegir el diseño adecuado. Se debe planificar y diseñar el conjunto de las combinaciones de alternativas, sujetos y objetos, que conforman los experimentos unitarios. Se describe cómo estos experimentos unitarios deben ser organizados y ejecutados.

La elección del diseño del experimento afecta el análisis estadístico y viceversa, por lo que al elegir el diseño del experimento se debe tener en cuenta qué análisis estadístico es el mejor para rechazar la hipótesis nula y aceptar la alternativa.

Luego de diseñar el experimento y antes de la ejecución es necesario contar con todo lo necesario para la correcta ejecución del mismo. La **instrumentación** involucra, de ser necesario, capacitación a los sujetos, preparación de los artefactos, construcción de guías, descripción de procesos, planillas y herramientas. También implica configurar el hardware, mecanismos de consultas y experiencias piloto, entre otros. La finalidad de esta fase es proveer todo lo necesario para la realización y monitorización del experimento.

4.3. Evaluación de la Validez

Una pregunta fundamental antes de pasar a ejecutar el experimento es cuán válidos serían los resultados. Existen cuatro categorías de amenazas a la validez: validez de la conclusión, validez interna, validez del constructo y validez externa.

Las amenazas que afectan la **validez de la conclusión** refieren a las conclusiones estadísticas. Amenazas que afecten la capacidad de determinar si existe una relación entre la alternativa y el resultado, y si las conclusiones obtenidas al respecto son válidas. Ejemplos de estas son la elección de los métodos estadísticos, y la elección del tamaño de la muestra, entre otros.

Las amenazas que influyen en la **validez interna** son aquellas referidas a observar relaciones entre la alternativa y el resultado que sean producto de la casualidad y no del resultado de la aplicación de un factor. Esta «casualidad» es provocada por elementos desconocidos que influyen sobre los resultados sin el conocimiento de los investigadores. Es decir, la validez interna se basa en asegurar que la alternativa en cuestión produce los resultados observados.

La **validez del constructo** indica cómo una medición se relaciona con otras de acuerdo con la teoría o hipótesis que concierne a los conceptos que se están midiendo. Un ejemplo se puede observar al momento de seleccionar los sujetos en un experimento. Si se utiliza como medida de la experiencia del sujeto el número de cursos que tiene aprobados en la universidad, no se está utilizando una buena medida de la experiencia. En cambio, una buena medida puede ser utilizar la

cantidad de años de experiencia en la industria o una combinación de ambas cosas.

La **validez externa** está relacionada con la habilidad para generalizar los resultados. Se ve afectada por el diseño del experimento. Los tres riesgos principales que tiene la validez externa son: tener los participantes equivocados como sujetos, ejecutar el experimento en un ambiente erróneo y realizar el experimento en un momento que afecte los resultados.

4.4. Operación

Luego de diseñar y planificar el experimento, éste debe ser ejecutado para recolectar los datos que se quieren analizar. La operación del experimento consiste en tres etapas: preparación, ejecución y la validación de los datos, que se muestran en la figura 5.

Figura 5: Fase de Operación del Experimento

En la etapa de preparación se seleccionan los sujetos y se preparan los artefactos a ser utilizados.

Es muy importante que los sujetos estén motivados y dispuestos a realizar las actividades que les sean asignadas, ya sea que tengan conocimiento o no de su participación en el experimento. Se debe intentar obtener consentimiento por parte de los participantes, que deben estar de acuerdo con los objetivos de la investigación. Los resultados obtenidos pueden volverse inválidos si los sujetos al momento que deciden participar no saben lo que tienen que hacer o tienen un concepto erróneo.

Es importante considerar la sensibilidad de los resultados que se obtienen de los sujetos, por ejemplo: es importante asegurar a los participantes que los resultados obtenidos sobre su rendimiento se mantienen en secreto y no se usarán para perjudicarlos en ningún sentido. Se debe tener en cuenta también los incentivos, ya que ayudan a motivar a los sujetos, pero se corre el riesgo de que participen sólo por el incentivo, lo que puede ser perjudicial para el experimento. En caso de no tener otra alternativa que no sea engañar a los sujetos, se debe procurar explicar y revelar el engaño lo más temprano posible.

Como se vio en la instrumentación, para que los sujetos comiencen la ejecución es necesario tener prontos todos los instrumentos, formularios, herramientas, guías y otros artefactos que sean necesarios para la ejecución del experimento. Muchas veces se debe preparar un

conjunto de instrumentos especial para cada sujeto y otras se utiliza el mismo conjunto de artefactos para todos los sujetos.

Existen muchas formas distintas de ejecutar los experimentos, la duración varía desde días hasta años.

Los datos pueden ser recolectados de las siguientes formas:

- Manualmente mediante el llenado de formularios por parte de los sujetos.
- Manualmente soportado por herramientas.
- Mediante entrevistas.
- Automáticamente por herramientas.

La primera es la forma más común y no requiere mucho esfuerzo por parte del experimentador. Tanto en los formularios como en los métodos soportados por herramientas no es posible identificar inconsistencias o defectos hasta que no se recolecte la información, o hasta que los sujetos los descubran. En las entrevistas, el contacto con los sujetos es mucho mayor permitiendo una mejor comunicación con ellos durante la recolección de datos. Éste método es el que requiere mas esfuerzo por parte del investigador.

Un aspecto muy importante a la hora de ejecutar los experimentos es el ambiente de ejecución, tanto si el experimento se realiza dentro de un proyecto de desarrollo común o si se crea un ambiente ficticio para su ejecución. En el primer caso el experimento no debería afectar el proyecto más de lo necesario, ya que la razón de realizar el experimento dentro de un proyecto es ver los efectos de las alternativas en el ambiente del proyecto. Si el experimento cambia demasiado el ambiente del proyecto, éstos efectos se perderían.

Cuando se obtienen los datos, se debe chequear que fueron recolectados correctamente y que son razonables. Algunas fuentes de error son que los sujetos llenen mal sus planillas, o no recolecten los datos seriamente, lo que hace que se descarten datos. Es importante revisar que los sujetos hagan un trabajo serio y responsable y que apliquen las alternativas en el orden correcto, en otras palabras: que el experimento sea ejecutado en la forma en que fue planificado. De lo contrario los resultados podrían ser inválidos.

4.5. Análisis e Interpretación

Luego de que finaliza la ejecución del experimento y se cuenta con los datos recolectados, comienza la fase de análisis de los mismos conforme a los objetivos planteados.

Un aspecto importante a considerar en el análisis de los datos es la **escala de medida**. La escala de medida utilizada para recolectar los datos restringe el tipo de cálculos estadísticos que se pueden realizar. Una medida es un mapeo de un atributo de una entidad a un valor de medida, por lo general un valor numérico. Las entidades son objetos que se observan en la realidad, por ejemplo, una técnica de verificación.

El propósito de mapear los atributos en un valor de medida es caracterizar y manipular los atributos formalmente. La medida seleccionada debe ser válida, por tanto, no debe violar ninguna propiedad necesaria del atributo que mide, y debe ser una caracterización matemática adecuada del atributo.

El mapeo de un atributo a un valor de medida puede realizarse de varias formas. Cada tipo de mapeo posible de un atributo se conoce como escala. Los tipos más comunes de escala son:

- Escala Nominal.- Es la menos poderosa de las escalas. Solo mapea el atributo de la entidad en un nombre o símbolo. El mapeo puede verse como una clasificación de las entidades acorde al atributo. Ejemplos de escala nominal son clasificaciones, etiquetados, entre otras.
- Escala Ordinal.- La escala ordinal categoriza las entidades según un criterio de ordenación. Es más poderosa que la escala nominal. Ejemplos de criterios de ordenación son «mayor que», «mejor que» y «más complejo». Ejemplos de escala nominal son grados, complejidad del software, entre otras.
- Escala de intervalo.- La escala de intervalo se utiliza cuando la diferencia entre dos medidas es significativa, pero no el valor en si mismo. Este tipo de escala ordena los valores de la misma forma que la escala ordinal, pero existe la noción de «distancia relativa» entre dos entidades. Esta escala es más poderosa que la ordinal. Ejemplos de escala de intervalo son la temperatura medida en Celsius o Fahrenheit.
- Escala ratio (cociente de dos números).- Si existe un valor cero significativo y la división entre dos medidas es significativa, se puede utilizar una escala ratio. Ejemplos de escala ratio son distancia, temperatura medida en Kelvin, etc.

Después de obtener los datos es necesario interpretarlos para llegar a conclusiones válidas. La interpretación se realiza en tres etapas: caracterizar el conjunto de datos usando estadística descriptiva, reducción del conjunto de datos y realización de las pruebas de hipótesis que se ilustran en la figura 6.

Figura 6: Fase de Análisis e Interpretación de los Datos del Experimento

4.5.1. Estadística Descriptiva

La **estadística descriptiva** se utiliza antes de la prueba de hipótesis, para entender mejor la naturaleza de los datos y para identificar datos falsos o anormales. Los aspectos principales que se examinan son: la tendencia central, la dispersión y la dependencia. A continuación se presentan las medidas más comunes de cada uno de estos aspectos. Para ello se asume que existen $x_1 \dots x_n$ muestras.

Las **medidas de tendencia central** indican «el medio» de un conjunto de datos. Entre las medidas más comunes se encuentran: la media aritmética, la mediana y la moda.

La *media aritmética* se conoce como el promedio, y se calcula sumando todas las muestras y dividiendo el total por el número de muestras:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{1}$$

La *media*, denotada $\bar{x}$, resume en un valor las características de una variable teniendo en cuenta a todos los casos. Es significativa para las escalas de intervalo y ratio.

La *mediana*, denotada $\tilde{x}$, representa el valor medio de un conjunto de datos, tal que el número de muestras que son mayores que la mediana es el mismo que el número de muestras que son menores que la mediana. Se calcula ordenando las muestras en orden ascendente o descendente, y seleccionando la observación del medio. Este cálculo está bien definido si n es impar. Si n es par, la mediana se define como la media aritmética de los dos valores medios. Esta medida es significativa para las escalas ordinal, de intervalo y ratio.

La *moda* representa la muestra más común. Se calcula contando el número de muestras para cada valor único y seleccionando el valor con más cantidad. La moda esta bien definida si hay solo un valor más común que los otros. Si este no es el caso, se calcula como la mediana de las muestras más comunes. La moda es significativa para las escalas nominal, ordinal, de intervalo y ratio.

La media aritmética y la mediana son iguales si la distribución de las muestras es simétrica. Si la distribución es simétrica y tiene un único valor máximo, las tres medidas son iguales.

Las medidas de tendencia central no proveen información sobre la dispersión del conjunto de datos. Cuanto mayor es la dispersión, más variables son las muestras, cuanto menor es la dispersión, más homogéneas a la media son las muestras.

Las **medidas de dispersión** miden el nivel de desviación de la tendencia central, o sea, que tan diseminados o concentrados están los datos respecto al valor central. Entre las principales medidas de dispersión están: la varianza, la desviación estándar, el rango y el coeficiente de variación.

La *varianza* (s^2) que presenta una distribución respecto de su media se calcula como la media de las desviaciones de las muestras respec-

to a la media aritmética. Dado que la suma de las desviaciones es siempre cero, se toman las desviaciones al cuadrado:

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 \quad (2)$$

Se divide por $n-1$ y no por n, porque dividir por $n-1$ provee a la varianza de propiedades convenientes. La varianza es significativa para las escalas de intervalo y ratio.

La *desviación estándar*, denotada s, se de ne como la raíz cuadrada de la varianza:

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \quad (3)$$

A menudo esta medida se prefiere sobre la varianza porque tiene las mismas dimensiones (unidad de medida) que los valores de las muestras. En cambio, la varianza se mide en unidades cuadráticas. La desviación estándar es significativa para las escalas de intervalo y ratio.

La dispersión también se puede expresar como un porcentaje de la media. Este valor se llama *coeficiente de variación*, y se calcula como:

$$100 \cdot \frac{s}{\bar{x}} \quad (4)$$

Esta medida no tiene dimensión y es significativa para la escala ratio. Permite comparar la dispersión o variabilidad de dos o más grupos.

El **rango** de un conjunto de datos es la distancia entre el valor máximo y el mínimo:

$$range = x_{max} - x_{min} \quad (5)$$

Es una medida significativa para las escalas de intervalo y ratio. Cuando el conjunto de datos consiste en muestras relacionadas de a pares (xi; yi) de dos variables, X e Y, puede ser interesante examinar la dependencia entre estas variables. Las principales medidas de dependencia son: regresión lineal, covarianza y el coeficiente de correlación lineal.

4.5.2. Reducción del Conjunto de Datos

Para las pruebas de hipótesis se utilizan métodos estadísticos. El resultado de aplicar estos métodos depende de la calidad de los datos. Si los datos no representan lo que se cree, las conclusiones que se derivan de los resultados de los métodos son incorrectas. Errores en el conjunto de datos pueden ocurrir por un error sistemático, o por lo que se conoce en estadística con el nombre de outlier. Un outlier es un dato mucho más grande o mucho más chico de lo que se puede esperar observando el resto de los datos.

Las estadísticas descriptivas se ven fuertemente influenciadas por aquellas observaciones que su valor dista significativamente del resto

de los valores recolectados. Estas observaciones llevan el nombre de *outliers*.

Los *outliers* influencian las medidas de dispersión, aumentando la variabilidad de lo que se está midiendo. En algunos casos se realiza un análisis acerca de estos valores que difieren mucho de la media y se decide quitarlos de los datos a analizar porque no son representativos de la población, ya que fueron causados por algún tipo de anomalía: errores de medición, variaciones no deseadas en las características de los sujetos, entre otras.

Quitar *outliers* requiere de un análisis pormenorizado, por quitar outliers se demoró en detectar el agujero de la capa de ozono. [1]

Una vez identificado un outlier se debe identificar su origen para decidir qué hacer con él. Si se debe a un evento raro o extraño que no volverá a ocurrir, el punto puede ser excluido. Si se debe a un evento extraño que puede volver a ocurrir, no es aconsejable excluir el valor del análisis, pues tiene información relevante. Si se debe a una variable que no fue considerada, debería ser considerado para basar los cálculos y modelos también en esta variable.

4.5.3. Pruebas de Hipótesis

El objetivo de la **prueba de hipótesis** es ver si es posible rechazar cierta hipótesis nula H_0. Si la hipótesis nula no es rechazada, no se puede decir nada sobre los resultados. En cambio, si es rechazada, se puede declarar que la hipótesis es falsa con una significancia dada (α). Este nivel de significancia también es denominado nivel de riesgo o probabilidad de error, ya que se corre el riesgo de rechazar la hipótesis nula cuando en realidad es verdadera. Este nivel está bajo el control del experimentador.

Para probar H_0 se define una unidad de prueba t y un área crítica C, la cual es parte del área sobre la que varía t. A partir de estas definiciones se formula la prueba de significancia de la siguiente forma:

- Si $t \in C$, rechazar H_0
- Si $t \notin C$, no rechazar H_0

Por ejemplo, un experimentador observa la cantidad de defectos detectados por LOC de una técnica de verificación desconocida bajo determinadas condiciones, y quiere probar que no es la técnica B, de la cual sabe que en las mismas condiciones (programa, verificador, etc.) detecta 1 defecto cada 20 LOC. El experimentador sabe que también pueden haber otras técnicas que detecten 1 defecto cada 20 LOC. A partir de esto se define la hipótesis nula: "H_0: La técnica observada es la B". En este ejemplo, la unidad de prueba t es cada cuantos LOC se detecta un defecto y el área crítica es

[1] En 1985 los científicos británicos anunciaron un agujero en la capa de ozono sobre el polo sur. El reporte fue descartado ya que observaciones más completas, obtenidas por instrumentos satelitales, no mostraban nada inusual. Luego, un análisis más exhaustivo reveló que las lecturas de ozono en el polo sur eran tan bajas que el programa que las analizaba las había suprimido automáticamente como outliers en forma equivocada.

$C = \{1, 2, 3, \ldots, 19, 21, 22, \ldots\}$. La prueba de significancia es: si $t \leq 19$ o $t \geq 21$, rechazar H_0, de lo contrario no rechazar H_0.

Si se observa que $t = 20$, la hipótesis no puede ser rechazada ni se pueden derivar conclusiones, pues pueden haber otras técnicas que detecten un defecto cada 20 LOC.

El área crítica, C, puede tener distintas formas, lo más común es que tenga forma de intervalo, por ejemplo: $t \leq a$ o $t \geq b$. Si C consiste en uno de estos intervalos es unilateral. Si consiste de dos intervalos ($t \leq a$, $t \geq b$, donde $a < b$), es bilateral.

Hay varios métodos estadísticos, de aquí en adelante denotados *tests*, que pueden utilizarse para evaluar los resultados de un experimento, más específicamente para determinar si se rechaza la hipótesis nula. Cuando se lleva a cabo un *test* es posible calcular el menor valor de significancia posible (denotado p-valor) con el cual es posible rechazar la hipótesis nula. Se rechaza la hipótesis nula si el p-valor asociado al resultado observado es menor o igual que el nivel de significancia establecido.

Las siguientes son tres probabilidades importantes para la prueba de hipótesis:

- α = P(cometer el error tipo I) = P(rechazar H_0 | H_0 es verdadera). Es la probabilidad de rechazar H_0 cuando es verdadera.
- β = P(cometer el error tipo II) = P(no rechazar H_0 | H_0 es falsa). Es la probabilidad de no rechazar H_0 cuando es falsa.
- Poder = $1 - \beta$ = P(rechazar H_0 | H_0 es falsa). El poder de prueba es la probabilidad de rechazar H_0 cuando es falsa.

El experimentador debería elegir un test con un poder de prueba tan alto como sea posible. Hay varios factores que afectan el poder de un test. Primero, el test en sí mismo puede ser más o menos efectivo. Segundo, la cantidad de muestras: mayor cantidad de muestras equivale a un poder de prueba más alto. Otro aspecto es la selección de una hipótesis alternativa unilaterial o bilateral. Una hipótesis unilateral da un poder mayor que una bilateral.

La probabilidad de cometer un error tipo I se puede controlar y reducir. Si la probabilidad es muy pequeña, sólo se rechazará la hipótesis nula si se obtiene evidencia muy contundente en contra de esta hipótesis. La probabilidad máxima de cometer un error tipo I se conoce como la significancia de la prueba (α).

Los valores de uso más común para la significancia de una prueba son 0.01, 0.05 y 0.10. La significancia es en ocasiones presentada como un porcentaje, tal como 1%, 5% o 10%. Esto quiere decir que el experimentador está dispuesto a permitir una probabilidad de 0.01, 0.05, o 0.10 de rechazar la hipótesis nula cuando es cierta, o sea, de cometer un error tipo I.

El valor de la significancia es seleccionado antes de comenzar a hacer el experimento en una de varias formas.

El valor de α puede estar establecido en el área de investigación, por ejemplo: se puede obtener de artículos que se publican en revistas

científicas. Otra forma de seleccionarlo es que sencillamente sea impuesto por la persona o compañía para la cual se trabaja. Finalmente, puede ser seleccionado tomando en cuenta el costo de cometer un error tipo I. Mientras más alto el costo, más pequeña debe ser la probabilidad α de cometer un error tipo I. El valor usual de α en las ciencias naturales y sociales es de 0.05. En Ingeniería de Software, el valor de α aún no se encuentra establecido.

Existen dos tipos de tests: paramétricos y no paramétricos. Los **tests paramétricos** están basados en un modelo que involucra una distribución específica. En la mayoría de los casos, se asume que algunos de los parámetros involucrados en un test paramétrico están normalmente distribuidos. Los tests paramétricos también requieren que los parámetros puedan ser medidos al menos en una *escala de intervalo*. Si los parámetros no pueden medirse en al menos una escala de intervalo, generalmente no se puede utilizar un test paramétrico. En este caso hay un amplio rango de tests no paramétricos disponible.

Los **tests no paramétricos** no asumen lo mismo respecto a la distribución de los parámetros, son más generales que los paramétricos. Un test no paramétrico se puede utilizar en vez de un test paramétrico, pero el caso inverso no siempre puede darse.

En la elección entre un test paramétrico y un test no paramétrico hay dos aspectos a considerar:

- Aplicabilidad.- Es importante que las suposiciones en cuanto a las distribuciones de parámetros y las que conciernen a las escalas sean realistas.
- Poder.- El poder de los tests paramétricos es generalmente mayor que el de los tests no paramétricos. Por lo tanto, los test paramétricos requieren menos datos (experimentos más pequeños), que los tests no paramétricos, siempre que sean aplicables.

Aunque es un riesgo utilizar tests paramétricos cuando no se cuenta con las condiciones requeridas, en algunos casos vale la pena tomar el riesgo. Algunas simulaciones han mostrado que los tests paramétricos son bastante robustos a las desviaciones de las pre-condiciones (escala de intervalo), mientras las desviaciones no sean demasiado grandes.

En el caso de las pruebas paramétricas, se exige que la distribución de la muestra se aproxime a una normal. Para poder utilizar aproximación normal se requiere un tamaño mínimo de la muestra, dependiendo del $p(value)$ que se requiera (11). En el cuadro 1 se muestran los tamaños mínimos de muestra para los distintos $p(value)$.

Los test paramétricos más usados en experimentos de Ingeniería de Software son:

- ANOVA (*ANalysis Of VAriance*) (18).
- ANOM (*ANalysis Of Means*) (6).

Ambos tests (ANOVA y ANOM), pueden utilizarse para diseños de un solo factor con múltiples alternativas. En ambos test la hipótesis nula

p(value)	Tamaño mínimo de muestra
0.50	n = 30
0.40 ó 0.60	n = 50
0.30 ó 0.70	n = 80
0.20 ó 0.80	n = 200
0.10 ó 0.90	n = 600

Cuadro 1: Estadísticas descriptivas de la Efectividad

refiere a la igualdad de las medias (como es habitual en los test paramétricos):

$H_0 : \bar{x}_1 = \bar{x}_2 = \ldots = \bar{x}_I$

En ANOVA, la variación en la respuesta se divide en la variación entre los diferentes niveles del factor (los diferentes tratamientos) y la variación entre individuos dentro de cada nivel. El objetivo principal del ANOVA es contrastar si existen diferencias entre las diferentes medias de los niveles de las variables (factores).

En el caso de ANOM, este test no solamente responde a la pregunta de si hay o no diferencias entre las alternativas, sino que cuando hay diferencias, también dice cuáles alternativas son mejores y cuáles peores.

Los test no paramétricos más usado son:

- Kruskal Wallis.
- Mann-Whitney.

En el caso de los test no paramétricos, la hipótesis nula refiere a la igualdad de las medianas:

$H_0 : \tilde{x}_1 = \tilde{x}_2 = \ldots = \tilde{x}_I$

Rechazar H_0 significa que existe evidencia estadística como para afirmar de que hay diferencias entre las alternativas. En el caso de que hubiera más de dos alternativas, para conocer cuál es la alternativa que difiere es necesario comparar las alternativas de a dos.

En el caso de Kruskal Wallis, a pesar de no requerir una distribución normal para las muestras, sus resultados se pueden ver afectados por lo que se le llama «heterocedasticidad» de los datos. Cuando una muestra presenta datos heterocedásticos (no presentan homocedasticidad) el test de Kruskal Wallis podría dar un resultado no significativo (no rechazando H_0), aunque haya una diferencia real entre las muestras (debería rechazar H_0).

Para probar la homocedasticidad de los datos se suele utilizar el test de Levene. Las hipótesis del test de Levene son:

- $H_0 : \sigma_1 = \sigma_2 = \ldots = \sigma_k$ donde σ_a es la varianza de la muestra a.
- $H1 : \sigma_i \neq \sigma_j = \ldots = \sigma_k$ para al menos un par de muestras (i, j), donde σ_a es la varianza de la muestra a.

Para poder aplicar ANOVA, y en algunos casos Kruskal-Wallis, es necesario que el test de Levene no sea significativo (no se rechaza H_0), o sea, que las varianzas de las muestras sean similares o iguales. Esto prueba la homocedasticidad de los datos.

Una vez que se prueba que al menos dos de las k muestras provienen de poblaciones distintas (datos heterocedásticos) se puede aplicar, entre otros, el test de Mann-Whitney para comparar las muestras dos a dos.

Si se presume que una alternativa puede ser mejor o peor que el resto, esto quiere decir que hay un «ordenamiento» entre ellas, lo aconsejable es realizar un test de ordenamiento. Algunos test de ordenamiento son:

- Jonckheere-Terpstra Test. (5)
- Test para alternativas ordenadas L. (5)

Para los test de ordenamiento, las hipótesis que se plantean son las siguientes:

$H_0 : \tilde{x}_1 = \tilde{x}_2 = \ldots = \tilde{x}_I$

$H_1 : \tilde{x}_1 \leq \tilde{x}_2 \leq \ldots \leq \tilde{x}_I$ (con al menos una desigualdad estricta)

4.6. Presentación y Empaquetado

En la presentación y el empaquetado de un experimento es esencial no olvidar aspectos e información necesaria para que otros puedan replicar o tomar ventaja del experimento y del conocimiento ganado durante su ejecución.

El esquema de reporte de un experimento generalmente cuenta con los siguientes títulos: Introducción, Definición del Problema, Planificación del Experimento, Operación del Experimento, Análisis de Datos, Interpretación de los Resultados, Discusión y Conclusiones, y Apéndice.

En la *Introducción* se realiza una introducción al área y los objetivos de la investigación. En la *Definición del Problema* se describe en mayor profundidad el trasfondo de la investigación, incluyendo las razones para realizarla. En la *Planificación del Experimento* se detalla el contexto del experimento incluyendo las hipótesis, que se derivan de la definición del problema, las variables que se deben medir (tanto independientes como dependientes), la estrategia de medida y análisis de datos, los sujetos que participaran de la investigación y las amenazas a la validez.

En la *Operación del Experimento* se describe como preparar la ejecución del mismo, incluyendo aspectos que permitan facilitar la replicación y descripciones que indiquen cómo se llevaron a cabo las actividades. Debe incluirse la preparación de los sujetos, cómo se recolectaron los datos y cómo se realizó la ejecución.

En el *Análisis de Datos* se describen los cálculos y los modelos de análisis específicos utilizados. Se debe incluir información, como por ejemplo, tamaño de la muestra, niveles de significancia y métodos estadísticos utilizados, para que el lector conozca los pre-requisitos para el análisis. En la *Interpretación de los Resultados* se rechaza la hipótesis

nula o se concluye que no puede ser rechazada. Aquí se resume cómo utilizar los datos obtenidos en el experimento. La interpretación debe realizarse haciendo referencia a la validez. También se deben describir los factores que puedan tener un impacto sobre los resultados.

Finalmente, en *Discusión y Conclusiones* se presentan las conclusiones y los hallazgos como un resumen de todo el experimento, junto con los resultados, problemas y desviaciones respecto al plan. También se incluyen ideas sobre trabajos a futuro. Los resultados deberían ser comparados con los obtenidos por trabajos anteriores, de manera de identificar similitudes y diferencias. La información que no es vital para la presentación se incluye en el Apéndice. Esto puede ser, por ejemplo, los datos recavados y más información sobre sujetos y objetos. Si la intención es generar un paquete de laboratorio, el material utilizado en el experimento puede ser proveído en el apéndice.

5. Estudio de casos

Un estudio de casos en ingeniería de software es una investigación empírica que se basa en múltiples fuentes de evidencia para investigar un fenómeno de ingeniera de software contemporánea dentro de su contexto real, especialmente cuando los límites entre el fenómeno y su contexto no son claramente evidentes.

La conducción de un estudio de casos consiste de las siguientes fases: diseño, preparación, recolección, análisis y reporte. En esta sección se presentan los aspectos a considerar durante el diseño de estudio de casos y diversas técnicas de recolección de datos.

5.1. Diseño

Al diseñar un estudio de casos se deben considerar los siguientes elementos:

Razón fundamental del estudio:

El investigador debe tener clara la razón de llevar a cabo el estudio. Para investigaciones académicas una razón típica es generar una contribución, por ejemplo generar una nueva teoría o hipótesis. En la industria las razones pueden ser brindar una mejora a una organización o un proyecto.

Objetivo del estudio:

El objetivo general del estudio es una declaración de lo que el investigador espera alcanzar como resultado de la realización de ese estudio. El objetivo se refina en un conjunto de preguntas de investigación y éstas se responden mediante el análisis de los datos.

El caso y la unidad de Análisis:

Los investigadores hacen una distinción entre el estudio de casos y la unidad o unidades de análisis dentro del caso. Las unidades de análisis permiten definir qué es el caso. Cuando el estudio de caso se realiza sobre un objeto concreto, por ejemplo una persona (pacientes, líderes, estudiantes...), la unidad de análisis está muy clara porque es el propio objeto investigado. En cambio, en estudio de casos sobre

fenómenos o acontecimientos más complejos de definir, es necesario considerar una o varias unidades de análisis que permitan dar un paso más en la concreción de la investigación. Las unidades de análisis permiten definir los límites del caso para diferenciarlos de su contexto.

En los estudios de casos, el caso y la unidad de análisis deben ser seleccionadas intencionalmente. Esto se contrasta con las encuestas y experimentos, en donde los sujetos son una muestra de la población.

Tipos de diseños:

Yin propone una tipología que establece cuatro formatos básicos de estudio de casos, que resultan de la combinación de dos características: la primera es si el estudio incluye un único caso o si contiene más de uno (múltiple) (19). La segunda característica es si su análisis tiene una sola unidad de análisis, es decir, un sentido holístico, o si se divide en diversas unidades de análisis parciales (encapsulado). Tanto el estudio de un sólo caso, como el estudio de casos múltiples, puede ser, a su vez, holístico o encapsulado, resultando los cuatro tipos planteados por el autor. En el cuadro 2 se presenta la tipología mencionada.

	Diseño de caso único	**Diseño de múltiples casos**
Holístico	Tipo 1	Tipo 3
Encapsulado	Tipo 2	Tipo 4

Cuadro 2: Tipos básicos de diseño para estudios de casos

Holístico o encapsulado: La unidad de análisis puede ser un individuo, un grupo, una compañía, un país, etc. La unidad de análisis ayuda a definir el alcance del caso. El caso es con frecuencia un proceso, una institución, o un evento no tan bien definido como un individuo. La definición de la unidad de análisis está vinculada con la forma en que se presentaron las primeras preguntas de la investigación. Si solo se busca examinar la naturaleza general de una empresa o problema, se utiliza un enfoque holístico. Se procede así cuando no se logra identificar sub-unidades o sectores o cuando la naturaleza del estudio es holística. Si se examinan una o varias sub-unidades de una organización o programa, se utiliza un enfoque encapsulado.

Diseños simples o múltiples: Los diseños simples se utilizan cuando, de modo análogo a un experimento, un caso permite probar una nueva teoría, o establece las circunstancias en que valdrían ciertas proposiciones. También un diseño simple se aplica en casos únicos o extremos, o un caso "revelatorio", en el que se presenta a los ojos del investigador un fenómeno antes no estudiado. Los diseños múltiples, por otra parte, tienen la ventaja de que su evidencia es más convincente y el estudio resulta más robusto. Sus desventajas consisten en que no permiten tratar con el caso revelatorio, o raro, o crítico, de los casos simples y, además, requiere más recursos. El tema del número de casos que conviene analizar es debatido. Algunos autores se inclinan por el estudio de un solo caso y citan para avalar su posición ejemplos de casos clásicos, como Street Corner Society, que mostrarían la

importancia de concentrarse en el estudio a fondo de un único caso. Eisenhardt (2) sostienen en cambio que es posible obtener recursos para casos múltiples; de hecho, hay ejemplos de casos múltiples ya clásicos. Smith (8) relata que, en su experiencia, a medida que cada caso progresa a través de entrevistas los datos se van adecuando a un patrón, "en otras palabras, una teoría (va) emergiendo" y los datos sucesivos se hacen predecibles a partir de la teoría. Cuando se verifica este fenómeno, al cual se suele llamar saturación, puede decirse que el número de casos considerado es suficiente.

Pregunta de investigación:

Las preguntas de investigación son afirmaciones sobre el conocimiento que se busca, o se espera descubrir durante el estudio de casos.

El uso de estudio de casos es adecuado cuando la pregunta de investigación es un ¿cómo? o un ¿por qué?, cuando el investigador tiene poco control sobre los eventos y cuando el énfasis de la investigación está puesto sobre eventos contemporáneos dentro de su contexto en la vida real.

Una pregunta de investigación puede estar relacionada con una hipótesis, que es una supuesta explicación de un aspecto del fenómeno de estudio.

Proposiciones e Hipótesis:

Las proposiciones son predicciones acerca del mundo que pueden deducirse lógicamente de la teoría, nos orientan sobre los objetos que deben ser examinados en el estudio; desmenuzan las preguntas de tipo "cómo" y "por qué" para determinar qué debemos estudiar. Las hipótesis se generan a partir de las preposiciones y deben ser empíricamente testeables.

Control y aseguramiento de la calidad, cuestiones legales, éticas y profesionales:

Existen diversos métodos para asegurar la calidad del diseño del estudio de casos, dentro de los cuales se encuentran, que el diseño del estudio de casos sea revisado por personas ajenas al proyecto y/o conducir un estudio pilota para evaluar el diseño contruido.

Al comenzar la preparación del estudio de casos el investigador debe comprometerse a proteger a los candidatos del estudio. Como parte de la protección debe comprometerse a realizar un estudio de casos que:

- tenga el consentimiento de todos los candidatos que formaran parte del estudio, alertando del objetivo y que estos sean voluntarios del mismo.
- proteja al candidato de cualquier engaño en el estudio
- proteja la privacidad y confidencialidad de los datos de los candidatos (nombres, edades)
- tome precauciones sobre candidatos vulnerables (niños)

5.2. Recolección de datos

En esta sección se detallan diversas tecnicas de recolección de datos. Las técnicas de recolección de datos se pueden dividir en tres grados según Lethbridge (12):

- Primer grado: Estos son métodos directos, donde el investigador está en contacto directo con el entrevistado y recopila los datos en tiempo real. Ejemplos de estos métodos son entrevistas, grupos focales, encuestas Delphi y observaciones.
- Segundo grado: Estos son métodos indirectos donde el investigador recoge directamente los datos sin tener que interactuar con el entrevistado. Ejemplos de este método son herramientas automáticas que monitorean y observan.
- Tercer grado: estos son los métodos en donde el investigador analiza el trabajo que ya está disponible. Este enfoque se utiliza cuando se analizan especificaciones, documentos, informes, etc.

La técnica de primer grado tiende a ser más cara de aplicar que la segunda o tercera técnica debido a que se requiere un esfuerzo significativo por parte del investigador y los sujetos. Una ventaja de las técnicas de primer y segundo grado es la facilidad de controlar los datos recolados por el investigador y el contexto de recolección. Las técnicas de tercer grado son más baratas pero no ofrecen el mismo control al investigador.

Es importante decidir cuidadosamente que datos recolectar y como recolectarlos. Mientras los objetivos y las preguntas de investigación del estudio de casos están siendo decididos es imposible definir esta selección. Sin embargo, cuando el estudio de casos ya está en progreso, el investigador tiene una mayor apreciación de los datos deseables, los disponibles y los factibles de recolectar y luego analizar.

Entrevistas: La recolección de datos basada en entrevistas es de las más usadas y mas importantes en los estudios de casos en ingeniería de software. Casi todos los estudios de casos implican algún tipo de entrevista, ya sea para la recolección de datos primarios o para validaciones de tipos de datos. La razón de ello es que gran parte del conocimiento que es de interés no está disponible en ningún otro lugar más que en las mentes de las personas que participan del estudio de casos. Las entrevistas pueden ser llevadas a cabo de diferentes maneras, pero en general el investigador habla y hace preguntas al interrogado, quien responde a estas. Dentro de las variantes al conducir una entrevista podemos detallar, sesiones de entrevistas más o menos estructuradas, entrevistas más o menos largas, preguntas más o menos específicas.

El dialogo entre el investigador y los entrevistados es guiado por un conjunto de preguntas. Las preguntas están basadas en los tópicos de interés del estudio de casos y pueden ser abiertas o cerradas. Una pregunta abierta permite una respuesta amplia, sin embargo una pregunta cerrada limita la respuesta a un cierto rango de opciones. La

ventaja de las preguntas abiertas es que el alcance de la respuesta no tiene límite a diferencia de las cerradas. La ventaja de las preguntas cerradas es que es más fácil analizar los resultados.

Por otra parte las entrevistas se dividen en semi estructuradas, no estructuradas y completamente estructuradas. En las entrevistas no estructuradas las preguntas son formuladas de forma abierta. En este caso la conversación se realiza en base a los intereses del entrevistado y del investigador. Este tipo de entrevistas en más factible para estudios exploratorios. En las entrevistas semi estructuradas las preguntas son planificadas, pero no necesariamente se piden en el orden en que fueron listadas. El estilo de preguntas es un mix entre abiertas y cerradas. Entrevistas semi estructuradas son comunes en estudios de casos de ingeniería de software. En las entrevistas completamente estructuradas las preguntas son planificadas en detalle y se piden en el mismo orden en que se planeo. Para este tipo de entrevistas es deseable que las preguntas sean cerradas, generalmente resultan ser cuestionarios.

Durante la fase de planificación del estudio de casos se determina a quién entrevistar intentando involucrar personas con diferentes roles y personalidades. La cantidad de entrevistados debe ser decidida durante el estudio de casos. El criterio para determinar el máximo número de entrevistados es la saturación, es decir, cuando no se obtiene más información agregando un nuevo integrante.

Una vez formuladas las preguntas de la entrevista, pueden ser utilizadas sobre un primer conjunto de entrevistados (entrevista piloto). Esto permite comprobar que las preguntas fueron comprendidas por los entrevistados y que las respuestas brindadas son útiles para el investigador. Las entrevistas pilotos son conducidas de igual forma que una entrevista tradicional, pero con un extra de atención para conocer la comprensión de las preguntas.

Una sesión de entrevista se puede dividir en varias fases. Al comienzo el investigador presenta los objetivos de la entrevista y del estudio de casos y explica como los datos recabados de la entrevista van a ser usados. Durante esta etapa el investigador puede pedir permiso de grabar la entrevista. Luego de esta introducción tanto el entrevistado como el investigador se sienten más cómodos y relajados. Las preguntas de la entrevista, que constituyen la parte más larga de la entrevista se realizan a continuación de la introducción. Si la entrevista contiene preguntas personales o sensibles, concernientes a aspectos políticos, opiniones sobre colegas, conflictos, la competencia, etc., se debe tener especial cuidado. En esta situación es importante que el entrevistado confié en el entrevistador, y que este ultimo garantice la confidencialidad de los datos brindados.

Diferentes tipos de preguntas pueden hacerse de acuerdo con los siguientes tres modelos. El modelo embudo comienza con preguntas más abiertas y continua con preguntas más especificas. El modelo pirámide comienza con preguntas especificas y luego se van realizando preguntas abiertas a lo largo de la entrevista. En el modelo reloj de arena se realiza un mix de preguntas específicas y abiertas.

Grupos focales: A diferencia de las entrevistas, en los grupos de enfoque el investigador realiza una serie de preguntas a un grupo de

personas al mismo tiempo. Este tipo de enfoque es muy útil de usar en estudios cualitativos. En grupo las personas tienden a soltarse más en el sentido de confirmar o rechazar hechos que han ocultado en la entrevista individual. Es rentable (costo/efectividad) debido a que varias personas son entrevistadas al mismo tiempo. Este tipo de enfoque tiene la debilidad de que el grupo puede ser desordenado, hablar uno encima del otro, etc. En estos casos de debe contar con un moderador con experiencia que pueda llevar adelante la sesión.

Observaciones: Las observaciones son conducidas para investigar como una cierta actividad es conducida por un grupo de personas. Existen varias alternativas de observaciones, una de ellas es monitorear al grupo grabando un video y luego analizar lo grabado, observar una reunión, o utilizar herramientas que hagan preguntas a los participantes cada ciertos intervalos de tiempo.

Datos de archivos: Datos de archivos se refiere a datos que se encuentran disponibles en archivos. Ejemplos de estos son: minutas de reuniones, documentos técnicos, documentos de gestión, registros financieros, reportes, etc. Datos de archivo es una técnica de recolección de datos de tercer grado en un estudio de casos.

Métricas: Las técnicas de recolección mencionadas anteriormente se enfocan en datos cualitativos, sin embargo los datos cuantitativos también son de importancia en un estudio de casos. Medir el software es el proceso de representar las entidades del software, los procesos y productos en valores cuantitativos. En importante definir métricas que sean realmente de interés para que luego la recolección sea exitosa. En este método se definen en primera instancia los objetivos, las preguntas son refinadas basadas en dichos objetivos y luego las métricas se derivan de las preguntas, por lo tanto las métricas que se recogen son relevantes.

Para fortalecer la validez de la investigación empírica se recomienda tomar múltiples perspectivas hacia el objeto de estudio y ofrecer así una visión amplia, lo que se denomina triangulación. Se pueden aplicar los siguientes cuatro tipos de triangulación:

- Datos de origen: utilizando más de una fuente de datos o recogiendo los mismos datos en diferentes ocasiones.
- Observadores: utilizando más de un observador en el estudio.
- Metodológica: combinando diferentes tipos de métodos de recolección de datos.
- Teoría: utilizando teorías o puntos de vistas alternativos.

6. Investigación

El GrIS ha llevado adelante una serie de experimentos formales para conocer el comportamiento de distintas técnicas de verificación, ha empaquetado un experimento realizado y ha evaluado el funcionamiento del paquete. Además propuso un marco de comparación de experimentos formales.

Actualmente han culminado 4 experimentos. El primero se llevó a cabo en la Facultad de ingeniería en el 2008 (15). En este experimento se entrenó a un grupo de 17 estudiantes para aplicar 5 técnicas de verificación a un pequeño programa escrito en Java. Los estudiantes registran los defectos que encuentran. Cada estudiante aplicó sólo una de las técnicas. Las técnicas son inspecciones, partición en clases de equivalencia y valores limites, tablas de decisión, trayectorias linealmente independientes y cubrimiento de condición múltiple. Las conclusiones más relevantes fueron tres. En primer lugar se detectó que los defectos de performance no son fáciles de encontrar. En segundo lugar el costo de la verificación unitaria es alto y el porcentaje de defectos que se detecta es bajo. Por último, con la técnica de inspecciones se detecta una mayor variedad de defectos que con las otras técnicas.

En el siguiente experimento formal se busca conocer la efectividad de 5 técnicas de verificación unitaria (14). Las técnicas son: inspecciones, partición en clases de equivalencia y valores limites, tablas de decisión, trayectorias linealmente independientes y cubrimiento de condición múltiple. El diseño propuesto es de un factor con múltiples alternativas. El factor es la técnica de verificación y las alternativas las 5 posibilidades. El experimento es ejecutado por un conjunto de 14 estudiantes que aplican las técnicas sobre 4 programas diferentes desarrollados especialmente para este experimento. Los 14 sujetos participaron en el experimento anterior, donde fueron entrenados con el uso de las técnicas. Cada sujeto (menos uno) verifica 3 de los 4 programas aplicando una técnica distinta. El análisis de resultados se realiza utilizando estadística descriptiva y la prueba de hipótesis no paramétrica Mann-Whitney. Los resultados de la prueba indican que las técnicas partición en clases de equivalencia y tablas de decisión son más efectivas que trayectorias linealmente independientes.

En el 2011 se lleva a cabo un experimento formal que compara el comportamiento de las técnicas de verificación Cubrimiento de sentencias (CS) y Todos los usos (TU) (17). El diseño de este experimento es también de un factor con dos alternativas. Un grupo de 21 estudiantes realizan pruebas sobre un único programa en Java. En este caso el experimento se divide en dos experiencias, una con 10 sujetos (5 ejecutan CS y 5 ejecutan TU) y otra con 11 sujetos (5 ejecutan CS y 6 ejecutan TU). Como resultado utilizando estadística descriptiva se obtiene que la técnica TU tiene una mayor efectividad que la técnica CS. La efectividad promedio de TU es de 34,3% y 29,2% la de CS. Sin embargo, los test estadísticos realizados no rechazan la hipótesis de igualdad en la efectividad de ambas técnicas. Por otro lado, se encuentra que la técnica TU es notoriamente más costosa que la técnica CS; 328 minutos para TU y 133 minutos para CS es lo que llevó en promedio desarrollar los casos de prueba para el software bajo prueba. Los test estadísticos aplicados muestran que en este experimento hay suficiente evidencia estadística para afirmar que TU es más costosa que CS.

Parte de los estudiantes que participaron del experimento anterior forman parte de un nuevo experimento que busca comparar el comportamiento de las técnicas de verificación Cubrimiento de Senten-

cias (CS) y Todos los Usos (TU). Se busca analizar las técnicas con el propósito de conocer su efectividad y costo a nivel unitario, en el contexto de un experimento controlado. Participan 14 sujetos del experimento anterior, probando sobre un mismo programa, 6 de ellos aplican CS y 8 aplican TU. La ejecución del experimento se realiza en una única instancia con una duración de 8 semanas. Los resultados obtenidos con los test de hipótesis indican que no existe evidencia estadística para afirmar que una técnica es más efectiva que la otra. Respecto a la comparación del costo de las técnicas, se concluye que existe evidencia estadística que indica que TU es más costosa que CS. Este experimento no se encuentra aún publicado.

Además de haber realizado diversos experimentos, es de interés del GrIS contar con un marco de comparación de experimentos que permita compararlos formalmente y avanzar en la construcción de nuevos experimentos basándose en experimentos anteriores. Se propone un marco de comparación de experimentos formales que evalúan técnicas de verificación y como ejemplo se utiliza el marco con algunos experimentos conocidos (16). Se buscan características relevantes de los experimentos para poder compararlos, siendo los objetivos de los experimentos el primer punto de interés de forma de determinar si tienen el mismo propósito general. Luego el marco propone examinar los factores y alternativas de los diseños elegidos, así como las características de los sujetos participantes, la duración del experimento, la distribución de los sujetos, los lineamientos seguidos para la asignación de las técnica y programa a los sujetos, etc. Este marco busca proveer formalidad al momento de comparar distintos experimentos. Estas comparaciones no son triviales debido a la alta variabilidad de las características de los experimentos. Como resultado se obtiene una mayor comprensión de los aspectos más relevantes de cada experimento y se los puede comparar rápidamente según distintos aspectos relevantes.

Un paquete de laboratorio es el contenedor del conocimiento y materiales necesarios para replicar un experimento. En 2012 se construye un paquete de laboratorio (PL) para un experimento controlado de ingeniería de software que se realizó en el marco de trabajo del GrIS (1); para esto utilizamos la propuesta de empaquetamiento de experimentos de Solari (9). El estudio tiene dos objetivos: obtener una instancia de paquete para un experimento concreto y validar la propuesta genérica. Para realizar la instancia del PL se siguió un proceso definido que incluyó actividades de revisión, verificación y validación de los resultados. Los resultados del trabajo confirman la viabilidad y completitud de la propuesta de PL. Se ha obtenido un documento que abarca las distintas actividades del proceso experimental y contiene el conocimiento relativo al experimento en una única estructura.

Luego de presentada la propuesta de estructura de PL para experimentos (1) se busca validar la misma (10). Se realiza una evaluación a una réplica de experimento que utiliza un PL estructurado de acuerdo a la propuesta. Se pretende validar que el uso del PL implica una mejora con respecto a otros PL no estructurados. En los resultados obtenidos se pudo comprobar que el PL utilizado es completo con respecto al

proceso de investigación experimental, cubriendo las actividades de replicación en mayor grado que los PL no estructurados. También se comprobó que el documento tiene un grado de usabilidad aceptable desde el punto de vista del replicador responsable. La evaluación de la replicación muestra que la misma se ha realizado en forma más eficaz que otras anteriores. Esto se ha valorado analizando los incidentes de replicación y dudas surgidas. En cuanto a la eficiencia, la replicación evaluada fue más compleja que otras anteriores y demandó mayor esfuerzo. Sin embargo, no se han observado efectos negativos por el uso del PL.

Además, hace varios años que se realizan pruebas de procesos de desarrollo de software en el marco de una asignatura llamada Proyecto de Ingeniería de Software (13). Si bien estas pruebas no son formales, es interesante en un futuro formalizarlas.

7. Conclusiones

En el reporte se presentan conceptos teóricos básicos de la Ingeniería de Software Empírica y las principales técnicas para la investigación empírica: encuestas, estudios de casos y experimentos. Se pretende que este documento pueda ser utilizado por estudiantes e investigadores que se encuentran realizando trabajos de ISE en el marco del trabajo del GrIS.

También se describen algunos de los trabajos de investigación que ha realizado el GrIS, experimentos formales que buscan conocer el comportamiento de distintas técnicas de verificación, el paquete de laboratorio para experimentos y la evaluación realizada al mismo, y se describe el marco de comparación para experimentos formales.

Bibliografía

(1) C. Apa, M. Solari, D. Vallespir, and S. Vegas. Construcción de un paquete de laboratorio para un experimento en ingeniería de software. In *Proceedings Ibero-American Conference on Software Engineering*, 2011. 6

(2) K. M. Eisenhardt. *Building theories from case study research.* The Academy of Management Review, 1989. 5.1

(3) N. E. Fenton and S. L. Pfleeger. *Software Metrics: A Rigorous and Practical Approach, Revised.* Course Technology, February 1998. 1

(4) N. Juristo and A. M. Moreno. *Basics of Software Engineering Experimentation.* Kluwer Academic Publishers, 2001. 1

(5) E. J. Martínez. Notas del curso de posgrado maestría en estadística matemática. Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, 2004. 4.5.3

(6) P. Nelson, M. Coffin, and K. Copeland. *Introductory statistics for engineering experimentation.* Elsevier Science, California, 2003. 4.5.3

(7) R. Runeson, Höst and Regnell. *Case Study Research in Software Engineering.* John Wiley & Sons, New Jersey, USA, 2012. 1

(8) N. C. Smith. *The Case Study: A Useful Research Method For Information Management.* Journal of Information Technology, London, 1990. 5.1

(9) M. Solari. *Propuesta de Paquete de Laboratorio para Experimentos de Ingeniería de Software.* PhD thesis, Universidad Politécnica de Madrid, 2012. 6

(10) M. Solari, C. Apa, and S. Vegas. Evaluación del uso de un paquete de laboratorio en una replicación experimental. In *VIII Experimental Software Engineering Latin American Workshop (ESELAW 2011)*, 2011. 6
(11) M. Spiegel. *Estadística - 2da Edición*. Mc.Graw-Hill, Madrid, 1991. 4.5.3
(12) S. E. S. T.C. Lethbridge and J. Singer. *Studying software engineers: data collection techniques for software field studies*. Empirical Software Engineering, 2005. 5.2
(13) J. Triñanes. Construcción de un banco de pruebas de modelos de proceso. In *Jornadas Iberoamericanas de Ingeniería de Software e Ingeniería del Conocimiento*, 2004. 6
(14) D. Vallespir, C. Apa, S. De León, R. Robaina, and J. Herbert. Effectiveness of five verification techniques. In IEEE-Computer-Society, editor, *Proceedings of the International Conference of the Chilean Computer Society*, 2009. 6
(15) D. Vallespir and J. Herbert. Effectiveness and cost of verification techniques: Preliminary conclusions on five techniques. In IEEE-Computer-Society, editor, *Proceedings of the Mexican International Conference in Computer Science*, 2009. 6
(16) D. Vallespir, S. Moreno, C. Bogado, and J. Herbert. Towards a framework to compare formal experiments that evaluate verification techniques. In *Proceedings of the Mexican International Conference in Computer Science*, 2009. 6
(17) D. Vallespir, S. Moreno, C. Bogado, and J. Herbert. Comparando las técnicas de verificación todos los usos y cubrimiento de sentencias. In *Jornadas Iberoamericanas de Ingeniería de Software e Ingeniería del Conocimiento*, 2010. 6
(18) C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000. 1, 4.5.3
(19) R. K. Yin. *Case Study Research: Design and Methods*. SAGE Publications, 2003. 5.1

Appendix 3

Data Quality Metrics for the PSP

This appendix presents the information, categorization and results for each metric defined for the quality assessment of the data recollected by the students who performed the PSP I/II revised course and the PSP Fundamentals and Advance course.

Metric Categorization						
#	Name	Quality dimension	Quality factor	Source	Category	Quality Problem
1	Domain integrity in planning phase time	Consistency	Domain integrity	Grading checklist	Data error	Domain integrity rules
2	Domain integrity in design phase time	Consistency	Domain integrity	Grading checklist	Data error	Domain integrity rules
3	Domain integrity in design review phase time	Consistency	Domain integrity	Grading checklist	Data error	Domain integrity rules
4	Domain integrity in code phase time	Consistency	Domain integrity	Grading checklist	Data error	Domain integrity rules
5	Domain integrity in code review phase time	Consistency	Domain integrity	Grading checklist	Data error	Domain integrity rules
6	Domain integrity in compile phase time	Consistency	Domain integrity	Grading checklist	Data error	Domain integrity rules
7	Domain integrity in unit testing phase time	Consistency	Domain integrity	Grading checklist	Data error	Domain integrity rules
8	Domain integrity in post mortem phase time	Consistency	Domain integrity	Grading checklist	Data error	Domain integrity rules
9	Domain integrity in defect removal time	Consistency	Domain integrity	Grading checklist	Data error	Domain integrity rules
10	Domain integrity in reused parts	Consistency	Domain integrity	Grading checklist	Data error	Domain integrity rules
11	Domain integrity in total time	Consistency	Domain integrity	Grading checklist	Data error	Domain integrity rules
12	Domain integrity in the planned size (A&M)	Consistency	Domain integrity	Use of tool	Data error	Domain integrity rules
13	Domain integrity in actual total size	Consistency	Domain integrity	Use of tool	Data error	Domain integrity rules
14	Domain integrity in actual added size	Consistency	Domain integrity	Use of tool	Data error	Domain integrity rules
15	Domain integrity in actual A&M size	Consistency	Domain integrity	Use of tool	Data error	Domain integrity rules
16	Uniqueness in defects records	Uniqueness	Duplication	Use of tool	Data error	Duplicate register
22	Suspicious interrupt time when no interruption occurs	Consistency	Intra-relationship integrity	Grading checklist	Questionable	Intra-relationship rules
23	Suspicious interrupt time when big delta occurs	Consistency	Intra-relationship integrity	Grading checklist	Questionable	Intra-relationship rules

24	Injection phase vs. Removal fase	Consistency	Intra-relationship integrity	Grading checklist	Data error	Intra-relationship rules
25	Domain integrity in Fix Defect	Consistency	Intra-relationship integrity	Grading checklist	Data error	Intra-relationship rules
26	Actual Total Size vs. Size Estimating Template	Consistency	Intra-relationship integrity	Grading checklist	Questionable	Intra-relationship rules
34	Test Report existence	Completeness	Coverage	Grading checklist	Questionable	Non-existing register
35	Parts existence	Completeness	Coverage	Grading checklist	Data error	Non-existing register
37	Defect log existence	Completeness	Coverage	Use of tool	Questionable	Non-existing register
38	Null value in planning phase time	Completeness	Density	Grading checklist	Data error	Null value
39	Null value in design phase time	Completeness	Density	Grading checklist	Data error	Null value
40	Null value in design review phase time	Completeness	Density	Grading checklist	Data error	Null value
41	Null value in code phase time	Completeness	Density	Grading checklist	Data error	Null value
42	Null value in code review phase time	Completeness	Density	Grading checklist	Data error	Null value
43	Null value in unit testing phase time	Completeness	Density	Grading checklist	Data error	Null value
44	Null value in post mortem phase time	Completeness	Density	Grading checklist	Data error	Null value
45	Null value in the defect description	Completeness	Density	Grading checklist	Data error	Null value
46	Null value in defect removal time	Completeness	Density	Grading checklist	Data error	Null value
47	Null value in fix defect	Completeness	Density	Grading checklist	Data error	Null value
48	Null value in problem description	Completeness	Density	Grading checklist	Data error	Null value
49	Null value in improvement description	Completeness	Density	Grading checklist	Data error	Null value
50	Null value in expected results	Completeness	Density	Grading checklist	Data error	Null value
51	Null value in actual results	Completeness	Density	Grading checklist	Data error	Null value
52	Null values in part name	Completeness	Density	Grading checklist	Data error	Null value
53	Null value in added part type	Completeness	Density	Grading checklist	Data error	Null value
54	Null value in plan items of added parts	Completeness	Density	Grading checklist	Data error	Null value

55	Null value in plan relative size of added parts	Completeness	Density	Grading checklist	Data error	Null value
56	Null value in plan size of added parts	Completeness	Density	Grading checklist	Data error	Null value
57	Null value in total time	Completeness	Density	Grading checklist	Data error	Null value
58	Null value in total actual time	Completeness	Density	Use of tool	Data error	Null value
59	Null value in total planned time	Completeness	Density	Use of tool	Data error	Null value
60	Null value in the planned size (A&M)	Completeness	Density	Use of tool	Data error	Null value
61	Null value in actual total size	Completeness	Density	Use of tool	Data error	Null value
62	Null value in actual added size	Completeness	Density	Use of tool	Data error	Null value
63	Null value in actual added and modified size	Completeness	Density	Use of tool	Data error	Null value
64	Outliers in the time by planning phase	Accuracy	Syntactic accuracy	Grading checklist	Questionable	Out of range value
65	Outliers in the time by design phase	Accuracy	Syntactic accuracy	Grading checklist	Questionable	Out of range value
66	Outliers in the time by design review phase	Accuracy	Syntactic accuracy	Grading checklist	Questionable	Out of range value
67	Outliers in the time by code phase	Accuracy	Syntactic accuracy	Grading checklist	Questionable	Out of range value
68	Outliers in the time by code review phase	Accuracy	Syntactic accuracy	Grading checklist	Questionable	Out of range value
69	Outliers in the time by compile phase	Accuracy	Syntactic accuracy	Grading checklist	Questionable	Out of range value
70	Outliers in the time by unit testing phase	Accuracy	Syntactic accuracy	Grading checklist	Questionable	Out of range value
71	Outliers in the time by post mortem phase	Accuracy	Syntactic accuracy	Grading checklist	Questionable	Out of range value
75	Defect removal date vs Phase date time	Consistency	Referential integrity	Grading checklist	Data error	Referential integrity rules
76	Defects removal time vs planning phase time	Consistency	Referential integrity	Grading checklist	Data error	Referential integrity rules
77	Defects removal time vs design phase time	Consistency	Referential integrity	Grading checklist	Data error	Referential integrity rules
78	Defects removal time vs design review phase time	Consistency	Referential integrity	Grading checklist	Data error	Referential integrity rules

79	Defects removal time vs code phase time	Consistency	Referential integrity	Grading checklist	Data error	Referential integrity rules
80	Defects removal time vs code review phase time	Consistency	Referential integrity	Grading checklist	Data error	Referential integrity rules
81	Defects removal time vs compile phase time	Consistency	Referential integrity	Grading checklist	Data error	Referential integrity rules
82	Defects removal time vs unit testing phase time	Consistency	Referential integrity	Grading checklist	Data error	Referential integrity rules
83	Defects removal time vs post mortem phase time	Consistency	Referential integrity	Grading checklist	Data error	Referential integrity rules
84	Not recorded defects	Consistency	Referential integrity	Use of tool	Questionable	Referential integrity rules
85	Granularity in defect times	Accuracy	Precision	Integrated base	Data error	Precision in defects time
86	Correctness in Project Ids	Accuracy	Semantic accuracy	Integrated base	Data error	Incorrect Project Id
87	Duplicate students	Uniqueness	Duplication	Integrated base	Data error	Duplicate register
88	Invalid reference in rlogddetail table	Consistency	Referential integrity	Integrated base	Data error	Invalid reference
89	Invalid reference in rlogtdetail table	Consistency	Referential integrity	Integrated base	Data error	Invalid reference
90	Invalid reference in rparts table	Consistency	Referential integrity	Integrated base	Data error	Invalid reference
91	Invalid reference in rparttypestandard table	Consistency	Referential integrity	Integrated base	Data error	Invalid reference
92	Invalid reference in rphasedata table	Consistency	Referential integrity	Integrated base	Data error	Invalid reference
93	Invalid reference in rphases table	Consistency	Referential integrity	Integrated base	Data error	Invalid reference
94	Invalid reference in rpips table	Consistency	Referential integrity	Integrated base	Data error	Invalid reference
95	Invalid reference in rprocesses table	Consistency	Referential integrity	Integrated base	Data error	Invalid reference
96	Invalid reference in rprocessphase table	Consistency	Referential integrity	Integrated base	Data error	Invalid reference
97	Invalid reference in rprogramsize table	Consistency	Referential integrity	Integrated base	Data error	Invalid reference
98	Invalid reference in rprojects table	Consistency	Referential integrity	Integrated base	Data error	Invalid reference
99	Invalid reference in rpspassgtdata table	Consistency	Referential integrity	Integrated base	Data error	Invalid reference
100	Invalid reference in rscheduleweeks table	Consistency	Referential integrity	Integrated base	Data error	Invalid reference

101	Invalid reference in rsetadd table	Consistency	Referential integrity	Integrated base	Data error	Invalid reference
102	Invalid reference in rsetbase table	Consistency	Referential integrity	Integrated base	Data error	Invalid reference
103	Invalid reference in rsetreuse table	Consistency	Referential integrity	Integrated base	Data error	Invalid reference
104	Invalid reference in rtasks table	Consistency	Referential integrity	Integrated base	Data error	Invalid reference
105	Invalid reference in rtaskscheduleplans table	Consistency	Referential integrity	Integrated base	Data error	Invalid reference
106	Invalid reference in rtestreports table	Consistency	Referential integrity	Integrated base	Data error	Invalid reference
107	Invalid reference in ruserprofiles table	Consistency	Referential integrity	Integrated base	Data error	Invalid reference

Metric Information						
#	Name	Definition	Semantic	Result measurement unit	Granularity	Measurement method
1	Domain integrity in planning phase time	Planning registered time must be grater than zero	Find the values less than or equal to zero on the attribute *actual_minutes* for the *planning phase*	Boolean	Cell	SQL Query
2	Domain integrity in design phase time	Design registered time must be grater than zero	Find the values less than or equal to zero on the attribute *actual_minutes* for the *design phase*	Boolean	Cell	SQL Query
3	Domain integrity in design review phase time	Design review registered time must be grater than zero	Find the values less than or equal to zero on the attribute *actual_minutes* for the *design review phase*	Boolean	Cell	SQL Query
4	Domain integrity in code phase time	Code registered time must be grater than zero	Find the values less than or equal to zero on the attribute *actual_minutes* for the *coding phase*	Boolean	Cell	SQL Query
5	Domain integrity in code review phase time	Code review registered time must be grater than zero	Find the values less than or equal to zero on the attribute *actual_minutes* for the *code review phase*	Boolean	Cell	SQL Query
6	Domain integrity in compile phase time	Compile registered time must be equal or grater than zero	Find the values less than zero on the attribute *actual_minutes* for the *compile phase*	Boolean	Cell	SQL Query
7	Domain integrity in unit testing phase time	Unit Testing registered time must be grater than zero	Find the values less than or equal to zero on the attribute *actual_minutes* for the *unit testing phase*	Boolean	Cell	SQL Query
8	Domain integrity in post mortem phase time	Post Mortem registered time must be grater than zero	Find the values less than or equal to zero on the attribute *actual_minutes* for the *post mortem phase*	Boolean	Cell	SQL Query
9	Domain integrity in defect removal time	Defect removal time must be grater than zero for each defect	Find values less than or equal to zero in the attribute *fix_time* in minutes	Boolean	Cell	SQL Query
10	Domain integrity in reused parts	Each reused parts must have a value grater than zero in plan or actual	Find values less than or equal to zero in the attributes: *planned_size* and *actual_size*	Boolean	Cell	SQL Query

11	Domain integrity in total time	If the PROBE method choosen is D, total time must be grater than zero	For each program that selected the PROBE method D, find values less than or equal to zero in the attribute *total_time*	Boolean	Cell	SQL Query
12	Domain integrity in the planned size (A&M)	Total planned size (A&M) must be grater than zero	Find values less than or equal to zero in the attribute *total_planned_size*	Boolean	Cell	SQL Query
13	Domain integrity in actual total size	The actual size *Total* must be grater than zero	Find values less than or equal to zero in the attribute *Total*	Boolean	Cell	SQL Query
14	Domain integrity in actual added size	The actual size *Added* must be grater than zero	Find values less than or equal to zero in the attribute Added	Boolean	Cell	SQL Query
15	Domain integrity in actual A&M size	The actual size *A&M* must be grater than zero	Find values less than or equal to zero in the attribute A&M	Boolean	Cell	SQL Query
16	Uniqueness in defects records	It doesn't exist two defects registered at the same time	Find the defects that where registered exactly at the same time, and the time is not 00:00:00	Boolean	Tuple	SQL Query
22	Suspicious interrupt time when no interruption occurs	Interrupt times are considered suspicious if contain the value 0 for all time records	Find time records that meet that interrupt time recorded is equal to 0 for all phases and all programs	Boolean	Tuple	SQL Query
23	Suspicious interrupt time when big delta occurs	Interrupt times is considered suspicious if delta time is greater than 180 minutes in some record	Find time records that meet that the interrupt time is 0 and the time spent is more than 180 minutes	Boolean	Tuple	SQL Query
24	Injection phase vs. Removal fase	For the same defect, the injection phase has to be prior to the removal phase	Find the defect records that meet that: *defect_injection_phase > defect_removal_phase,* according to the order of the phases	Boolean	Tuple	SQL Query
25	Domain integrity in Fix Defect	The *Fix Defect id* of a defect must precede the id defect that is being recorded and it must belog to the same program	For each defect that has an associated fix_defect, find those that do not meet that: *(defect_id > id_fix_defect) AND (program_id_of_defect = prog_id_of_fix_defect)*	Boolean	Tuple	SQL Query
26	Actual Total Size vs. Size Estimating Template	The actual total size entered in the Planning Summary is greater (up to a 10%) or equal the following: *B - D + A&M - M + R*	Find the values of T (actual_total_size) where: *T >= (B - D + A&M - M + R)*1,10*	Boolean	Cell	SQL Query

34	Test Report existence	The test report contains at least the amount of records required by each program. * 8 exercise course: Ej 1, 2 tests. Ej 2, 2 tests. Ej 3, 4 tests. Ej 4, 2 tests. Ej 5, 3 tests. Ej 6, 3 tests. Ej 7, 4 tests. Ej 8, 2 tests. * 7 exercise course: Ej 1, 2 tests. Ej2, 2 tests. Ej 3, 4 tests. Ej 4, 2 tests. Ej 5, 3 tests. Ej 6, 3 tests. Ej 7, 4 tests.	Find the programs that do not contain at least the amount of records required for this program in the Test Report	Boolean	Tuple	SQL Query
35	Parts existence	It exists at least one base part or one added part in the Size Estimating Template	Find the programs that do not have at least one record of base part nor one record of added part in Size Estimating Template	Boolean	Tuple	SQL Query
37	Defect log existence	For each student, it exists at least one defect record in the Defect log, considering all the programs	Find the student that have not recorded at least one defect in all the course programs	Boolean	Tuple	SQL Query
38	Null value in planning phase time	There must be a register of the time spent on the planning phase	Find the null values on the attribute *actual_minutes* for the planning phase	Boolean	Cell	SQL Query
39	Null value in design phase time	There must be a register of the time spent on the design phase	Find the null values on the attribute *actual_minutes* for the design phase	Boolean	Cell	SQL Query
40	Null value in design review phase time	There must be a register of the time spent on the design review phase	Find the null values on the attribute *actual_minutes* for the design review phase	Boolean	Cell	SQL Query
41	Null value in code phase time	There must be a register of the time spent on the coding phase	Find the null values on the attribute *actual_minutes* for the coding phase	Boolean	Cell	SQL Query
42	Null value in code review phase time	There must be a register of the time spent on the code review phase	Find the null values on the attribute *actual_minutes* for the code review phase	Boolean	Cell	SQL Query
43	Null value in unit testing phase time	There must be a register of the time spent on the unit testing phase	Find the null values on the attribute *actual_minutes* for the unit testing phase	Boolean	Cell	SQL Query
44	Null value in post mortem phase time	There must be a register of the time spent on the post mortem phase	Find the null values on the attribute *actual_minutes* for the post mortem phase	Boolean	Cell	SQL Query
45	Null value in the defect description	All the defects must have a description	Find null values in the attribute *description*	Boolean	Cell	SQL Query
46	Null value in defect removal time	All the defects must have a removal time	Find null values in the attribute *fix time* in minutes	Boolean	Cell	SQL Query

47	Null value in fix defect	Defect injected in Compile or Testing must have a "*Fix Defect*" associated	Find the defect injected in testing or compilation phases where the attribute *fix_defect* is null	Boolean	Cell	SQL Query
48	Null value in problem description	The PIP formulary must have a non-empty problem description	Find null values in the attribute *problem description* in the PIP formulary	Boolean	Cell	SQL Query
49	Null value in improvement description	The PIP formulary must have a non-empty improvement description	Find null values in the attribute *improvement description* in the PIP formulary	Boolean	Cell	SQL Query
50	Null value in expected results	The Test Report field expected results cannot be null	Find all test report records where the attributes *expected_result* is null	Boolean	Cell	SQL Query
51	Null value in actual results	The Test Report field actual results cannot be null	Find all test report records where the attributes *actual_results* is null	Boolean	Cell	SQL Query
52	Null values in part name	The part names (*base, added* and *reused*) cannot be null	Find null values in the attribute *name_of_part*	Boolean	Cell	SQL Query
53	Null value in added part type	Each added part cannot contain a null value in the *Part Type* field	Find null values in the attribute *part_type*	Boolean	Cell	SQL Query
54	Null value in plan items of added parts	Each added part cannot contain a null value in the *Plan items* field	Find null values in the attribute *methods*	Boolean	Cell	SQL Query
55	Null value in plan relative size of added parts	Each added part cannot contain a null value in the *Plan relative size* field	Find null values in the attribute *rel_size*	Boolean	Cell	SQL Query
56	Null value in plan size of added parts	Each added part cannot contain a null value in the *Plan size* field	Find null values in the attribute *planned_loc*	Boolean	Cell	SQL Query
57	Null value in total time	If the PROBE method choosen is D, total time cannot be null	For each program that selected the PROBE method D, find null values in the attribute *total_time*	Boolean	Cell	SQL Query
58	Null value in total actual time	The total actual time in the *Project Plan Summary* cannot be null	Find null values in the attribute *actual_total_time*	Boolean	Cell	SQL Query
59	Null value in total planned time	The total planned time cannot be null	Find null values in the attribute *planned_total_time*	Boolean	Cell	SQL Query
60	Null value in the planned size (A&M)	The planned size (A&M) cannot be null	Find null values in the attributes *planned_total_size*	Boolean	Cell	SQL Query
61	Null value in actual total size	The actual *Total* size cannot be null	Find null values in the attribute: *Total*	Boolean	Cell	SQL Query
62	Null value in actual added size	The actual size *Added* cannot be null	Find null values in the attribute: *Added*	Boolean	Cell	SQL Query
63	Null value in actual added and modified size	The actual size *A&M* cannot be null	Find null values in the attribute: *A&M*	Boolean	Cell	SQL Query

64	Outliers in the time by planning phase	The times recorded for the planning phase are not far than 3 standard deviations from the average	Calculate the average time spent on the planning phase, per program and per course. Then find the values of times which are outside the range [max(average - 3*standard_dev, 1), average + 3*standard_dev]	Boolean	Cell	SQL Query
65	Outliers in the time by design phase	The times recorded for the design phase are not far than 3 standard deviations from the average	Calculate the average time spent on the design phase, per program and per course. Then find the values of times which are outside the range [max(average - 3*standard_dev, 1), average + 3*standard_dev]	Boolean	Cell	SQL Query
66	Outliers in the time by design review phase	The times recorded for the design review phase are not far than 3 standard deviations from the average	Calculate the average time spent on the design review phase, per program and per course. Then find the values of times which are outside the range [max(average - 3*standard_dev, 1), average + 3*standard_dev]	Boolean	Cell	SQL Query
67	Outliers in the time by code phase	The times recorded for the code phase are not far than 3 standard deviations from the average	Calculate the average time spent on the code phase, per program and per course. Then find the values of times which are outside the range [max(average - 3*standard_dev, 1), average + 3*standard_dev]	Boolean	Cell	SQL Query
68	Outliers in the time by code review phase	The times recorded for the code review phase are not far than 3 standard deviations from the average	Calculate the average time spent on the code review phase, per program and per course. Then find the values of times which are outside the range [max(average - 3*standard_dev, 1), average + 3*standard_dev]	Boolean	Cell	SQL Query

69	Outliers in the time by compile phase	The times recorded for the compile phase are not far than 3 standard deviations from the average	Calculate the average time spent on the compile phase, per program and per course. Then find the values of times which are outside the range [max(average - 3*standard_dev, 0), average + 3*standard_dev]	Boolean	Cell	SQL Query
70	Outliers in the time by unit testing phase	The times recorded for the unit testing phase are not far than 3 standard deviations from the average	Calculate the average time spent on the unit testing phase, per program and per course. Then find the values of times which are outside the range [max(average - 3*standard_dev, 1), average + 3*standard_dev]	Boolean	Cell	SQL Query
71	Outliers in the time by post mortem phase	The times recorded for the post mortem phase are not far than 3 standard deviations from the average	Calculate the average time spent on the post mortem phase, per program and per course. Then find the values of times which are outside the range [max(average - 3*standard_dev, 1), average + 3*standard_dev]	Boolean	Cell	SQL Query
75	Defect removal date vs Phase date time	The date on which a defect was removed must be between the start date and the end date of the phase in which the defect was removed.	Find the records of defect that don't meet: *phase_start_date < defect_removal_date < phase_end_time* , where the phase is the one in which the defect was removed. Only considering defects with correct accuracy	Boolean	Tuple	SQL Query
76	Defects removal time vs planning phase time	The total defects removal time for the planning phase must be lower than the time recorded for that phase	For the planning phase, find records that match that: *sum (defect_removal_time_in_planning) >= time_of_planning*	Boolean	Tuple	SQL Query
77	Defects removal time vs design phase time	The total defects removal time for the design phase must be lower than the time recorded for that phase	For the design phase, find records that match that: *sum (defect_removal_time_in_design) >= time_of_design*	Boolean	Tuple	SQL Query
78	Defects removal time vs design review phase time	The total defects removal time for the design review phase must be lower than the time recorded for that phase	For the design review phase, find records that match that: *sum (defect_removal_time_in_design_review) >= time_of_design_review*	Boolean	Tuple	SQL Query

79	Defects removal time vs code phase time	The total defects removal time for the code phase must be lower than the time recorded for that phase	For the code phase, find records that match that: *sum (defect_removal_time_in_code) >= time_of_code*	Boolean	Tuple	SQL Query
80	Defects removal time vs code review phase time	The total defects removal time for the code review phase must be lower than the time recorded for that phase	For the code review phase, find records that match that: *sum (defect_removal_time_in_code_review) >= time_of_code_review*	Boolean	Tuple	SQL Query
81	Defects removal time vs compile phase time	The total defects removal time for the compile phase must be lower than the time recorded for that phase	For the compile phase, find records that match that: *sum (defect_removal_time_in_compile) >= time_of_compile*	Boolean	Tuple	SQL Query
82	Defects removal time vs unit testing phase time	The total defects removal time for the unit testing phase must be lower than the time recorded for that phase	For the unit testing phase, find records that match that: *sum (defect_removal_time_in_unit_testing) >= time_of_unit_testing*	Boolean	Tuple	SQL Query
83	Defects removal time vs post mortem phase time	The total defects removal time for the post mortem phase must be lower than the time recorded for that phase	For the post mortem phase, find records that match that: *sum (defect_removal_time_in_post_mortem) >= time_of_post_mortem*	Boolean	Tuple	SQL Query
84	Not recorded defects	If the time recorded for the Testing phase is more than 20 minutes, then there must be registered at least one defect that was removed in testing	Find the students that spent more than 20 minutes in the testing phase and do not registered at least one defect removed for that program in the testing phase	Boolean	Tuple	SQL Query
85	Granularity in defect times	The hours, minutes and seconds, corresponding to the time when the defect was registered, are not all equal to 0	Find the defects that meet that "hh:mm:ss" are equal to 0, where the format is "yyyy-mm-dd hh:mm:ss"	Boolean	Cell	SQL Query
86	Correctness in Project Ids	The project id must be normalized so that the same project identificators (numbers) are used for all users, depending on the course type	Find the projects that meet that, sorted by end date, have the correct psp process, but its ids do not correspond with reality	Boolean	Cell	SQL Query and Programming
87	Duplicate students	No repeated students are allowed in the database	Find students that have same name, same tutor and same data	Boolean	Tuple	SQL Query and Programming
88	Invalid reference in rlogddetail table	Discard the rows which fields have invalid values as foreign keys and set the foreign key in database to ensure consistency	Find the rows with invalid fk values and discard them	Boolean	Tuple	SQL Query and Programming

89	Invalid reference in rlogtdetail table	Discard the rows which fields have invalid values as foreign keys and set the foreign key in database to ensure consistency	Find the rows with invalid fk values and discard them	Boolean	Tuple	SQL Query and Programming
90	Invalid reference in rparts table	Discard the rows which fields have invalid values as foreign keys and set the foreign key in database to ensure consistency	Find the rows with invalid fk values and discard them	Boolean	Tuple	SQL Query and Programming
91	Invalid reference in rparttypestandard table	Discard the rows which fields have invalid values as foreign keys and set the foreign key in database to ensure consistency	Find the rows with invalid fk values and discard them	Boolean	Tuple	SQL Query and Programming
92	Invalid reference in rphasedata table	Discard the rows which fields have invalid values as foreign keys and set the foreign key in database to ensure consistency	Find the rows with invalid fk values and discard them	Boolean	Tuple	SQL Query and Programming
93	Invalid reference in rphases table	Discard the rows which fields have invalid values as foreign keys and set the foreign key in database to ensure consistency	Find the rows with invalid fk values and discard them	Boolean	Tuple	SQL Query and Programming
94	Invalid reference in rpips table	Discard the rows which fields have invalid values as foreign keys and set the foreign key in database to ensure consistency	Find the rows with invalid fk values and discard them	Boolean	Tuple	SQL Query and Programming
95	Invalid reference in rprocesses table	Discard the rows which fields have invalid values as foreign keys and set the foreign key in database to ensure consistency	Find the rows with invalid fk values and discard them	Boolean	Tuple	SQL Query and Programming
96	Invalid reference in rprocessphase table	Discard the rows which fields have invalid values as foreign keys and set the foreign key in database to ensure consistency	Find the rows with invalid fk values and discard them	Boolean	Tuple	SQL Query and Programming
97	Invalid reference in rprogramsize table	Discard the rows which fields have invalid values as foreign keys and set the foreign key in database to ensure consistency	Find the rows with invalid fk values and discard them	Boolean	Tuple	SQL Query and Programming
98	Invalid reference in rprojects table	Discard the rows which fields have invalid values as foreign keys and set the foreign key in database to ensure consistency	Find the rows with invalid fk values and discard them	Boolean	Tuple	SQL Query and Programming
99	Invalid reference in rpspassgtdata table	Discard the rows which fields have invalid values as foreign keys and set the foreign key in database to ensure consistency	Find the rows with invalid fk values and discard them	Boolean	Tuple	SQL Query and Programming
100	Invalid reference in rscheduleweeks table	Discard the rows which fields have invalid values as foreign keys and set the foreign key in database to ensure consistency	Find the rows with invalid fk values and discard them	Boolean	Tuple	SQL Query and Programming
101	Invalid reference in rsetadd table	Discard the rows which fields have invalid values as foreign keys and set the foreign key in database to ensure consistency	Find the rows with invalid fk values and discard them	Boolean	Tuple	SQL Query and Programming
102	Invalid reference in rsetbase table	Discard the rows which fields have invalid values as foreign keys and set the foreign key in database to ensure consistency	Find the rows with invalid fk values and discard them	Boolean	Tuple	SQL Query and Programming
103	Invalid reference in rsetreuse table	Discard the rows which fields have invalid values as foreign keys and set the foreign key in database to ensure consistency	Find the rows with invalid fk values and discard them	Boolean	Tuple	SQL Query and Programming

104	Invalid reference in rtasks table	Discard the rows which fields have invalid values as foreign keys and set the foreign key in database to ensure consistency	Find the rows with invalid fk values and discard them	Boolean	Tuple	SQL Query and Programming
105	Invalid reference in rtaskscheduleplans table	Discard the rows which fields have invalid values as foreign keys and set the foreign key in database to ensure consistency	Find the rows with invalid fk values and discard them	Boolean	Tuple	SQL Query and Programming
106	Invalid reference in rtestreports table	Discard the rows which fields have invalid values as foreign keys and set the foreign key in database to ensure consistency	Find the rows with invalid fk values and discard them	Boolean	Tuple	SQL Query and Programming
107	Invalid reference in ruserprofiles table	Discard the rows which fields have invalid values as foreign keys and set the foreign key in database to ensure consistency	Find the rows with invalid fk values and discard them	Boolean	Tuple	SQL Query and Programming

Measurement Results						
#	Name	# Measured tuples	# Erroneous tuples	% Erroneous tuples	#Different users with at least one erroneous tuple	% Users who committed errors
1	Domain integrity in planning phase time	3050	24	0,79%	19	4,66%
2	Domain integrity in design phase time	3050	23	0,75%	19	4,66%
3	Domain integrity in design review phase time	1846	34	1,84%	26	6,37%
4	Domain integrity in code phase time	3050	5	0,16%	4	0,98%
5	Domain integrity in code review phase time	1846	17	0,92%	13	3,19%
6	Domain integrity in compile phase time	3050	0	0,00%	0	0,00%
7	Domain integrity in unit testing phase time	3050	18	0,59%	16	3,92%
8	Domain integrity in post mortem phase time	3050	110	3,61%	73	17,89%
9	Domain integrity in defect removal time	20916	177	0,85%	77	18,87%
10	Domain integrity in reused parts	4679	183	3,91%	108	26,47%
11	Domain integrity in total time	735	5	0,68%	3	0,74%
12	Domain integrity in the planned size (A&M)	194	0	0,00%	0	0,00%
13	Domain integrity in actual total size	194	0	0,00%	0	0,00%
14	Domain integrity in actual added size	194	0	0,00%	0	0,00%
15	Domain integrity in actual A&M size	3050	74	2,43%	61	14,95%
16	Uniqueness in defects records	20916	347	1,66%	30	7,35%
22	Suspicious interrupt time when no interruption occurs	408	100	24,51%	100	24,51%
23	Suspicious interrupt time when big delta occurs	40967	57	0,14%	41	10,05%
24	Injection phase vs. Removal fase	20916	81	0,39%	30	7,35%
25	Domain integrity in Fix Defect	959	149	15,54%	39	9,56%
26	Actual Total Size vs. Size Estimating Template	2448	17	0,69%	8	1,96%

34	Test Report existence	2426	479	19,74%	213	52,21%
35	Parts existence	2448	3	0,12%	2	0,49%
37	Defect log existence	408	0	0,00%	0	0,00%
38	Null value in planning phase time	3050	0	0,00%	0	0,00%
39	Null value in design phase time	3050	0	0,00%	0	0,00%
40	Null value in design review phase time	1846	0	0,00%	0	0,00%
41	Null value in code phase time	3050	0	0,00%	0	0,00%
42	Null value in code review phase time	1846	0	0,00%	0	0,00%
43	Null value in unit testing phase time	3050	0	0,00%	0	0,00%
44	Null value in post mortem phase time	3050	0	0,00%	0	0,00%
45	Null value in the defect description	20916	32	0,15%	17	4,17%
46	Null value in defect removal time	20916	0	0,00%	0	0,00%
47	Null value in fix defect	958	421	43,95%	156	38,24%
48	Null value in problem description	3826	189	4,94%	70	17,16%
49	Null value in improvement description	3826	272	7,11%	98	24,02%
50	Null value in expected results	8191	147	1,79%	83	20,34%
51	Null value in actual results	8191	149	1,82%	83	20,34%
52	Null values in part name	20445	229	1,12%	110	26,96%
53	Null value in added part type	8525	289	3,39%	129	31,62%
54	Null value in plan items of added parts	8525	0	0,00%	0	0,00%
55	Null value in plan relative size of added parts	8525	830	9,74%	224	54,90%
56	Null value in plan size of added parts	8525	0	0,00%	0	0,00%
57	Null value in total time	8525	0	0,00%	0	0,00%
58	Null value in total actual time	3050	0	0,00%	0	0,00%
59	Null value in total planned time	8525	0	0,00%	0	0,00%
60	Null value in the planned size (A&M)	8525	0	0,00%	0	0,00%

61	Null value in actual total size	8525	0	0,00%	0	0,00%
62	Null value in actual added size	8525	0	0,00%	0	0,00%
63	Null value in actual added and modified size	8525	0	0,00%	0	0,00%
64	Outliers in the time by planning phase	3050	87	2,85%	54	13,24%
65	Outliers in the time by design phase	3050	84	2,75%	55	13,48%
66	Outliers in the time by design review phase	3692	150	4,06%	51	12,50%
67	Outliers in the time by code phase	3050	64	2,10%	42	10,29%
68	Outliers in the time by code review phase	3692	104	2,82%	37	9,07%
69	Outliers in the time by compile phase	3050	67	2,20%	46	11,27%
70	Outliers in the time by unit testing phase	3050	95	3,11%	65	15,93%
71	Outliers in the time by post mortem phase	3050	171	5,61%	112	27,45%
75	Defect removal date vs Phase date time	20916	4594	21,96%	355	87,01%
76	Defects removal time vs planning phase time	3050	0	0,00%	0	0,00%
77	Defects removal time vs design phase time	3050	3	0,10%	3	0,74%
78	Defects removal time vs design review phase time	1846	23	1,25%	20	4,90%
79	Defects removal time vs code phase time	3050	2	0,07%	2	0,49%
80	Defects removal time vs code review phase time	1846	37	2,00%	33	8,09%
81	Defects removal time vs compile phase time	3050	303	9,93%	174	42,65%
82	Defects removal time vs unit testing phase time	3050	154	5,05%	104	25,49%
83	Defects removal time vs post mortem phase time	3050	7	0,23%	7	1,72%
84	Not recorded defects	3050	105	3,44%	81	19,85%
85	Granularity in defect times	20916	1512	7,23%	149	36,52%
86	Correctness in Project Ids	439	38	8,66%	38	9,31%
87	Duplicate students	456	8	1,75%	8	1,96%
88	Invalid reference in rlogddetail table	125496	2	0,00%	2	0,49%

89	Invalid reference in rlogtdetail table	122901	0	0,00%	0	0,00%
90	Invalid reference in rparts table	71520	74	0,10%	12	2,94%
91	Invalid reference in rparttypestandard table	6	0	0,00%	0	0,00%
92	Invalid reference in rphasedata table	43984	0	0,00%	0	0,00%
93	Invalid reference in rphases table	15	0	0,00%	0	0,00%
94	Invalid reference in rpips table	7652	0	0,00%	0	0,00%
95	Invalid reference in rprocesses table	14	0	0,00%	0	0,00%
96	Invalid reference in rprocessphase table	92	0	0,00%	0	0,00%
97	Invalid reference in rprogramsize table	12200	1	0,01%	1	0,25%
98	Invalid reference in rprojects table	9150	0	0,00%	0	0,00%
99	Invalid reference in rpspassgtdata table	6100	206	3,38%	206	50,49%
100	Invalid reference in rscheduleweeks table	100404	0	0,00%	0	0,00%
101	Invalid reference in rsetadd table	59675	15	0,03%	1	0,25%
102	Invalid reference in rsetbase table	18335	0	0,00%	0	0,00%
103	Invalid reference in rsetreuse table	23395	5	0,02%	2	0,49%
104	Invalid reference in rtasks table	109960	0	0,00%	0	0,00%
105	Invalid reference in rtaskscheduleplans table	408	0	0,00%	0	0,00%
106	Invalid reference in rtestreports table	16382	9	0,05%	4	0,98%
107	Invalid reference in ruserprofiles table	408	0	0,00%	0	0,00%

Appendix 4

Data Quality Problems in the PSP

This appendix presents a detailed explanation of the data quality problems that were found in the Personal Software Process recollected data.

1. Out of Range Value

There are specific values for which is possible to define the range they should belong to. However, many of the registered data during this process might take values that are not close to one another, and yet correspond to correct events of reality. For instance, the amount of defects that each student detects and records may be one, as well as one-hundred and, any of those might correspond to an incorrect value. This is why the first step is to define the relevant data to then establish a range and measure.

In this regard, two cases are considered. On one hand, the recorded time for each phase and each process. On the other hand, the amount of new parts added by each program and course. It is important to consider, nonetheless, that if a value is out of the determined range it does not necessarily mean that the value is incorrect. The fact that outlier but correct values exist, is part of every learning process. In spite of knowing this, being able to discriminate those outlier (but correct) values from real errors can be complex.

Due to the fact that all the fields involved in the present quality problem are editable (its values are entered in the tool), errors can happen when typing.

A possible source of errors regarding time, is that they might have been gathered incorrectly, either because it was not completely understood how to gather them or because an error took place when they were calculated or recorded.

Because times are recorded using a chronometer included in the tool, the student might forget to start it when beginning the work, or forget to record pauses and interruptions that can take place.

Another possible source of error can be due to the misuse of the tool itself, since double-clicking on any of the time fields (start and end) will automatically insert values for the current date and time. If this were to happen without the student noticing, the correct registry might be overwritten and the calculated times (delta) would not be the real ones.

In the case of new parts, it is possible that entering them was omitted, or that an error occurred when defining them (as a consequence of misunderstanding the process). Both situations might be the cause for entering an inaccurate amount of them.

Because the definition of parts is closely related to the learning process of the PROBE method, not understanding entirely this method could definitely result in another source of error.

Measurement: the granularity is at cell level, because it involves the time fields recorded in each phase as well as the amount of parts (new, base or reused) that are entered. The way to measure this quality problem consists on establishing, for all the cell values involved, if they are within a certain range or not. For each and every case identified (time and amount of parts) the criteria to correctly determine the range to be considered to evaluate the values is established, and the outliers are then identified through SQL queries.

In any of these two cases, the maximum value for the range to consider could be defined a priori. For the minimum value instead, it is certainly known that at least one new part must have been recorded, and that the time can never be less than 1. Therefore, it is necessary to statistically determine an interval considering the medium value and the standard deviation of the recorded values. Those values within the range [maximum (1, media - 3 *standard deviation), media + 3 *standard deviation] will be considered free of error, while those values outside that interval will be considered as to containing errors and therefore analyzed separately. Note that in the case of base or reused parts the minimum could reach 0 (that is, none being entered).

2. Incorrect Project Identifier

Project identifiers must be standardized in a way that these can be used by any user. Identifiers may vary depending on the course they correspond to:

- Identifiers must be in the range from 408 to 414 for the seven assignment course (the current)
- Identifiers must be in the range from 400 to 407 for the eight assignment course

In addition and for all users, there must be a correspondence between the project that is being rendered in reality and the identifier used. If not, it is unfeasible to perform data analysis by project. The known cause of this quality problem is that the tool used for recording defects can both create and delete projects. As a consequence, projects' identifiers do not retain the semantics, hence it isn't possible to identify which projects correspond to those in reality.

Measurement: the granularity is at cell level. According to the established order by date of completion of the course, it seeks to find all projects that are associated to the PSP process but its project identifier does not correspond to reality.

The measurement of this quality problem is to verify if the amount of exercises and the level and order in which they were created is correct. This can be done executing SQL queries and programming. If so, projects' identifiers are renamed as so they are consistent with each other and with reality. Otherwise, these exercises are dismissed given that the student did not apply PSP correctly.

3. Precision in Times

Defects are recorded in the tool with the following date format “yyyy-mm-dd hh:mm:ss". The hours, minutes and seconds corresponding to the time at which the defect was recorded should not all equal 0.

In this quality problem it is desired to measure the level of detail in the recorded times. If indeed they are all 0, then the desired precision is not reached and therefore it is not possible to know the exact time at which a defect was recorded. The possible causes mentioned for the out of range quality problem regarding time, apply to this problem as well.

Another known cause for this quality problem lies in the use of the tool. If the calendar is clicked on selecting the date, when trying to enter the times, the tool will auto fill the time field with the time 00:00:00. If that is not updated, it entails the existence of error in the time value.

Measurement: The granularity is at cell level because involves the values of times in which defects are recorded. The way to measure this quality problem consists on establishing, by SQL queries, whether there are time values where all equal 0.

4. Null Value

It is of interest to know what information was recorded and what was omitted. Knowing what caused that omission and, if possible, determining the value the null number should take, is of interest as well. The existence of null values influences the data analysis conducted, since it becomes necessary to leave those null values aside to obtain statistics. As an example, when calculating the average, a null value in any of the values recorded will affect the outcome if it is not considered as such.

First, it is necessary to identify which fields admit null values and which do not, according to the updated database schema. Then, those fields that admit null values are identified, though in reality the value should actually be different from null (the fact that null values are admitted is an error in the schema of the database). The latter case is the one to measure. It is assumed that the control of the fields declared as non-zero is done correctly by the tool.

On the other hand, it is meaningless to consider certain records in which any of their most significant values are complete, in spite of existing in the base. Albeit measuring the non-existing records (tuples) that should be in the database, having empty records generated in the database is useless. This means that they contain null values either in all their attributes or in those most important. For example, if there is a record of a process improvement proposal but there is no text detailing the proposal itself, the existence of the tuple in the base is insignificant.

The cause for omitting the fields could be any of the following:

- The student omits entering the value (by accidental omission or by being incapable of determining it)
- The student considers that entering the value is not necessary or important (because it is not mandatory to define it, it can remain null).

- Due to an error in handling the data (whether it is the tool or the database) which causes that the value entered by the student is not properly recorded.

Measurement: the granularity is at cell level because those fields that should have a value different than null intervene. Values of the defects' records, the program's size estimation, project planning, process improvement proposals and testing reports, among others, are involved. The way of measurement is to verify if they contain null values by executing SQL queries.

5. Non-existing Records

Among this quality problem, those records that do not exist (tuples) in the database but exist in reality are identified, and therefore their entry was omitted. This means there is a portion of data from reality which is not reflected in the database. Once again, if the total data universe is unavailable, the statistics analysis conducted based on this data sample will not accurately reflect reality, but only a part of it.

We can discriminate two different cases. On one hand, possible errors for which at least one record in the database should exist, but the non-existence cannot be assumed as a quality problem. An example is a non-existing record in the PIP form. Although at least one process improvement proposal should be entered for each program (PSP 0.1 onwards), if this does not happen it cannot be considered as error.

On the other hand, there are errors that are considered data errors, such as the absence of at least one base or added part. It is pointless imagining that it does not exist, in reality, at least one base or added part by each program that should be recorded in the tool. Because of this, its omission is considered a data quality problem.

The reason for the existence of this quality problem is the same that for the null value problem, with the difference that in this particular case non-existing records (no values) in the analyzed database are identified.

Measurement: the granularity is at tuple level, because it involves records that should exist in the database but their entry was omitted. Process improvement projects are included, as well as testing reports, base parts and added and defects. The measurement of this quality problem is to verify whether these records exist or not in the database, by executing SQL queries.

6. Domain Integrity Rules

For some attributes, it is possible to define the domain to which their values should always belong to. In this case, the valid domain for certain values have to always be greater than zero. If invalid values were entered, then an error occurred during the recording (whether by distraction or for not understanding the process).

The main cause this rule is not met is the lack of controls' definition that avoid the entry of negative or equal to zero values, whether it may be at database level or the application's.

Measurement: the granularity is at cell level because it involves recording fields of times, defects, size estimation and project planning, that shall meet the defined rule.

The measurement of this quality problem is to verify if the domain integrity rule defined is met, by executing SQL queries.

7. Intra-relationship Integrity Rules

A set of rules for certain attributes of the same table is defined, which must be met in the database under study. If any of these rules is violated , the data consistency will be affected, hence any analysis conducted from this data will be as well.

The main reason these rules are not met is the lack of definition of the restrictions in the database (as of design). In this case, an error might occur when a student records certain incorrect information of the process, by violating a rule that is not controlled in the base (whether by distraction or by not understanding the process). As an example, it can happen that the injection phase of the defect occurs after its removal. Which, of course, cannot happen in reality.

Measurement: granularity can be at cell or tuple level, depending on the error. It involves times, defects, size estimation and planning projects, both fields and records. The measurement is to verify if the intra-relationship integrity rules defined for the reality under study are followed, by SQL queries.

8. Referential Integrity Rules

A set of rules on certain attributes of different tables is defined, which must be met in the database under study. Once again, not following these rules will affect the data consistency.

The main reason why these rules are not met, is the lack of definition of the restrictions in the database (as of design). As an example, if the total time that took to remove the defects in a certain phase is added up, it can be greater than the time recorded for such phase.

Measurement: granularity is at tuple level because it involves attributes on the recording tables, both of defects and times. The measurement of this quality problem consists of verifying if the referential integrity rules defined for the reality under study are followed, by executing SQL queries.

9. Invalid Reference

In the current analysis, it is necessary to consider the meeting of the rules among different tables' attributes. When doing an instantiation of this in the database under study, it is possible to identify certain references of nonexistent tuples, resulting in invalid references.

The source of this quality problem is an error in the design of the database's schema, because the definition of foreign keys over certain attributes is omitted. Another possible cause could be due to the manual entry of certain identifiers, where an existence's control of such reference is nonexistent.

As an example, if a defect's record refers to a previous defect's correction, the x_defect_id entered by the user should belong to the defect's identifiers the student has

previously entered. Nonetheless, this is a text field the user enters manually, and the tool does not control whether it is a valid identifier.

For this quality problem, all foreign keys of all tables are analyzed in order to verify that there are no invalid references. This might cause an inconsistency that affects any data analysis and consider requiring project data, which in most cases are of great importance.

Measurement: granularity is at tuple level because it involves those tuples that refer to nonexistent project identifiers. The measurement of this quality problem is to verify if there are tuples with invalid references, by executing SQL queries.

10.Duplicate Register

This quality problem is identified when two or more records are an exact copy of each other. Two situations exist:

- When the value in the key and other attributes is the same (or in any case, null values).
- Despite having different primary key, they refer to the same object in reality and have the same data in the defined fields (according to the duplication criteria considered).

Although the controls of the tool used prevent the existence of duplicate records with the same primary keys, all necessary checks are performed to verify that there are no repeated records in the base under study (according to the criteria to be defined).

Within this quality problem two cases are considered:

- Defects' duplication. Correspond to defects recorded at the exact same time. They are not considered in this case times 00:00:00
- Student's duplication. Correspond to students that have the same name, author and other data.

The cause for this quality problem can be due to a mistake on the student's behalf (e.g. by recording the same defect more than once) or to an error in the tool that leads to repeated records stored in the base (e.g. entering the same student more than once). It is important to consider this quality problem, because if not, results obtained from the data analysis performed would be mistaken. For example, if records of defects are duplicate, i.e. referring to the same defect in reality, the total number of defects will not be real (the number of records will be above the real number).

Measurement: granularity is at tuple level, because it involves those tuples (defect's records) that are duplicate. The measurement of this quality problem is to verify whether duplicate tuples exist, according to the defined duplication criteria, by executing SQL queries.

Appendix 5

Extended Abstract presented for the TSP Symposium 2013

Another Experiment on the Impact of the PSP on Software Quality: Trying to eliminate the programming learning effect

Diego Vallespir, Fernanda Grazioli, Leticia Pérez, Silvana Moreno
Universidad de la República
Montevideo, Uruguay
{dvallesp, grazioli, lperez, smoreno}@fing.edu.uy

Extended abstract

Data collected in the Personal Software Process (PSP) courses indicate that the PSP improves the quality of the products developed [1, 2]. 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 of the ways of knowing if the PSP produces improvements in the quality of the software is by doing a statistical analysis of the evolution of the results obtained by the students in each program. If the programs developed during the course by the students are of a better quality as the course progresses (for example, less defects in UT), then it can be inferred that the PSP is responsible for the improvement.

However, since the programs of the course are in the same application domain, the improvement could be due to programming repetition (learning effect). Recently, a study that compares the data obtained from different versions of the PSP courses (in which the practice of the PSP is introduced at different moments as the exercises advance) concludes that the changes in quality are most plausible regarding mastering PSP techniques rather than program repetition.

Our work aims at contributing in this same sense but using a different approach. Consequently, our research question is: Are the improvements observed in the PSP courses due to the introduction of the phases and techniques of the PSP or due to the program repetition? For knowing this we designed and performed a controlled experiment. 12 Software Engineering undergraduate students of the Universidad de la República, which are the subjects of our experiment, performed the same exercises of the PSP for engineers I/II course in its version of 8 exercises but without applying the PSP techniques.

For the first program the students use the PSP0 and for the seven remaining programs, the PSP0.1. 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 quality of the software products due to program repetition.

Therefore, we define the goal of this work as:

Analyze and compare the data collected at eight program assignments
for the purpose of evaluating software quality improvements
with respect to ***defect density in unit testing / total defect density***
from the viewpoint of a researcher
in the context of the PSP0.1 level training of 12 software engineers undergraduate students.

In order to know the quality of the products we base on two measures that are normally used in the experiments that involve the PSP: defect density in unit test and total defect density of the program (dependent variables of the experiment). We compare programs by pairs to find if the changes in each program are statistically significant. Therefore, we propose a null hypothesis and an alternative hypothesis for each dependent variable studied:

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)

H0 tot def: Median (Total defect density i) = Median (Total defect density j)
H1 tot def: Median (Total defect density i) <> Median (Total defect density j)

Where i, j are the numbers of the programs (1 to 8) and i < j

Results

Table 1 presents median and interquartile range of defect density in unit test for the programs from 2 to 8. A difference between the median of program 2 and the rest of the programs is perceived.

Table 1 – Median and interquartile range for DDUT

	Pr 2	**Pr 3**	**Pr 4**	**Pr 5**	**Pr 6**	**Pr 7**	**Pr 8**
Median	56.98	18.13	18.48	36.38	18.40	13.78	8.59
IQR	21.20	31.84	18.14	30.19	17.11	25.11	12.20

The students of our experiment are 12 (few samples) and the data of each one in the 8 exercises of the PSP (repeated measures) are considered. In a context of few samples and repeated measures the most suitable statistical test for our hypotheses is Wilcoxon signed-ranks [4]. This test is used to compare two sets of scores that come from the same participants and when normality cannot be assumed. We used the 2-tailed

Wilcoxon test because we do not know a priori if the dependent variable will increase or reduce its value.

Table 2 presents the result of applying the Wilcoxon test to each pair of programs for the hypothesis of defect density in UT. 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 indicate that the null hypothesis has been rejected (p<=0.05) and that there has been an improvement in defect density in UT as the students advance in the exercises. 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 are two motives that can explain this behavior. The first is based on the fact that 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. The second possibility is that the systematization in the recording of defects by the students (from the PSP0) has its positive effects and produces fewer injected defects.

Table 2 – Wilcoxon test for DDUT

Prog.	**3**	**4**	**5**	**6**	**7**	**8**
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.

Since the experiment does not change the level of PSP used (PSP0.1) the results of this experiment indicate that the repetition of programs in the same application domain and the collection of data of the processes do not improve defect density in UT by themselves.

Conclusions

The presented results contribute to the elimination of an important threat to the validity of different experiments performed with the PSP. This result, together with a previous one [3], indicates that the practices introduced by the PSP and not program repetition would contribute to the improvement of software quality.

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.

Note: We will have analyzed other variables for the TSP Symposium which we have not been able to analyze now for time constraints: total defect density, time spent in UT per Kloc and average time in UT per defect.

References

[1] 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.

[2] 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.

[3] 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.

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

Appendix 6

Publications

During the thesis, two articles were published. The first one was accepted for the proceedings of the TSP Symposium 2012 and included in a SEI Special Report. The other was accepted and presented in the IX Jornadas Iberoamericanas de Ingeniería de Software e Ingeniería del Conocimiento (Iberoamerican Conference in Software Engineering and Knowledge Engineering), 2012. This appendix contains both articles.

- ***A Cross Course Analysis of Product Quality Improvement with PSP***
  Fernanda Grazioli, William Nichols.
  Proceedings TSP Symposium 2012: Delivering agility with discipline (Special Report Software Engineering Institute, Carnegie Mellon University, CMU/SEI-2012-SR-015), pp.76—89, Saint Petersburg, Florida, EEUU, September 2012.

- ***Un Estudio de la Calidad de los Datos Recolectados durante el Uso del Personal Software Process (An Study of the Quality of the Data Collected During the Use of the Personal Software Process)***
  Carolina Valverde, Fernanda Grazioli, Diego Vallespir
  IX Jornadas Iberoamericanas de Ingeniería de Software e Ingeniería del Conocimiento (JIISIC), pp. 37—44, Lima, Perú, Noviembre de 2012.

7 A Cross Course Analysis of Product Quality Improvement with PSP

Fernanda Grazioli, Universidad de la República
William Nichols

7.1 Introduction and Background

These days, more and more businesses develop, combine, and include software in their products in different ways. Companies need to develop software to support the design, manufacture, or delivery of the products and services they provide. Therefore, although they might not realize it, all businesses are becoming software businesses. As the software component of their business grows, schedule delays, cost overruns, and quality problems caused by software become their main business problems. This is why, despite their best management efforts, companies find their risk of failure increasing along with the increase in the size or complexity of the software they produce.

Software products are made of hundreds to millions of lines of code, each one handcrafted by a software engineer. Software businesses depend on people, so their technical practices and experience strongly influence the outcome of the development process.

The Personal Software Process (PSP) is a defined and measured software process designed to be used by an individual software engineer. The PSP directly addresses software business needs by improving the technical practices and individual abilities of software engineers, and by providing a quantitative basis for managing the development process. By improving individual performance, PSP can improve the performance of the organization.

For many years, the Software Engineering Institute (SEI) has trained software engineers in PSP. Over that period, the course format has changed twice. Several versions of the course use the same exercises, but introduce process steps in modified sequences. An earlier version of the course has several published studies demonstrating improvement in developer performance with process insertion, but the retrospective analysis left some threats to external validity [Paulk 2006; Hayes 1997; Wohlin 1998; Rombach 2008; Kemerer 2009; Paulk 2010]. One threat is the confounding of process insertion with the gaining of domain experience as related programs are developed. A related threat is that observations might alter the subject performance as in the Hawthorne effect [Mayo 1949]. Moreover, there are not yet studies about how the latest two course versions are working, nor are there studies about how different approaches to introducing process correlate with performance and quality results in PSP courses. The PSP community and the SEI need to know how effectively these courses work. The academic and industrial communities need assurance that the process can be taught effectively and that the process insertion would have positive and substantial benefits.

Given this situation, the objective of this study is to use the PSP data from the latest two course formats to determine whether reviews and design improve product quality, or if such improvement is only a consequence of gaining experience in the problem domain. We measure quality as the quantity of defects found per KLOC in Unit Testing. 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. The consequence of high defect density in software engineering is typically seen in the form of bug-fixing or rework effort incurred on projects. Typical defect densities of delivered products range from one to five defects/KLOC [Davis 2003].

7.1.1 Concept Introduction on PSP Courses

The PSP courses incorporate what has been called a "self-convincing" learning strategy that uses data from the engineer's own performance to improve learning and motivate use. The last two course versions introduce the PSP practices in steps corresponding to six PSP process levels. The older version name is "PSP for Engineers I/II (PSPI/II)" and the latest version name is "PSP Fundamentals and Advanced (PSP Fund/Adv)."

Each level builds on the developed capabilities and historical data gathered in the previous level. Engineers learn to use the PSP by writing seven or eight programs (depending on the course version), and by preparing written reports. Engineers may use any design method or programming language in which they are fluent. The programs typically contain around one hundred lines of code (LOC) and require a few hours on average to be completed. While writing the programs, engineers gather process data that are summarized and analyzed during a postmortem phase. There are three basic measures in the PSP: development time, defects, and size. All other PSP measures are derived from these three basic measures.

During the course, the students were given eight or seven exercises (eight in PSPI/II and seven in PSP Fund/Adv), which were mainly programs for statistical calculations. PSP has a maturity framework that shows its progression on improvement phases, also called levels. Students completed their exercises while following the process attained at each PSP level.

The PSP levels introduce the following set of practices incrementally:

- PSP0: Description of the current software process, basic collection of time and defect data
- PSP0.1: Definition of a coding standard, basic technique to measure size, basic technique to collect process improvement proposals
- PSP1: Techniques to estimate size and effort, documentation of test results
- PSP1.1: Task planning and schedule planning
- PSP2: Techniques to review code and design
- PSP2.1: Introduction of design templates

Table 16 shows which PSP level is applied on each program assignment, for each course version.

Table 16: 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

Program Assignment	PSP Fund/Adv	PSP I/II
6	PSP 2.1	PSP 2.1
7	PSP 2.1	PSP 2.1
8	----	PSP 2.1

7.2 Data Set and Statistical Model

We used data from the eight-program course version, PSPI/II, taught between June 2006 and June 2010. Additionally, we used data from the seven-program course version, of PSP Fund/Adv, taught between December 2007 and September 2010. These courses were taught by the SEI at Carnegie Mellon University or by SEI partners, by a number of different instructors in multiple countries.

We began with 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 clean and remove errors and questionable data.

To determine the cuts on the data set, we first developed an integrated data storage model. We designed that model to support the analysis and the assessment of data quality, based on the data quality theory. Data quality is an investigation area that has generated a great workload in the last years and it is mainly focused on defining the aspects of data quality [Batini 2006; Lee 2002; Neely 2005; Strong 1997] and on proposing techniques, methods and methodologies to the measurement and treatment of the data quality [Batini 2006; Lee 2002;Wang 1995].

The first step toward identifying quality problems was to understand the reality and context to be analyzed. This includes the Personal Software Process in itself [Humphrey 1995], exploring the tool for recording data and the model of the database, in addition to the Grading Checklists used by instructors for the correction of exercises performed during the course. Afterwards, we analyzed the dimensions and quality factors proposed by Batini and Scannapieco [Batini 2006] to this set of data, which are interesting to measure and consider. In this way, we thoroughly identified and defined possible quality problems that the data under study might contain, we implemented the algorithms required for cleaning and collecting the metadata, and finally, we executed those algorithms. Major data quality problems were related to the consistency, accuracy, completeness, and uniqueness dimensions. This meant that following that data quality process, our data set was reduced to 93 subjects in total, 45 from the PSP Fund/Adv course and 47 from the PSPI/II course.

Differences in performance between engineers are typically the greatest source of variability in software engineering research, and this study is no exception. However, the design of the PSP training class and the standardization of each engineer's measurement practice allow the use of statistical models that are well suited for dealing with the variation among engineers.

In the summarized analyses presented, we studied the changes in engineers' data over seven programming assignments. Rather than analyzing changes in group averages, this study focuses on the average changes of individual engineers. Some engineers performed better than others from the first assignment, and some improved faster than others during the training course. To

discover the pattern of improvement in the presence of these natural differences between engineers, the statistical method known as the repeated measures analysis of variance (ANOVA) is used [Tabachnick 1989]. In brief, the repeated measures analysis of variance takes advantage of situations where the same people are measured over a succession of trials. By treating previous trials as baselines, the differences in measures across trials (rather than the measures themselves) are analyzed to uncover trends across the data. This allows for differences among baselines to be factored out of the analysis. In addition, the different rates of improvement between people can be viewed more clearly. If the majority of people change substantially (relative to their own baselines), the statistical test will reveal this pattern. If only a few people improve in performance, the statistical test is not likely to suggest a statistically significant difference, no matter how large the improvement of these few people.

7.3 Analysis and Results

First, we define the following variables and terms:

- Subject –student who performs a complete PSP course.
- Course Type –a PSP course version. It can be PSP Fund/Adv or PSPI/II.
- Program Assignment or Program Number –an exercise that a student has performed during the PSP course. It can be 1, 2, 3, 4, 5, 6, 7 or 8.
- PSP Level –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. Since we wanted to analyze the introduction of concepts 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. So the PSP Level variable can be seen in plots as 0, 1, 2, or 2.1, respectively.
- Defect Density in Unit Testing (DDUT) – 1000 · Total defects removed in testing / Actual added and modified LOC

As we stated in the introduction, the objective of this study is to use the PSP data from the latest two course formats to demonstrate whether reviews and design improve the quality of a product in test. And we use defect density as a measure of quality, so we define defect density in unit testing as the independent variable in our analyses. Figure 34 shows a bar-whisker chart of DDUT grouped by course type and PSP level. Figure 35 shows a bar-whisker chart of DDUT, but in this case grouped by course type and program assignment. These charts are descriptive and allow us to get a clearer idea of the defect density behavior.

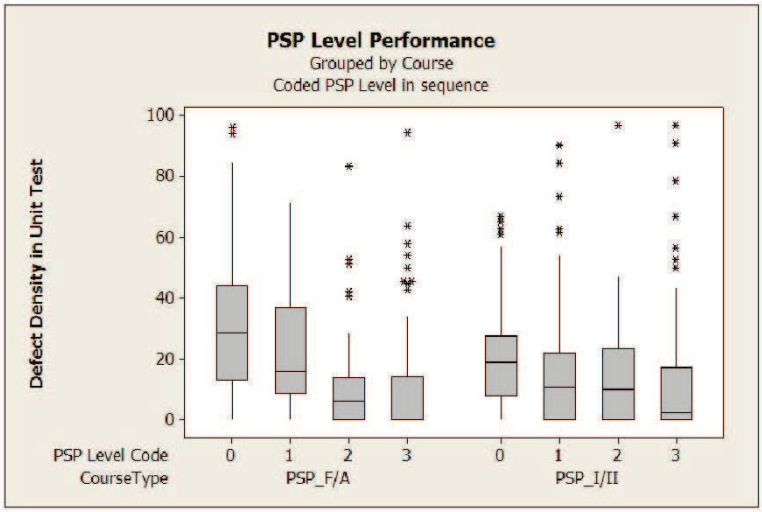


Figure 34: Defect Density in Unit Testing grouped by Course Type and PSP Level

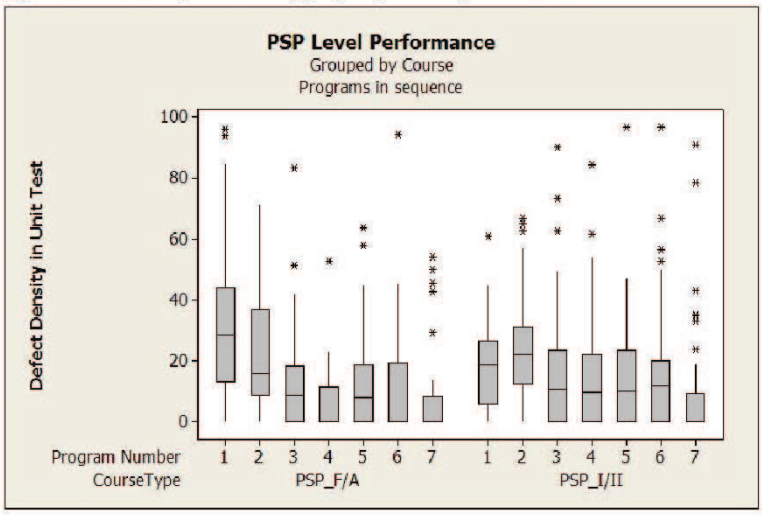


Figure 35: Defect Density in Unit Testing Grouped by Course Type and Program Assignment

To reach that objective, we considered a direct way through an ANCOVA analysis, using PSPI/II and PSP Fund/Adv as categorical values to segment the data into two parts: the program assignment as the independent variable and the PSP level as the hidden variable. But we could not

come to a conclusion because the correlation between the PSP level and the program assignment was strong. Also, using ordinal program numbers in ANCOVA may not be a sound approach. Correlations between the factors are a strong anti-indication for ANCOVA [Tabachnick 1989]. Therefore, we decided to create a more indirect procedure and analyze the results based on specific differences between the two courses using the PSP level.

We next developed an indirect procedure to examine relationships between program number, PSP level, and performance in the data. It consists of three steps, each one based on an ANOVA analysis using defect density in unit testing as the independent variable.

Before running any ANOVA, we checked to see if the data satisfy all assumptions for the correct use. The only assumption that could not be fully satisfied was the normality assumption. We transformed the data into normal form using different techniques as suggested in [Tabachnick 1989], and the one that worked better was a log transformation and adding a constant to the zero observations. Nonetheless, some deviations from normality persisted at very low defect levels. Recent works recommend not to log-transform count data [O'Hara 2010] for two reasons. First, low counts cause distortions. Second, such transformation typically does not affect the results if the data is unimodal. The data satisfies the unimodal mounded condition, and the fact that transformed and untransformed PSP data provides comparable results has been previously observed with [Hayes 1997]. We did our analysis considering both transformed and untransformed data.

The first step of the analysis procedure consisted of performing a series of one-way ANOVA between each program number using course as the grouping factor. This establishes whether the assignments have different DDUT means between the courses. If there are not different means between courses, then the analysis is done because the program-by-program results for the two courses do not differ. We ran seven tests, one for each program number and, since we found significant differences, we proceeded to the next step.

The second step of the analysis procedure consisted of performing two-way (repeated) ANOVA. Separate, repeated ANOVA analyses grouping by program number were performed for each course type. We applied the two-tailed significance test at 0.05, which is equivalent to a 0.025 significance level for a one-tailed test. Table 17 and Table 18 summarize the ANOVA discussed above for this step for the courses PSP Fund/Adv and PSP I/II, respectively. A third two-way ANOVA grouping by course and program number was performed on the combined data, and the results are summarized in Table 19.

Both the separate course data and the combined data showed a general downward trend in defect level with program number, irrespective of process level. This was as we expected based on the correlation between PSP level and program number. We found no statistical significance between consecutive programs for either the PSP Fund/Adv or the PSP I/II. However, for the combined data set, we found significant reduction in defect levels between programs 2 and 3, and 3, and 4. The significance in the combined set results from the increased sample size.

In the PSP Fund/Adv course we found that there is significance between Program 1 and Programs 3, 4, 5, 6, and 7. We interpret that this shows improvements between PSP0 and PSP2 or PSP2.1 (depending on which program from 3 to 7). In regard to PSP I/II, we found that there is

significance between Program 1 and Program 7, and this is consistent with improvements between PSP0 and PSP2.1.

We also found that there is significance in PSP Fund/Adv between Program 2 and Program 3, 4, 5, 6, and 7. This makes sense because it shows improvement between PSP1 and PSP2 or PSP2.1 (depending on which program from 3 to 7). In regard to PSP I/II, we found that there is significance between Program 2 and Program 4, 5, 6, 7. This also makes sense because it shows improvement between PSP0 and PSP1 or PSP2 or PSP2.1 (also depending on which program from 4 to 7).

Table 17: ANOVA Outputs for Program Assignment Comparison in PSP Fund/Adv

PSP Fund/Adv				
Program Assignment (I)	**Program Assignment (J)**	**PSP Level**	**Mean difference (I-J)**	**Sig.**
1	2	1	*,154*	*,999*
	3	2	*1,364*	*,003*
	4	2	*2,348*	*,000*
	5	2.1	*1,602*	*,000*
	6	2.1	*2,139*	*,000*
	7	2.1	*2,347*	*,000*
2	3	2	*1,210*	*,012*
	4	2	*2,193*	*,000*
	5	2.1	*1,448*	*,001*
	6	2.1	*1,985*	*,000*
	7	2.1	*2,192*	*,000*
3	4	2	*,983*	*,082*
	5	2.1	*,237*	*,994*
	6	2.1	*,774*	*,301*
	7	2.1	*,982*	*,082*
4	5	2.1	*-,745*	*,347*
	6	2.1	*-,208*	*,997*
	7	2.1	*-,001*	*1,000*
5	6	2.1	*,537*	*,731*
	7	2.1	*,744*	*,349*
6	7	2.1	*,207*	*,997*

Table 18: ANOVA Outputs for Program Assignment Comparison in PSP I/II

PSP I/II				
Program Assignment (I)	**Program Assignment (J)**	**PSP Level**	**Mean difference (I-J)**	**Sig.**
1	2	0.1	*-,485*	*,820*
	3	1	*,502*	*,795*
	4	1.1	*,730*	*,381*
	5	2	*,816*	*,248*
	6	2.1	*,948*	*,109*
	7	2.1	*1,517*	*,001*
2	3	1	*,987*	*,083*
	4	1.1	*1,216*	*,012*
	5	2	*1,301*	*,005*
	6	2.1	*1,434*	*,001*

PSP I/II				
Program Assignment (I)	Program Assignment (J)	PSP Level	Mean difference (I-J)	Sig.
	7	2.1	*2,003*	*,000*
3	4	1.1	*,228*	*,995*
	5	2	*,314*	*,975*
	6	2.1	*,446*	*,871*
	7	2.1	*1,015*	*,067*
4	5	2	*,0854*	*1,000*
	6	2.1	*,2179*	*,996*
	7	2.1	*,786*	*,290*
5	6	2.1	*,132*	*1,000*
	7	2.1	*,701*	*,433*
6	7	2.1	*,569*	*,682*

Table 19 shows a summary of the ANOVA for the combined data.

Table 19: ANOVA Outputs for Program Assignment Comparison Combined Course Data

Program Assignment (I)	Program Assignment (J)	Mean difference (I-J)	Sig.
1	2	*-,166*	*,509*
	3	*,933*	*,000*
	4	*1,539*	*,000*
	5	*1,209*	*,000*
	6	*1,544*	*,000*
	7	*1,932*	*,000*
2	3	*1,099*	*,000*
	4	*1,705*	*,000*
	5	*1,375*	*,000*
	6	*1,710*	*,000*
	7	*2,098*	*,000*
3	4	*,606*	*,016*
	5	*,276*	*,271*
	6	*,611*	*,015*
	7	*,999*	*,000*
4	5	*-,330*	*,188*
	6	*,005*	*,985*
	7	*,393*	*,117*
5	6	*,335*	*,182*
	7	*,723*	*,004*
6	7	*,388*	*,122*

As a summary of this step, we can say that for each course we found significant difference only between assignments with different PSP levels. For the combined course we found a significant difference between programs 2 and 3 (introduced in PSP level 2 in PSP Fundamentals) and programs 3 and 4 (PSP the second use of level 2 in Fundamentals and PSP level 1.0 to 1.1 estimation by parts, in PSP I).

According to the design and review techniques introduced in the corresponding PSP levels, we expected these improvements. Figure 36 shows the estimated marginal means of the log-

transformation of defect density in unit testing versus program number, for both courses. The graphic shows how the two courses perform differently. The declining defect level is more consistent and larger in PSP Fundamentals through the introduction of PSP 2.0. Defect levels appear to be more consistent by the end of the courses.

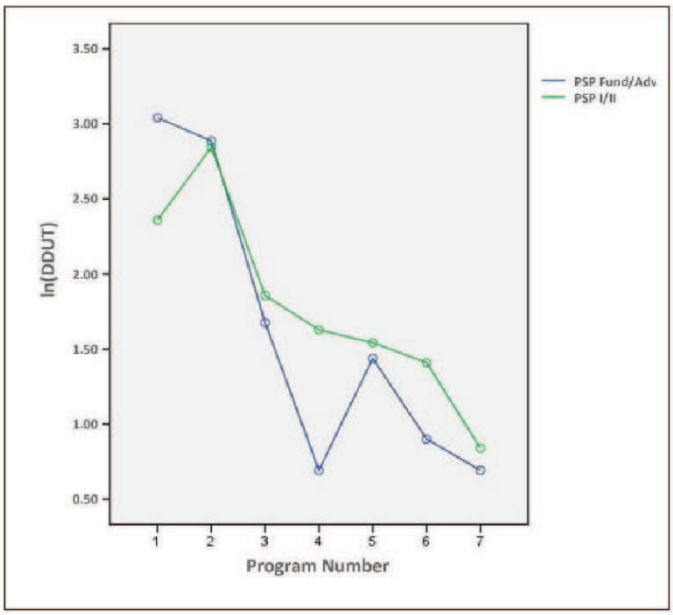


Figure 36: Comparison of Estimated Marginal Means of Ln(DDUT) versus Program Number between PSP Fund/Adv and PSP I/II

The third and last step of the analysis procedure consists of performing a two-way ANOVA grouping by PSP level and course to set bounds on the importance of PSP level as a predictor.

After this test, we found that there is significant difference between PSP0 and PSP1, between PSP0 and PSP2, and also between PSP0 and PSP2.1. Looking at it in more detail, results show that

- PSP1 was a factor of 0.2 more effective than PSP0 at an alpha level of 0.05 with a confidence range of the differences of [0.074, 0.939].
- PSP2 was a factor of 0.4 more effective than PSP0 at an alpha level of 0.05 with a confidence range of the differences of [1.028, 1.888].
- PSP2.1 was a factor of 0.45 more effective than PSP0 at an alpha level of 0.05 with a confidence range of the differences of [1.373, 2.133].

We also found that there is significant difference between PSP1 and PSP2, and between PSP1 and PSP2.1. Looking at it in more detail, results show that

- PSP2 was a factor of 0.22 more effective than PSP1 at an alpha level of 0.05 with a confidence range of the differences of [0.521, 1.381].
- PSP2.1 was a factor of 0.28 more effective than PSP1 at an alpha level of 0.05 with a confidence range of the differences of [0.866, 1.626].

Figure 37 shows the 95% confidence intervals of the log-transformation of defect density in unit testing for each PSP level, for both courses.

As a summary of this step, we can say that in both courses there is significant difference between all PSP levels, except between PSP2 and PSP2.1. The lack of significance between PSP2 and PSP2.1 may be because 1) the difference is too small to resolve (power of the sample), 2) the log-transformation hides the results (transforming those zeros will reduce the effect), or 3) there is no difference.

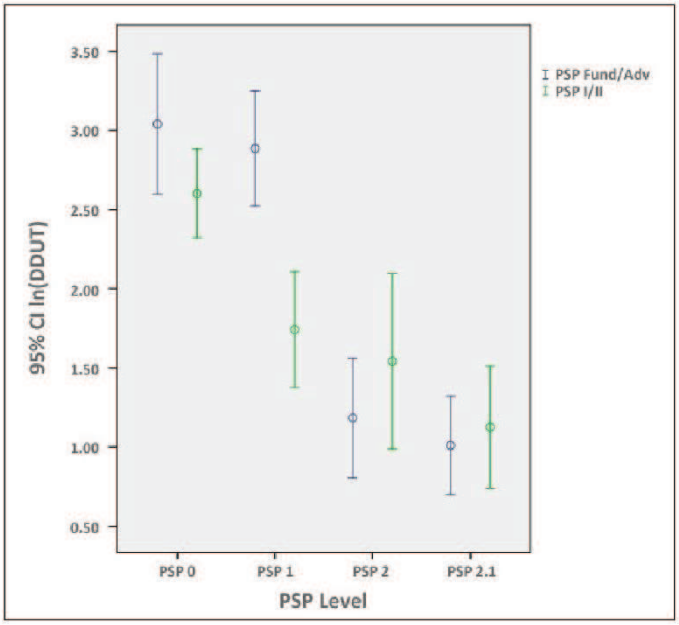


Figure 37: 95% Confidence Interval of Ln(DDUT) for each PSP Level in PSP Fund/Adv and PSP I/II

7.4 Threats to Validity and Limitations

By definition, defect density depends on the number of defects removed. But the number of defects found and removed in the test phase depends on the student's experience and on how good

the student is in doing unit testing. Therefore, we have a threat related to testing because the tests —those that are coincident with the treatment—may influence the student behavior.

According to the history, these courses were taught largely, but not entirely, at different times. Newer development environments and changes in the computer language instruction may alter subject behavior or the defect injection profile.

For a correct application of ANOVA, there is an assumption that the subjects are randomly selected for the treatments. We did not select the students; they were the ones that selected the course, and there is no precondition to do one course or another. So the random selection seems to be satisfied. But on the other hand, the students who took the PSP Advanced are more likely to go on to instruction or teaching. So, this group might respond better to the PSP instruction, and this could be seen as a threat to validity.

Even after transformation of the data, the normality assumption for ANOVA could not be satisfied. The distributions of defect density tend to be positively skewed, with long tails extending to the right and a truncated range at zero. This type of non-normal distribution is to be expected given the source of the data. There can never be a negative count for defects, so the truncation at zero is expected. In addition, we would expect many small values of defect density and relatively fewer large values. This positively skewed distribution is expected particularly when engineers (as a group) reduce the defect density of the programs as they improve their quality during the course. This is the type of data where either a logarithmic or inverse transformation can be used to create a more nearly normal distribution [Tabachnick 1989]. Based on our examination of the effects of these two types of transformations on the distribution of residuals, the logarithmic transformation was used in the confirmatory analysis.

We are comparing program assignments of two course versions as if they were identical. This can be considered a threat to validity. While the program assignments are very similar between the courses with the same programming exercise, they differ in the process elements and process used, so they are not exactly the same in both courses.

7.5 Conclusions

Previous studies of Personal Software Process [Hayes 1997, Rombach 2008] have examined the effect of the PSP on the performance of software engineers. The improvements, including product quality, were found to be statistically significant, and the observed results were considered generalizable beyond the involved participants. Those studies only considered students of the first version of the PSP course, which uses 10 program assignments and where there is a strong correlation between the program assignment and PSP level. Those studies may have some threats to external validity. In this work we try to face the generalization threat and consider the latest two course versions to see how different process introduction approaches correlate with quality results in PSP courses.

In this analysis we considered the work of 93 software engineers, who during PSP work developed eight or seven programs, depending on the course version. Each subject took the complete PSP course, either PSP for Engineers I and II or PSP Fundamentals and Advanced. We analyzed the data collected by each student to see how review and design improve the quality of a product in test.

Both courses appear to be effective in demonstrating use of design and reviews and both show reduction in defect injections. Levels achieved at end of the course are consistent with best-in-class practice. This cross-course comparison allowed us to discover that a "Hawthorne effect" is not as plausible as "gaining experience in the problem domain" or PSP techniques associated with PSP level as a causal explanation for the improvements. The strong association with PSP level suggests that learning effects are most plausible regarding mastering PSP techniques rather than general domain knowledge. This might be further examined in a future study with an analysis of phase injection and removal.

Because PSP level changes so rapidly in the PSP Fundamentals and PSP I, program number and PSP process level are tightly correlated in a way that makes separating the effects difficult. These results cannot ensure that the observed improvements are exclusively due to mastering the process techniques introduced in the PSP. We propose future analysis to obtain more generalizable results. The first approach would be a control experiment, consisting of introducing students to PSP, then remaining at PSP 1.0 through the first seven program assignments used in PSP I/II and PSP Fundamentals/Advanced. In this way, we can see how domain learning affects software quality improvement and then compare that result with the results of this study. A second approach would be an extended PSP course with at least three exercises at each PSP level. We believe that this would permit the process changes to stabilize so that we could more directly examine improvements between programs with and without process change.

In future analysis of these data from PSP Fundamentals/Advanced and PSP I/II, we will examine improvements in other dimensions, such as size estimation, effort estimation, defect yield, and productivity, to determine how effectively these courses work in those dimensions.

7.6 Acknowledgments

We thank Jim McCurley of SEI, Gabriela Mathieu, and Diego Vallespir of Universidad de la República for discussions of ANOVA analysis and suggestions of different statistical methods. We also thank the reviewers for their valuable contributions.

7.7 Author Biographies

Fernanda Grazioli
Graduate Student
Universidad de la República

Fernanda Grazioli is a graduate student at the Engineering School at the Universidad de la República, an honorary collaborator of the Software Engineering Research Group (GRIS), and a member of the Software and System Process Improvement Network in Uruguay (SPIN Uruguay).

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) Program. 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 publication topics 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.

[Batini 2006]
Batini, C. & Scannapieco, M. *Data Quality: Concepts, Methodologies and Techniques*. Springer-Verlag, 2006.

[Davis 2003]
Davis, B. N. & Mullaney, J. L. *The Team Software Process (TSP) in Practice: A Summary of Recent Results* (CMU/SEI-2003-TR-014), Software Engineering Institute, 2003. http://www.sei.cmu.edu/library/abstracts/reports/03tr014.cfm

[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://www.sei.cmu.edu/library/abstracts/reports/97tr001.cfm

[Humphrey 1995]
Humphrey, Watts S. *A Discipline for Software Engineering*. Addison-Wesley, 1995. http://www.sei.cmu.edu/library/abstracts/books/0201546108.cfm

[Kemerer 2009]
Kemerer, Chris & Paulk, Mark. "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.

[Lee 2002]
Lee, Y. W.; Strong, D. M.; Kahn, B. K.; & Wang, R. Y. "AIMQ: A Methodology for Information Quality Assessment." *Information & Management, 40, 2* (December 2002): 133-146.

[Mayo 1949]
Mayo, E. *Hawthorne and the Western Electric Company*. "The Social Problems of an Industrial Civilization." Routledge, 1949.

[Neely 2005]
Neely, M. P. "The Product Approach to Data Quality and Fitness for Use: A Framework for Analysis," 221–236. *Proceedings of the 10th International Conference on Information Quality* Boston, MA, November 2005. MIT Press, 2005.

[O'Hara 2010]
O'Hara, R. B.; & Kotze, D. J. "Do not log-transform count data." *Methods in Ecology and Evolution 1* (2010): 118–122.

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

[Paulk 2006]
Paulk, Mark C. "Factors Affecting Personal Software Quality." *CrossTalk: The Journal of Defense Software Engineering 19*, 3 (March 2006): 9–13.

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

[Strong 1997]
Strong, D. M.; Lee, Y. W.; & Wang, R. Y. "Data Quality in Context," *Communications of the ACM 40*, 5 (1997): 103–110.

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

[Wang 1995]
Wang, R. Y.; Reddy, M. P.; & Kon, H. B. "Toward Quality Data: An Attribute-Based Approach." *Decision Support Systems13,* 3–4 (March 1995):349-372.

[Wohlin 1998]
Wohlin, C. & Wesslen, A. "Understanding software defect detection in the Personal Software Process," 49–58. *Proceedings of the Ninth International Symposium on Software Reliability Engineering*, Paderborn, Germany, November 1998. IEEE, 1998.

Un Estudio de la Calidad de los Datos Recolectados durante el Uso del Personal Software Process

Carolina Valverde, Fernanda Grazioli, Diego Vallespir
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
{mvalverde, grazioli, dvallesp}@fing.edu.uy

Resumen

Al usar un proceso de desarrollo de software, los individuos y los equipos generan datos acerca de su uso. Estos datos son fundamentales para el seguimiento de los proyectos de desarrollo de software. Es necesario contar con datos de calidad para tomar las decisiones acertadas durante un proyecto. Sin embargo, muchas veces estos datos no cuentan con la calidad necesaria. Este artículo presenta un estudio de la calidad de los datos del uso del Personal Software Process (PSP). Identificamos 10 posibles problemas de calidad y definimos 91 métricas para medirlos. Encontramos que un 1,34 % del total de los objetos (datos) medidos tiene algún error. Conocer y limpiar los datos de mala calidad ayudan a prevenir la toma de decisiones inadecuadas durante un proyecto de desarrollo de software.

1. Introducción

Los proyectos de desarrollo de productos de software utilizan (generalmente) procesos de desarrollo de software. Se busca mediante el uso de estos procesos construir productos de calidad, dentro del plazo y los costos establecidos. Durante el uso de un proceso los individuos y equipos generan datos acerca de su uso. Por ejemplo, registros de esfuerzo (tiempo empleado en cada fase), registros de fallas (y/o defectos) y registro de versiones. Normalmente, existen herramientas que dan soporte a los procesos para, entre otras cosas, simplificar el registro de los datos mencionados [13].

Los datos que se registran del uso del proceso son fundamentales para el seguimiento del proyecto. El análisis de los mismos influye directamente en la toma de decisiones. Sobre estos datos, en particular en los procesos que se basan en el control estadístico, se realizan análisis estadísticos para controlar y seguir el proyecto así como para realizar estimaciones y predicciones [7].

Los datos recolectados pueden ser de mala calidad. Esto provoca que los análisis que se realicen (estimación de tamaño, costo, plazo) brinden muchas veces resultados incorrectos y entonces que las decisiones que se tomen (basadas en esos resultados) sean las equivocadas. Decisiones equivocadas pueden llevar a grandes pérdidas y al fracaso del proyecto [10].

Lamentablemente, en una extensa revisión de la literatura, Bachmann encuentra que casi no existen trabajos que estudien la calidad de los datos que se recolectan durante el uso de un proceso de desarrollo de software [3]. La mayoría de esos pocos trabajos encontrados estudia los datos de los registros de defectos y los datos del software de control de versiones, dejando de lado otros datos generados durante el uso de un proceso de desarrollo de software.

En otro estudio los autores encuentran que la mala calidad de los datos del proceso de desarrollo afecta la calidad del producto de software desarrollado [2]. Debido al severo impacto negativo que la mala calidad de los datos puede tener es que Shepperd manifiesta "...por esto sugiero que este tema [calidad de datos] debe convertirse en una alta prioridad entre los investigadores de ingeniería de software empírica" [12].

También se debe tener en cuenta que la Calidad de Datos es un área de investigación en sí misma, en la cual se ha generado un gran volumen de trabajo (sobre todo en los últimos años) enfocado principalmente a: definir los distintos aspectos de la calidad de los datos [4, 9, 11, 14], y proponer técnicas, métodos y metodologías para la medición y para el tratamiento de la calidad de los datos [4, 9, 15].

Indudablemente, importa menos la cantidad de datos de la que se disponga que la calidad de los mismos. Dicho de otra forma, la calidad de las decisiones es fundamental en cualquier proyecto; y decisiones de calidad solamente se pueden tomar contando con datos de calidad [6].

Nuestra investigación busca realizar un aporte en el estudio de la calidad de los datos recolectados durante el uso de procesos de desarrollo de software.[1] Esta investigación

[1] Debe quedar claro que la calidad de los datos del uso de un proceso, la calidad del proceso y la calidad del producto son diferentes. Tanto la calidad del proceso como la del producto son extensamente abordadas por

se diferencia de los artículos encontrados en la revisión bibliográfica de Bachmann en dos grandes aspectos:

- Buscamos analizar todos los tipos de datos que se generan durante el uso de un proceso de desarrollo de software y no solamente algunos tipos de datos.

- Utilizamos la disciplina Calidad de Datos como el fundamento y la base de nuestro estudio.

En este artículo presentamos una primera evaluación de la calidad de los datos del uso de un proceso de desarrollo de software utilizando datos del Proceso Personal de Software (*Personal Software Process*, de ahora en mas PSP). El PSP es un proceso para un individuo para desarrollar módulos y pequeños programas de software [7]. El PSP es altamente instrumentado e incluye un marco de medición riguroso; esto lo hace un proceso adecuado para nuestra investigación.

Nuestros datos provienen de cursos dictados desde junio de 2006 hasta setiembre de 2010. Estos cursos fueron dictados por el *Software Engineering Institute* (SEI) de la Universidad Carnegie Mellon o por asociados (*partner*) del SEI; incluyendo un número diferente de instructores en múltiples países. Contamos con 408 sujetos (ingenieros que realizaron el curso) en nuestros datos.

Encontramos un único artículo que evalúa la calidad de los datos del PSP [8]. En dicho artículo se presenta la evaluación de una versión "vieja" del curso del PSP. En esa versión del curso los estudiantes deben realizar muchos cálculos de forma manual para obtener ciertos datos del uso del proceso. En las versiones de cursos más nuevas, que son las usadas en nuestro trabajo, se utiliza una herramienta que hace los cálculos automáticamente. Los estudiantes, por ejemplo, tenían que realizar estimaciones utilizando el método de regresión lineal de forma manual. Los tipos de problemas en los datos que presentan los autores son diferentes a los que nosotros proponemos. Esto se debe a la diferencia en las herramientas utilizadas, cómo se recolectan los datos y cómo se realizan los cálculos derivados de esos datos (manual o automáticamente).

Para realizar nuestro estudio adoptamos un enfoque sistemático, disciplinado y estructurado para la evaluación de la calidad de los datos. Utilizamos la propuesta de Batini y Scannapieco, que proviene de la disciplina Calidad de Datos [4]. Utilizamos e instanciamos dicha propuesta para luego aplicarla en los datos del PSP. Esta instanciación puede resultar repetible en estudios similares.

La Figura 1 presenta el trabajo completo que realizamos. Luego del dictado de los cursos del PSP se identificaron los posibles problemas de calidad de los datos. Para cada problema, se identificaron y ejecutaron métricas específicas cuyos resultados fueron registrados. Luego se realizó la limpieza y migración de los datos. Este artículo presenta los problemas de calidad, las métricas definidas y la medición realizada en la base de datos del caso de estudio, dejando fuera las actividades de limpieza de datos por un tema de espacio.

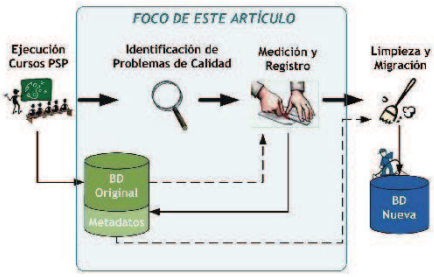


Figura 1. Etapas del estudio realizado

El artículo está organizado de la siguiente manera. El PSP se presenta en la sección 2. La sección 3 presenta las dimensiones y factores de la calidad de datos que son utilizados en este trabajo. La sección 4 presenta la metodología de trabajo y los problemas de calidad de datos en el uso del PSP. La sección 5 presenta un ejemplo de una métrica en particular. En la sección 6 se presentan los resultados y en la 7 las conclusiones.

2. El Personal Software Process

"El PSP es un proceso de mejora personal que ayuda a controlar, gestionar y mejorar la forma de trabajo" [7]. El proceso está dividido en fases que se van completando mientras se desarrolla el producto de software.

Para cada una de las fases del proceso existen guías que ayudan a seguir de forma correcta las actividades a desarrollar en cada fase. En cada una de estas el ingeniero de software recolecta datos sobre el tiempo utilizado en la fase y datos sobre los defectos que se han removido en la fase. Para cada defecto encontrado se registra: el tipo (siguiendo una taxonomía de defectos), el tiempo que llevó detectarlo y removerlo, la fase en la cual fue inyectado y la fase en la cual fue removido. La Figura 2 presenta las fases del PSP, las guías y la recolección de datos.

El PSP se enseña mediante un curso armado especialmente para ese fin. Durante el curso los estudiantes desarrollan programas mientras, de forma progresiva, aprenden el PSP. Existen diferentes versiones del curso, unas con 10 programas, otras con 8 y una con 7. En este artículo presentamos el análisis de la calidad de los datos recolectados

la literatura en ingeniería de software y en forma genérica por la literatura en gestión de la calidad. Sin embargo, como ya presentamos, la calidad de los datos recolectados durante el uso de un proceso de desarrollo de software casi no es abordada en la literatura.

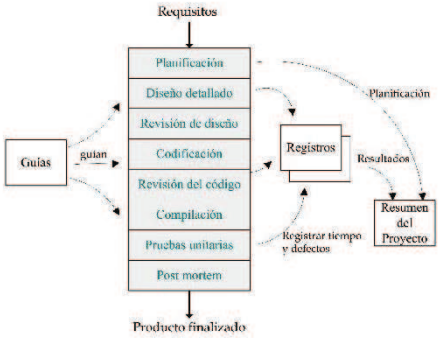


Figura 2. El Proceso Personal de Software

durante el dictado de cursos de 7 y 8 ejercicios. Los cursos fueron dictados por el *Software Engineering Institute* (SEI) de la Universidad Carnegie Mellon o por SEI *Partners*.

Durante el desarrollo de los programas del curso los estudiantes aprenden a planificar, desarrollar y evaluar el propio proceso utilizando las prácticas propuestas por el PSP. Para realizar el primer ejercicio del curso el estudiante utiliza un proceso definido y simple llamado PSP 0. A medida que el curso avanza se agregan nuevas actividades, fases y elementos: formas de estimar tamaño y esfuerzo, cómo realizar una planificación en el PSP, revisiones de código, elementos del diseño, revisiones de diseño, etc. A medida que estos elementos se agregan el proceso cambia. El nombre del proceso y los elementos fundamentales que se agregan en cada uno se presentan en la Figura 3. El PSP 2.1 es el PSP completo.

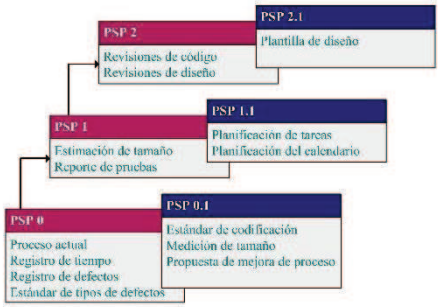


Figura 3. Los niveles del PSP en el curso

3. Calidad de Datos

Los datos constituyen un recurso muy valioso para las organizaciones al ser utilizados principalmente para la toma de decisiones, siendo de suma importancia para garantizar la sobrevivencia y éxito de las organizaciones.

La mala calidad de los datos influye de manera significativa y profunda en la efectividad y eficiencia de las organizaciones así como en todo el negocio, llevando en algunos casos a pérdidas multimillonarias [4]. Cada día se hace más notoria la importancia y necesidad en distintos contextos de un nivel de calidad adecuado para los datos.

Calidad de Datos es un área de investigación en sí misma, que ha avanzado mucho en los últimos años, generándose un gran volumen de trabajo científico en conferencias y *workshops* específicos [9, 11, 14, 15].

Existen distintos aspectos que hacen a la calidad de datos. Estos se conocen como dimensiones de calidad. En los trabajos del área de Calidad de Datos se propone gran variedad de conjuntos de dimensiones y de definiciones para las mismas [9, 11, 14, 15]. Sin embargo, existe un núcleo de dimensiones que es compartido por la mayoría de las propuestas. Este trabajo se basa en la propuesta de Batini y Scannapieco [4], que reúne estas dimensiones consensuadas.

En este trabajo utilizamos una abstracción de la calidad de datos [5], donde además de las dimensiones se definen otros conceptos para la clasificación y el manejo de la misma. Estos conceptos son el de factor, métrica y método de medición. Una dimensión de calidad captura una faceta (a alto nivel) de la calidad de los datos. Por otra parte, un factor de calidad representa un aspecto particular de una dimensión de calidad. Una dimensión puede ser entendida como un agrupamiento de factores que tienen el mismo propósito de calidad. Una métrica es un instrumento que define la forma de medir un factor de calidad. Un mismo factor de calidad puede medirse con diferentes métricas. A su vez, un método de medición es un proceso que implementa una métrica. Se pueden utilizar distintos métodos de medición para una misma métrica.

Las mediciones en una base de datos relacional se pueden realizar a varios niveles de granularidad: celda, tupla, tabla, e incluso a nivel de la base de datos entera. Por esto se definen funciones de agregación, que permiten pasar de un nivel de granularidad de datos a otro, obteniendo la calidad resumida para ese nuevo nivel. Por ejemplo, es posible obtener una medida de calidad de una tupla a partir de las medidas de calidad de cada una de sus celdas.

A continuación se presentan las dimensiones y factores de calidad utilizadas en este trabajo. De la propuesta de Batini y Scannapieco [4], no se considera la dimensión frescura relacionada con el tiempo, ya que los datos se consideran "frescos" y vigentes (respecto a la ejecución del proceso los

datos son eternamente vigentes).

Dimensión: Exactitud

La exactitud indica que tan precisos, válidos y libres de problemas están los datos. Establece si existe una correcta y precisa asociación entre los estados del sistema de información y los objetos del mundo real.

Existen tres factores de exactitud: exactitud semántica, exactitud sintáctica y precisión. La exactitud semántica se refiere a la cercanía que existe entre un valor v y un valor real v'. Interesa medir que tan bien se encuentran representados los estados del mundo real en el sistema de información.

La exactitud sintáctica se refiere a la cercanía entre un valor v y los elementos de un dominio D. Interesa saber si v corresponde a algún valor válido de D, sin importar si ese valor corresponde a uno del mundo real.

La precisión, por otra parte, se refiere al nivel de detalle de los datos.

Dimensión: Completitud

La completitud indica si el sistema de información contiene todos los datos de interés, y si los mismos cuentan con el alcance y profundidad que sea requerido. Establece la capacidad del sistema de información de representar todos los estados significativos de una realidad dada.

Existen dos factores de la completitud: cobertura y densidad. La cobertura se refiere a la porción de datos de la realidad que se encuentran contenidos en el sistema de información. La densidad se refiere a la cantidad de información contenida y faltante acerca de las entidades del sistema de información.

En un modelo relacional la completitud (en particular, la densidad) se caracteriza principalmente por los valores nulos. Un nulo puede indicar que dicho valor no existe, que existe pero no se conoce, o que no se sabe si existe en el mundo real.

Dimensión: Consistencia

Esta dimensión hace referencia al cumplimiento de las reglas semánticas que son definidas sobre los datos. La inconsistencia de los datos se hace presente cuando existe más de un estado del sistema de información asociado al mismo objeto de la realidad, y hay contradicciones entre dichos estados.

Las restricciones de integridad definen propiedades que deben cumplirse por todas las instancias de un esquema relacional. Se distinguen tres tipos de restricciones de integridad, las cuales se corresponden con los factores de esta dimensión.

Las restricciones de dominio, se refieren a la satisfacción de reglas sobre el contenido de los atributos de una relación.

Las restricciones intra-relación, se refieren a la satisfacción de reglas sobre uno o varios atributos de una relación. Las restricciones inter-relación, se refieren a la satisfacción de reglas sobre atributos de distintas relaciones.

Dimensión: Unicidad

La unicidad indica el nivel de duplicación de los datos. La duplicación ocurre cuando un objeto del mundo real se encuentra representado más de una vez en los datos, esto es, varias tuplas representan exactamente el mismo objeto. Distinguimos dos factores de la dimensión Unicidad. La duplicación, cuando la misma entidad aparece repetida de manera exacta, y la contradicción, cuando la misma entidad aparece repetida con contradicciones.

4. Problemas de Calidad en los Datos

Esta sección se subdivide en dos subsecciones. En la primera se presenta la metodología de trabajo y en la segunda se presentan los problemas de calidad identificados.

4.1. Metodología de Trabajo

El primer paso hacia la identificación de los problemas de calidad que podrían contener los datos bajo estudio, fue conocer la realidad y el contexto a analizar. Esto incluye el PSP, la herramienta para el registro de los datos y el propio modelo de la base de datos que utiliza la herramienta.

Una vez conocida la realidad y el contexto procedimos a analizar las dimensiones y factores de calidad propuestos por Batini y Scannapieco [4], que resulten interesantes de medir para dicho conjunto de datos.

Teniendo en cuenta cuáles son los datos relevantes y las dimensiones y factores de calidad a medir, se realizó una exploración de la herramienta utilizada en los cursos del PSP para registrar los datos del proceso, y un análisis de la estructura de la base de datos que los contiene. El foco estuvo en los valores que se ingresan de forma manual en la herramienta, ya que es allí donde ocurrirán la mayor cantidad de errores en los datos.

También fueron analizadas las *Grading Checklists*, utilizadas por los instructores para la corrección de los ejercicios realizados durante el curso, las cuales resultaron ser un gran aporte para identificar métricas relevantes para la calidad de datos en el PSP.

Clasificamos las métricas obtenidas en dos tipos: las que seguro miden un error en los datos y aquellas que permiten identificar datos sospechosos pero no nos permiten asegurar sea un error de datos. Por ejemplo, un valor negativo en un registro de tiempo es indudablemente un error en los datos. Por otro lado, resultaría sospechoso si un ingeniero no hubiese registrado ningún defecto considerando todos los programas desarrollados. En ese caso, es probable que el ingeniero haya olvidado registrar algún defecto durante el desarrollo, pero no podemos asegurarlo.

IX Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento
Lima, Perú

4.2. Definición de los Problemas de Calidad

La Figura 4 muestra la relación que existe entre dimensiones, factores, problemas y métricas, y cómo se aplican estos conceptos en un caso particular. Mientras que la definición de dimensión y factor de calidad es general y resulta aplicable en cualquier contexto, los problemas y métricas son específicos y definidos para el PSP, pudiendo ser utilizados para otros procesos de desarrollo de software.

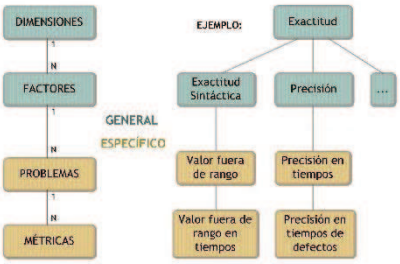


Figura 4. Relación entre Dimensiones, Factores, Problemas y Métricas

Las dimensiones de calidad de datos que se miden son: Exactitud, Completitud, Consistencia y Unicidad. En el Cuadro 1 se muestran todos los problemas identificados para cada factor y cada dimensión de calidad.

Dimensión	Factor	Problema de Calidad
Exactitud	Exactitud sintáctica	Valor fuera de rango
	Exactitud semántica	Identificador de proyecto incorrecto
	Precisión	Precisión en tiempos
Completitud	Densidad	Valor nulo
	Cobertura	Registro inexistente
Consistencia	Integridad de dominio	Reglas de integridad de dominio
	Integridad intra-relación	Reglas de integridad intra-relación
	Integridad referencial	Reglas de integridad referencial
		Referencia inválida
Unicidad	Duplicación	Registro Duplicado

Cuadro 1. Problemas de Calidad en los datos

A continuación se describe brevemente cada problema de calidad identificado. Para todos los casos, la unidad de medida del resultado es booleana: se mide si el objeto medido contiene o no un problema. Los problemas se miden, en la mayoría de los casos y salvo que se indique lo contrario, mediante la definición y ejecución de consultas *SQL*.

Valor fuera de rango

Los valores fueras de rango son aquellos que se sitúan fuera de un rango previamente definido como válido. Dichos valores podrían corresponder a valores anómalos y hacer que los resultados y conclusiones obtenidas al analizar los datos no reflejen fielmente la realidad.

Para cada uno de los casos identificados (por ejemplo, los tiempos) se establecen los criterios para determinar apropiadamente el rango que se va considerar para evaluar los valores, y se identifican los *outliers* mediante consultas en SQL. Un *outlier* es un valor que es inusualmente mayor o menor que otros valores en un conjunto de datos, pero que no corresponde necesariamente a un valor erróneo.

El rango se determina considerando el valor medio y la desviación estándar de los valores registrados. Los valores que se sitúen fuera de dicho rango se considerarán candidatos a contener errores, y por lo tanto serán analizados de manera aislada.

Identificador de proyecto incorrecto

Todos los usuarios deberían utilizar los mismos identificadores para hacer referencia a los mismos proyectos de la realidad.[2] De no ser así, resulta inviable poder realizar análisis de datos por proyecto.

Con este problema se identifican todos los proyectos que tienen asociado el proceso PSP correcto, pero su identificador de proyecto no se corresponde con la realidad.

Precisión en tiempos

Con este problema se desea medir el nivel de precisión de los tiempos registrados, ya que interesa conocer el momento de tiempo exacto en el cuál fue registrado un defecto. No debería suceder que las horas, minutos y segundos de los tiempos registrados sean todos iguales a 0.

Valor nulo

Interesa conocer qué información fue registrada y cuál fue omitida. Para aquella información que fue omitida, interesa conocer la causa de la omisión, y en caso que sea posible, determinar el valor que debería tomar en lugar de nulo.

Se identifican los campos que admiten nulos, pero deberían en la realidad contener algún valor distinto de vacío (el hecho de que admitan nulos es debido a un diseño incorrecto de la base de datos que utiliza la herramienta del curso del PSP).

Registro inexistente

Se identifican aquellos registros que no existen en la base pero sí existen en la realidad, y por lo tanto se omitió su ingreso. Esto significa que existe una porción de datos de la realidad que no se encuentra reflejada en la base de datos. Si

[2]En el PSP cada ejercicio (programa) tiene asignado un proceso (desde PSP0 a PSP2.1). Cada uno de estos programas tiene un identificador de proyecto.

no contamos con el universo total de datos, los análisis estadísticos que se lleven a cabo reflejará solamente una parte de la realidad estudiada.

Reglas de integridad de dominio

Para algunos atributos, es posible definir el dominio al cual sus valores deben pertenecer. En este caso, se define que el dominio válido para ciertos valores debe ser siempre mayor a cero.

Reglas de integridad intra-relación

Se definen un conjunto de reglas sobre ciertos atributos de una misma tabla, que deben ser satisfechas en la base bajo estudio. El hecho de que alguna de estas reglas sea violada, afecta la consistencia de los datos y por lo tanto cualquier análisis que se lleve a cabo a partir de estos.

Reglas de integridad referencial y Referencia inválida

Se definen un conjunto de reglas sobre ciertos atributos de diferentes tablas, que deben ser satisfechas en la base bajo estudio.

En particular, se identifican referencias hacia determinadas tuplas que no existen en la base y por lo tanto resultan ser referencias inválidas. Esto se debe a un error en el diseño del esquema de la base de datos, ya que se omite la definición de *foreign keys* sobre ciertos atributos.

Registro duplicado

Se identifica este problema de calidad cuando existen dos o más registros que aparecen repetidos de manera exacta. Existen dos situaciones:

- Cuando contienen el mismo valor en la clave y demás atributos (o en su defecto valores nulos). Este caso se contempla con controles del SGBD.
- A pesar de contener distinta clave primaria, hacen referencia al mismo objeto de la realidad y contienen los mismos datos en los campos que se definan. Para este caso se verifica que no existan registros repetidos (según el criterio definido) en la base bajo estudio.

Se consideran dos casos. La duplicación de defectos, que corresponde a defectos registrados a la misma hora exacta, y la duplicación de estudiantes, que corresponde a estudiantes que contienen el mismo nombre, mismo instructor y misma fecha de creación de perfil en la herramienta.

5. Ejemplo de Métrica

A modo de ejemplo en esta sección se presenta cómo se aplicó una métrica de un problema de calidad concreto, a un objeto de la base en particular. De esta forma se puede entender el trabajo realizado para cada uno de los problemas y métricas identificados.

Recordemos que una métrica es un instrumento que define la forma de medir un factor de calidad. En nuestro trabajo, definimos métricas para los problemas de calidad como el instrumento que asigna un número a un problema sobre uno o varios objetos de la base (atributos, tuplas y/o tablas específicas), dependiendo de su granularidad. Una misma métrica puede entonces ser utilizada sobre diferentes objetos.

Presentamos como ejemplo la métrica "Precisión en tiempos de defectos", que se encuentra definida dentro del problema de calidad "Precisión en tiempos". Este problema lo ubicamos dentro del factor Precisión y dimensión Exactitud.

Los ingenieros que realizan los cursos de PSP deben ingresar el momento exacto (fecha incluyendo hora, minuto y segundo) en el cual un defecto es removido. Esto se registra en la herramienta de forma manual, lo que puede ocasionar que se introduzcan errores en los datos. Dado que durante la exploración de la base de datos realizada para identificar los problemas de calidad detectamos que existían registros sin la precisión adecuada, resulta interesante medir cuáles son los registros que no cuentan con el detalle de hora, minuto y segundo.

Como vimos en la sección anterior, se estudió la realidad y el contexto, y se identificó un problema correspondiente a un factor de calidad y a una dimensión. Luego se define la métrica como instrumento de medición de dicho factor. Para esta métrica particular, la definición es: "Las horas, minutos y segundos, que corresponden a un registro de tiempo no son todos iguales a cero". También se debe especificar la forma de medición, la unidad del resultado y el método de medición. En este caso, la forma de medición consiste en encontrar los registros de defectos que cumplen que "hh:mm:ss" es igual a "00:00:00", donde el formato de fecha es "yyyy-mm-dd hh:mm:ss". Se define la unidad del resultado como un booleano, por lo tanto lo que obtenemos como resultado es si el registro medido es erróneo o no. El método de medición definido para esta métrica es la ejecución de una consulta SQL.

Al ejecutar la medición detectamos 1512 registros que contienen valores en el tiempo de defectos sin la precisión adecuada de un total de 20916 registros de defectos (7,23 %). Si lo vemos a nivel de ingeniero, detectamos 149 ingenieros distintos que cometieron este error al menos una vez de un total de 408 ingenieros (36,52 %).

En la Figura 5 se muestra un histograma que nos permite ver qué porcentaje de error cometen los ingenieros y cómo se distribuyen los mismos. En el mismo podemos observar que si bien el 63,48 % de los ingenieros tuvieron la totalidad de sus registros de defectos sin este error, también existen ingenieros que cometieron este error en la totalidad de sus registros (2,45 %).

Para poder entender por qué los ingenieros cometieron este error, decidimos explorar más a fondo la herramienta de registro utilizada en los cursos de PSP y encontramos que hay un defecto en la misma. Ya que al momento de re-

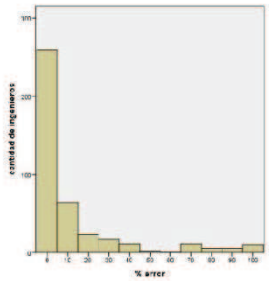


Figura 5. Distribución de ingenieros según porcentajes de error

gistrar un defecto, si se hace "doble *click*" en el campo de la fecha, el valor se pone correctamente con el formato "yyyy-mm-dd hh:mm:ss", pero sin embargo si se hace *click* en el calendario y se elige la fecha del día, la parte de "hh:mm:ss" queda con el valor "00:00:00", siendo esto último difícil de detectar por el ingeniero.

En esta sección no sólo vimos como definir una métrica y ejecutar la medición, sino una forma de visualizar los resultados y la interpretación de los mismos. Existen otras maneras de observar los resultados de la medición, como puede ser a través de un diagrama de dispersión para ver cómo evoluciona el porcentaje de error a medida que van avanzando los ejercicios del curso. De esa forma se puede detectar, por ejemplo, si los porcentajes de errores van bajando, si se mantienen constantes o si aumentan a medida que avanza el curso. Esta información puede ayudar (de forma temprana) a un instructor a ajustar problemas que tengan los estudiantes con la recolección de los datos. Recordemos que es importante lograr tener la mejor calidad de los datos posible, ya que muchas veces los ingenieros basan sus decisiones y estimaciones en datos históricos, y si los mismos tienen problemas de calidad, es probable que las decisiones tomadas no sean las más acertadas.

6. Resultados Generales

Se identificaron 10 problemas de calidad, y se definieron un total de 91 métricas para medir estos problemas aplicados sobre objetos (celdas y/o tuplas) de la base. Realizamos la medición de la totalidad de las métricas que fueron definidas. La ejecución de la medición para estas métricas se realizó en el 100 % de los casos de manera automática mediante sentencias SQL, donde para un 20 % de los casos se utilizaron algoritmos programados en PHP. Esta sección presenta los resultados más importantes y de forma general. Los resultados completos de nuestro análisis se encuentran en: www.fing.edu.uy/inco/grupos/gris/psp-data-quality.htm.

En el Cuadro 2 se muestra la cantidad de métricas definidas para cada problema de calidad, la cantidad de métricas que corresponden a errores y la cantidad que corresponden a casos sospechosos. Además se indica, para cada problema de calidad, el porcentaje de objetos con la presencia de ese problema de calidad según se consideren las métricas que miden errores o casos sospechosos.

Luego de ejecutar todas las mediciones, observamos que un 1,34 % del total de los objetos medidos tiene algún error. Si consideramos únicamente las métricas que miden errores (y no contamos los casos sospechosos) ese valor baja a un 0,99 %, y si consideramos solamente las métricas que miden casos sospechosos el porcentaje de objetos con error asciende a un 2,10 %. El estudio de Johnson y Disney presentaba un 4,8 % de errores en los datos [8]. En ese caso muchos de los errores en los datos se debían a cálculos manuales de datos derivados, situación que no ocurre con las herramientas actuales de soporte al PSP. Esta diferencia en la calidad de los datos debido a las herramientas utilizadas es también mencionada por Bachmann y Bernstein: "También discutimos el impacto de las características de los proyectos [estudiados] y concluimos que, por ejemplo, la naturaleza de los procesos de ingeniería de software, el uso de distintas herramientas de soporte a los procesos, [...] resultan en características diferentes en los datos" [1].

Estos resultados de calidad, vistos tanto a nivel global como a nivel particular para cada problema de calidad, pueden estar indicándonos que es necesaria una limpieza de los datos para garantizar que los futuros análisis estadísticos de los datos de estos cursos sean sobre datos válidos. Caso contrario los resultados y las conclusiones de dichos análisis pueden no corresponderse con la realidad.

Por otro lado, desde el punto de vista de la práctica de la ingeniería de software, queda claro que es necesario un análisis de la calidad de datos del uso de un proceso antes de utilizarlos. Evitar usar datos de mala calidad puede prevenir la toma de decisiones inadecuadas en el transcurso de un proyecto.

7. Conclusiones

En este artículo presentamos el uso de un enfoque sistemático, disciplinado y estructurado de la disciplina Calidad de Datos para identificar y medir los problemas de calidad en los datos recolectados durante el uso del PSP. No encontramos en la literatura otros estudios o publicaciones que aborden esta problemática con un enfoque como el que aquí se presenta. Esta propuesta puede ser utilizada, con adaptaciones, en otros procesos de desarrollo de software.

Los resultados muestran que el porcentaje de datos de

Problema de Calidad	# total de métricas	# métricas error	# métricas sospechosos	% objetos con error	% objetos sospechosos
Valor fuera de rango	8	0	8	0,00 %	3,20 %
Identificador de proyecto incorrecto	1	1	0	8,66 %	0,00 %
Precisión en tiempos	1	1	0	7,23 %	0,00 %
Valor nulo	26	26	0	1,32 %	0,00 %
Registro inexistente	3	1	2	0,12 %	16,90 %
Reglas de integridad de dominio	15	15	0	1,29 %	0,00 %
Reglas de integridad intra-relación	5	2	3	1,05 %	0,40 %
Reglas de integridad referencial	10	9	1	11,94 %	3,44 %
Referencia inválida	20	20	0	0,04 %	0,00 %
Registro duplicado	2	2	0	1,62 %	0,00 %

Cuadro 2. Cantidad de métricas y porcentajes de objetos con error para cada problema de calidad

mala calidad es menor al presentado en otro análisis realizado anteriormente [8]. Existe una diferencia importante entre nuestro estudio y el anterior: el uso de una herramienta de soporte al proceso. En el estudio anterior los datos eran recolectados en papel y los cálculos que había que realizar con dichos datos eran manuales. En nuestro estudio utilizamos cursos del PSP que ya contaban con el soporte de una herramienta que permite una mejor recolección de los datos y el cálculo automático de datos derivados.

La detección de errores en los datos así como su limpieza es importante en el contexto del uso de los procesos de desarrollo de software. Los datos que se generan durante el uso de un proceso son utilizados luego para predicciones, cálculos de avance del proyecto, etc. Si los datos que se utilizan son de mala calidad puede suceder que las decisiones que se tomen durante el proyecto, o acerca de un nuevo proyecto, sean las decisiones equivocadas.

Por otro lado, la ingeniería de software empírica utiliza datos de diversos experimentos o casos de estudio que luego analizar para confirmar o descartar ciertas hipótesis. Si los datos que se utilizan son de mala calidad las confirmaciones o rechazos realizados durante el estudio empírico pueden ser incorrectos.

Desde la perspectiva de la ingeniería de software empírica este artículo busca concientizar acerca de la importancia que tiene la disciplina de calidad de datos para los procesos de desarrollo de software. Desde la perspectiva de la calidad de datos, este trabajo muestra una aplicación de las técnicas de medición de calidad y de limpieza de datos a un dominio particular.

Referencias

[1] A. Bachmann and A. Bernstein. Software process data quality and characteristics: a historical view on open and closed source projects. In *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops*, IWPSE-Evol '09, pages 119–128, New York, NY, USA, 2009. ACM.

[2] A. Bachmann and A. Bernstein. When process data quality affects the number of bugs: Correlations in software engineering datasets. In *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, pages 62 –71, may 2010.

[3] A. J. E. Bachmann. *Why Should We Care about Data Quality in Software Engineering?* PhD thesis, University of Zurich, 2010.

[4] C. Batini and M. Scannapieco. *Data Quality: Concepts, Methodologies and Techniques*. Springer-Verlag Berlin Heidelberg, 2006.

[5] L. Etcheverry, V. Peralta, and M. Bouzeghoub. Qbox-foundation: a metadata platform for quality measurement. In *4th Data and Knowledge Quality Workshop*, 2008.

[6] E. Harper and D. Zubrow. Should you trust your data? *Software Quality Professional*, 13(4):4–8, 2011.

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

[8] P. M. Johnson and A. M. Disney. A critical analysis of PSP data quality: Results from a case study. *Empirical Softw. Engg.*, 4(4):317–349, Dec. 1999.

[9] Y. W. Lee, D. M. Strong, B. K. Kahn, and R. Y. Wang. Aimq: a methodology for information quality assessment. *Inf. Manage.*, 40:133–146, 2002.

[10] L. More. No data quality control? Expect to count the cost. *Computing*, pages 28–29, 2006.

[11] M. P. Neely. The product approach to data quality and fitness for use: A framework for analysis. In *Proceedings of the 10th International Conference on Information Quality MIT*, 2005.

[12] M. Shepperd. Data quality: Cinderella at the software metrics ball? In *Proceedings of the 2nd International Workshop on Emerging Trends in Software Metrics*, WETSoM '11, pages 1–4, New York, NY, USA, 2011. ACM.

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

[14] D. M. Strong, Y. W. Lee, and R. Y. Wang. Data quality in context. *Commun. ACM*, 40:103–110, 1997.

[15] R. Y. Wang, M. P. Reddy, and H. B. Kon. Toward quality data: an attribute-based approach. *Decis. Support Syst.*, 13:349–372, 1995.

References

[1] W. S. Humphrey, Winning with Software: An Executive Strategy, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[2] W. S. Humphrey and W. R. Thomas, Reflections on Management: How to Manage Your Software Projects, Your Teams, Your Boss, and Yourself (1st Edition), Addison-Wesley Professional, 2010.

[3] W. S. Humphrey, A Discipline for Software Engineering, Addison-Wesley, 1995.

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

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

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

[7] W. Hayes and J. W. Over, The Personal Software Process (PSP): An Empirical Study of the Impact of PSP on Individual Engineers, Techical Report CMU/SEI-97-TR-001, Software Engineering Institute, Carnegie Mellon University, December 1997.

[8] H. D. Rombach, J. Münch, A. Ocampo, W. S. Humphrey and D. Burton, "Teaching Disciplined Software Development," *Journal of Systems and Software,* vol. 81, no. 5, pp. 747-763, 2008.

[9] 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.

[10] W. Nichols, S. Küpper and U. Andelfinger, The Personal Software Process (PSP) Revisited: Empirical Benefits Analysis, Technical Report (draft version provided by the authors), 2013.

[11] M. C. Paulk, "Factors Affecting Personal Software Quality," *Cross-Talk: The Journal of Defense Software Engineering,* vol. 19, no. 3, pp. 9-13, 2006.

[12] C. Kemerer and M. C. Paulk, "The Impact of Design and Code Reviews on Software Quality: An Empirical Study Based on PSP Data," *IEEE Transactions on Software Engineering,* vol. 35, no. 4, 2009.

[13] C. Batini and M. Scannapieco, Data Quality: Concepts, Methodologies and Techniques, Springer, 2006.

[14] B. G. Tabachnick and L. S. Fidell, Using Multivariate Statistics, New York: Harper Collins, 1989.

[15] W. E. Deming, Out of the Crisis, MIT Center for Advanced Engineering Study, Cambridge, MA, 1982.

[16] J. Juran and F. Gryna, Juran's Quality Control Handbook, Fourth Edition, New York: McGraw-Hill Book Company, 1988.

[17] W. S. Humphrey, The Personal Software Process (PSP), Technical Report CMU/SEI-2000-TR-022, Software Engineering Institute, Carnegie Mellon University, November 2000.

[18] M. Fagan, "Design and Code Inspections to Reduce Errors in Program

Development," *IBM Systems Journal,* vol. 15, no. 3, 1976.

[19] M. Fagan, "Advances in Software Inspections," *IEEE Transactions on Software Engineering,* vol. 12, no. 7, July 1986.

[20] W. S. Humphrey, Managing the Software Process, Reading, MA: Addison-Wesley, 1989.

[21] M. C. Paulk, B. Curtis, M. B. Chrissis and C. V. Weber, The Capability Maturity Model: Guidelines for Improving the Software Process, Reading, Ma: Addison Wesley, 1995.

[22] J. Herbsleb, D. Zubrow, D. Goldenson, W. Hayes and M. C. Paulk, "Software Quality and the Capability Maturity Model," *Communications of the ACM,* vol. 40, no. 6, pp. 30-40, June 1997.

[23] W. S. Humphrey, The Team Software Process (TSP), Technical Report CMU/SEI-2000-TR-023, Software Engineering Institute, Carnegie Mellon University, November 2000.

[24] IEEE, Standard Glossary of Software Engineering Technology, ANSI/IEEE Std. 610. 12, 1990.

[25] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell and A. Wesslén, Experimentation in Software Engineering: An Introduction, Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[26] N. Juristo and A. M. Moreno, Basics of Software Engineering Experimentation, Kluwer Academic Publishers, 2001.

[27] N. E. Fenton and S. L. Pflegger, Software Metrics: A Rigorous and Practical Approach, Revised, Course Technology, February 1998.

[28] D. A. Kenny, D. A. Kashy and W. L. Cook, Dyadic Data Analysis, New York: Guilford, 2006.

[29] A. Marotta, Data Quality course material, Institute of Computing, Engineering College, UdelaR, 2009.

[30] A. J. E. Bachmann, Why Should We Care about Data Quality in Software Engineering?, PhD thesis, University of Zurich, 2010.

[31] A. Bachmann and A. Bernstein, "When Proces Data Quality Affects the Number of Bugs: Correlations in Software Engineering Datasets," *7th IEEE Working Conference on Mining Software Repositories (MSR),* pp. 62-71, May 2010.

[32] M. Shepperd, "Data Quality: Cinderella at the Software Metrics Ball?," *In Proceedings of the 2nd International Workshop on Emerging Trends in Software Metrics, WETSoM'11,* pp. 1-4, 2011.

[33] P. M. Johnson and A. M. Disney, "A Critical Analysis of PSP Data Quality: Results from a Case Study," *Empirical Software Engineering,* vol. 4, no. 4, pp. 317-349, December 1999.

[34] 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 IX Jornadas Iberoamericanas de Ingeniería de Software e Ingeniería del Conocimiento (JIISIC),* pp. 37-44, 2012.

[35] V. B. Kampenes, T. Dybå, J. E. Hannay and D. I. K. Sjøberg, "A Systematic Review of Effect Size in Software Engineering Experiments," *Information and Software Technology,* vol. 49, no. 11-12, pp. 1073-1086, November 2007.

[36] G. V. Glass, P. D. Peckham and J. R. Sanders, "Consequences of failure to meet

assumptions underlying effects analyses of variance and covariance," *Rev. Educ. Res.,* vol. 42, pp. 237-288, 1972.

[37] M. R. Harwell, E. N. Rubinstein, W. S. Hayes and C. C. Olds, "Summarizing Monte Carlo results in methodological research: the one- and two- factor fixed effects ANOVA cases," *J. Educ. Stat.,* vol. 17, pp. 315-339, 1992.

[38] L. M. Lix, J. C. Keselman and H. J. Keselman, "Consequences of assumption violations revisited: A quantitative revew of alternatives to the one-way analysis of variance F test," *Rev. Educ. Res.,* vol. 66, pp. 579-619, 1996.

[39] F. Grazioli and W. Nichols, "A Cross Course Analysis of Product Quality Improvement with PSP," *In Proceedings of the TPS Symposium 2012: Delivering Agility with Discipline. Special Report, Software Engineering Institute, Carnegie Mellon University CMU/SEI-2012-SR-015,* pp. 76-89, September 2012.

[40] I. Sommerville, Software Engineering - 9th Edition, Addison-Wesley, 2010.

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

[42] B. B. Agarwal and S. P. Tayal, Software Engineering, Laxmi Publications, 2008.

[43] R. L. Glass, "Matching methodology to problem domain," *Communications of the ACM,* vol. 47, no. 5, pp. 19-21, May 2004.

[44] A. Höfer and W. F. Tichy, "Status of empirical research in software engineering," *International Conference on Empirical Software Engineering Issues: Critical Assessment and Future Directions,* pp. 10-19, 2006.

[45] M. C. Paulk, C. V. Weber, S. García, M. B. Chrissis and M. Bush, Key Practices of the Capability Maturity Model, version 1.1, Technical Report CMU/SEI-93-TR-25, Software Engineering Institute, Carnegie Mellon University, February 1993.

[46] M. C. Paulk, B. Curtis, M. B. Chrissis and C. V. Weber, Capability Maturity Model for Software, version 1.1, Technical Report CMU/SEI-93-TR-24, Software Engineering Institute, Carnegie Mellon University, February 1993.