Universidade Federal de Pernambuco
Centro de Informática

Pós-graduação em Ciência da Computação

Comparing Different Testing Strategies for Software Product Lines

Paola Rodrigues Godoy Accioly

Dissertação de Mestrado

Recife
Abril, 2012

Universidade Federal de Pernambuco
Centro de Informática

Paola Rodrigues Godoy Accioly

Comparing Different Testing Strategies for Software Product Lines

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: *Paulo Henrique Monteiro Borba*
Co-orientador: *Rodrigo Bonifácio de Almeida*

Recife
Abril, 2012

Aos meus pais, Luiz e Marcia; As minhas irmãs Andrea e Renata e aos meus queridos sobrinhos Luigi, Luís Antônio e Luís Filipe — a dinastia dos Luis! Ao meu namorado, Italo.

Acknowledgements

Agradecimentos especiais a todos que contribuiram diretamente para a realização deste trabalho. Primeiramento ao meu orientador, professor Paulo Borba, pela sua orientação cuidadosa durante todas as fases desse trabalho. Ao professor Rodrigo Bonifácio, co-orientador deste trabalho, que mesmo a distância esteve sempre disponível para ajudar e aconselhar. Aos professores Cristiano Ferraz a Sérgio Soares pela disponibilidade e ajuda.

Gostaria também de agradecer aos membros do SPG pelos conselhos e amizade, em especial a Laís, Marcio, Leopoldo, Rodrigo e Társis. Agradeço também aos demais integrantes do LabES pelos momentos de descontração. Agradeço também ao CIn e aos seus funcionários pela estrutura e formação de qualidade que conheço desde quando ingressei em 2005 para a graduação em Ciência da Computação. Agradeço ao INES — Instituto Nacional de Ciência e Tecnologia para Engenharia de Software — e a CAPES por financiarem a minha pesquisa. Indiretamente, agradecimentos ao apoio de todos familiares e amigos.

I've missed more than 9000 shots in my career. I've lost almost 300 games. 26 times, I've been trusted to take the game winning shot and missed. I've failed over and over and over again in my life. And that is why I succeed.

—MICHAEL JORDAN

Resumo

Engenharia de Linhas de Produto de Software (LPS) é uma abordagem de desenvolvimento que reusa de forma estratégica uma base comum de artefatos que implementam um conjunto de sistema similares e compartilham características em comum, mas também são suficientemente distintos entre si. Alguns dos benefícios esperados por essa abordagem são: redução de *time-to-market* para lançamento de novos produtos, redução do esforço de manutenção e, indiretamente, a melhora da qualidade dos produtos.

Nesse contexto, a atividade de testes de LPS tem sido considerada um desafio, principalmente devido a enorme quantidade de produto gerados por uma LPS e também porque os requisitos variam de um produto para o outro. Para lidar com esses problemas, várias técnicas que derivam casos de testes funcionais para produtos específicos dentro de uma LPS tem sido propostas. No entanto, essa área de pesquisa ainda não possui muitas avaliações empíricas que mostrem o benefício de se utlizar casos de teste específicos por produto o que desencoraja a indústria a adotar tais técnicas.

Nesse trabalho apresentamos estudos que comparam empiricamente duas técnicas de desenvolvimento de casos de teste. Uma técnica genérica observada em um ambiente real de execução de testes e uma técnica de casos de teste específicos por produto cujos testes poderiam ser obtidos através de qualquer uma das técnicas de derivação de testes existente. Para comparar essas técnicas, conduzimos 5 experimentos controlados avaliando o impacto das técnicas sob o ponto de vista do processo de execução de testes, coletando métricas relacionadas ao esforço de execução de testes. Após a análise dos dados coletados alcançamos resultados que sugerem benefícios, tais como redução do tempo de execução, que a indústria poderia alcançar ao adotar uma técnica de derivação de testes.

Palavras-chave: Linhas de Produto de Software, Testes de caixa preta, Engenharia de Software Empírica

Abstract

Software Product Line Engineering (SPL) is a development approach that strategically reuses a common core of artifacts to implement a set of similar systems that share common characteristics, but are sufficiently distinct from each other. Some of the benefits of the SPL approach are: reduction of time-to-market, reduction of maintenance effort and indirectly enhancement of products quality.

In this context, SPL testing has been considered a challenging task, mainly due to the huge number of products that might be generated from an SPL and also because the requirements vary from one product to the other. To deal with these problems, several techniques for deriving product specific functional test cases have been recently proposed. However, this research area still lacks of empirical studies showing the benefits of using product specific test cases, discouraging the industry to adopt one derivation technique.

In this work we present studies that empirically compares two different test case design techniques. A generic technique that we have observed in a real test execution environment and a product specific technique whose functional test cases could be derived using any existing approach. In order to compare these techniques, we conducted 5 controlled experiments evaluating their impact from the point of view of the test execution process, collecting metrics related to test execution effort. After analyzing the data collected, we achieved some results suggesting the benefits, such as execution time reduction, that the industry could achieve by adopting a test derivation technique.

Keywords: Black-box Testing, Software Product Lines, Empirical Software Engineering

Contents

1 Introduction **1**
- 1.1 Summary of Goals 2
- 1.2 Outline 2

2 Background **3**
- 2.1 Software Product Lines 3
- 2.2 Black-box Testing 5
- 2.3 Empirical Sofware Engineering 8

3 Motivating Issues **11**
- 3.1 Test Case: User Sends an MMS with a Picture Attached 12
- 3.2 Test Case: User Checks Icon and Label on Mobile Phone Main Menu 13
- 3.3 Test Case: User Attaches Video to MMS 14
- 3.4 Consequences 15
- 3.5 Specific Test Cases for SPL 17

4 Evaluation Studies **19**
- 4.1 Experiment Definition 20
- 4.2 Hypothesis 20
- 4.3 Experiment Design 21
- 4.4 Experiment Operation 22
- 4.5 Test Suites Design 23
- 4.6 Experiments Outline 23
  - 4.6.1 First Experiment (Pilot) 24
  - 4.6.2 Second Experiment 25
  - 4.6.3 Third Experiment 25
  - 4.6.4 Fourth Experiment 26
  - 4.6.5 Fifth Experiment 26
- 4.7 First Experiment (Pilot) 26
  - 4.7.1 Products Selection 28
  - 4.7.2 Participants 28
  - 4.7.3 Lessons Learned 28
    - 4.7.3.1 Time Collection 29
    - 4.7.3.2 TaRGeT Problems 29
    - 4.7.3.3 Similar Test Cases for Both Features 29
- 4.8 Second Experiment 30

4.8.1 ManualTEST 30
4.8.2 Research Group Management System (RGMS) 30
4.8.3 Participants 32
4.8.4 Experiment Operation 32
4.8.5 Data Analysis 33
4.8.5.1 Time Analysis 33
4.8.5.2 Terminated CRs Analysis 37
4.8.6 Threats to Internal Validity 37
4.8.6.1 ManualTEST 38
4.8.6.2 Time Collection 38
4.9 Third Experiment 38
4.9.1 Instrumentation 39
4.9.1.1 TestWatcher 39
4.9.1.2 Time Collection Approach 40
4.9.2 Participants 40
4.9.3 Experiment Operation 40
4.9.4 Lessons Learned 41
4.10 Fourth Experiment 41
4.10.1 Participants 42
4.10.2 Experiment Operation 42
4.10.3 Data Analysis 42
4.10.3.1 Time Analysis 42
4.10.3.2 Terminated CR Analysis 46
4.10.4 Threats to Internal Validity 47
4.10.4.1 Configuration of Latin Square Replicas 47
4.10.4.2 Heterogeneous Environment 47
4.11 Fifth Experiment 48
4.11.1 Participants 48
4.11.2 Experiment Operation 48
4.11.3 Data Analysis 49
4.11.4 Time Analysis 49
4.11.5 Terminated CR Analysis 52
4.11.6 Threats to Internal Validity 53
4.11.7 Selected Experimental Case 53
4.12 Threats to External Validity 54

5 Conclusions 57
5.1 Related Work 58
5.2 Future Work 59

A Test Suites 61

B Data Analysis Script 75

List of Figures

2.1 Different models of mobile phones illustrating variabilities. 4
2.2 eShop feature model. 5
2.3 The iterative learning process [BHH05]. 8
2.4 Illustration of an experiment [BHH05]. 9

3.1 Toy example feature model. 11
3.2 Correct behavior for Carrier A products. 14
3.3 Possible consequences of generic test cases. 16

4.1 Generic and specific techniques. 19
4.2 Test suites. 23
4.3 TaRGeT simplified feature model. 27
4.4 ManualTEST interface. 31
4.5 RGMS feature model. 31
4.6 Second experiment box-plot graphic. 34
4.7 Individual results for each technique. 34
4.8 The regression model of our experimental design. 35
4.9 Box-cox technique. 36
4.10 TestWatcher: time collection tool. 39
4.11 Fourth experiment blox plot graphic. 43
4.12 Individual results for both approaches. 44
4.13 4th experiment Box-Cox graphic. 45
4.14 5th study box plot graphic. 50
4.15 Graphic comparing individual results. 51
4.16 Box-Cox curve. 52

List of Tables

2.1 Use case: user login. 6
2.2 Test case TC01. 7
2.3 Test case TC02. 7
2.4 Latin square design. 10

3.1 Test case: user sends MMS with picture attached. 12
3.2 Specific test case for products with Carrier A feature. 12
3.3 Test Case: User Checks Icon and Label on Main Menu 14
3.4 Test Case: user attaches video to MMS. 15

4.1 Layout of experiment design. 22
4.2 Experiment execution. 22
4.3 Generic test case. 24
4.4 Specific test case. 24
4.5 Second experiment average and standard deviation. 33
4.6 Second experiment ANOVA results. 36
4.7 Resume of reported CRs 37
4.8 4th experiment average and standard deviation. 43
4.9 Fourth experiment ANOVA results. 45
4.10 Reported CRs. 47
4.11 5th experiment average and standard deviation. 49
4.12 Fifth experiment ANOVA results. 51
4.13 Reported CRs. 53

A.1 SG_F1_1 62
A.2 SG_F1_2 63
A.3 SG_F1_3 64
A.4 SG_F1_4 65
A.5 SG_F1_5 65
A.6 SG_F1_6 66
A.7 SP1_F1_1 66
A.8 SP1_F1_2 67
A.9 SP1_F1_3 68
A.10 SP1_F1_4 69
A.11 SP1_F1_5 69
A.12 SP1_F1_6 70

A.13 SP2_F1_1 70
A.14 SP2_F1_2 71
A.15 SP2_F1_3 72
A.16 SP2_F1_4 73
A.17 SP2_F1_5 73
A.18 SP2_F1_6 74

B.1 Input file containing the collected data in the second experiment. 75

CHAPTER 1

Introduction

Software Product Line (SPL) engineering is a methodology that uses strategic reuse of a common core of artifacts to implement a set of similar systems with some variation among them that allow to attend specific requirements with respect to the same market segment. One of the main goals of the SPL approach is to increase the development productivity by reducing development costs and also reducing the time-to-market to release new products [PBvdL05].

In the SPL approach, as in all software engineering, efficient testing strategies are essential for achieving software quality and reliability. Between the different strategies of testing there is the black-box strategy, also known as functional testing, that focus on validating the software overall functionality based on its requirements specification [Bei95].

In the functional testing strategy, test cases are usually generated using use case scenarios as input. Since an SPL can generate thousands of different products, it is challenging to write functional test cases based on use case scenarios. This happens because the requirements change from one product configuration to another and the crosscutting nature of certain features, which are commonly scattered through different scenarios and tangled with the specification of other features.

In order to derive SPL test cases, different methodologies such as PLUTO [BG03] and ScenTED [RKPR05], that extend use case scenarios to represent variability and generate test cases for different product configurations, have been proposed. Nevertheless the research community still lacks of empirical studies that use and evaluate these proposals, in order to give a solid foundation for software product line testing in industry [TTK04, ER11].

Perhaps the absence of literature evidence about the benefits of such techniques discourage the industry to adopt them. As a result, from what we have observed in a real test execution environment, companies might use test documents with use case scenarios that usually describe family behavior as a whole, including optional and alternative steps in a single specification. For example, one test case that specifies the scenario of a report generator feature would contain all possible variants for report formats such as PDF, XHTML and XLS and testers would use this single test case to test all the products from the SPL, even in the case where some of the products are not configured with all of these options.

However, such test suites may hamper test manual execution because these inaccuracies can mislead testers that have to strictly follow the test case script. This can lead to some unwanted consequences, such as escaped defects, that is, when the tester doesn't find an error prior to the product release. In addition, testers may take longer to execute test cases and report defects that doesn't exist, decreasing test execution productivity.

Alternatively, with the adoption of an SPL test derivation technique that manages variability in test suites, it would be possible to generate different versions of the same test suite

customized for the different configurations in the product line. This way, testers wouldn't get confused during test execution process. From now on, in this work, we name the use of test suites that contain all variants specifications together as generic technique (GT). Likewise, we name the use of configurations customized test suites as specific technique (ST).

1.1 Summary of Goals

In this work we have the main goal of executing empiric evaluations that compares two different SPL test design techniques: the GT and the ST, and evaluate the impact of the use of these techniques on the test execution process by measuring the effort to execute test cases. With this contribution we seek to investigate the benefits and disadvantages on both techniques. Next section presents the outline of this work.

1.2 Outline

The remainder of this work is organized as follows.

- Chapter 2 reviews essential concepts used throughout this work. Namely, Software Product Lines, Black-box Testing, and Empirical Software Engineering.
- Chapter 3 discusses how test cases may turn up to be generic (describing family overall behavior) or specific (presenting each product behavior). Besides that, we discuss what are the consequences of generic test cases in a test execution environment.
- Chapter 4 it's the core of this work. It presents the planning, execution and results of the executed experiments.
- Chapter 5 summarizes the contributions of this research, discusses some related and future work, and presents our main conclusions.
- Appendix A presents an example of the generic and the specific test suites used in the experiments.
- Appendix B describes the script used to analyse data using the R statistical software [R D08].

CHAPTER 2

Background

The purpose of this chapter is to present enough background so that readers will be able to follow the main ideas used in our work. First we discuss Software Product Lines (SPL) in Section 2.1. In this context we present the SPL main characteristics such as the use of Feature Models as one of the artifacts that helps the SPL visualization and selection of products configurations. Next, Section 2.2 presents black-box testing which consists in a software testing strategy that takes into account the external functionality of the system under test. Finally, in Section 2.3, we present essential concepts of empirical studies applied to software engineering.

2.1 Software Product Lines

Pohl et al. described Software Product Line Engineering (SPL) as a development methodology that strategically reuses a common core of artifacts to define a family of systems instead of a single system [PBvdL05]. These reusable artifacts include the results from the different phases of software development such as requirements, analysis and design, implementation and tests.

The set of systems derived from a SPL process share common characteristics that serve to a common market domain, but are also sufficiently distinct from each other. For example, Figure 2.1 illustrates three different models of mobile phones running the simplified version of the Rain of Fire game, developed by Meantime Mobile Creations [Mea12]. As we can notice, these phones differ from memory capacity and screen size. As a result, the game has to adapt to these different characteristics. For instance, the phone on the left has only 64kb of memory, therefore, we can choose to remove the moving clouds image in the game background, present on the other two products, to reduce the use of memory.

In order to develop a family of systems that can adapt to satisfy such particular restrictions and other specification details, the engineering paradigm of SPL is divided in two levels, the Domain Engineering level and the Application Engineering level. The first one comprehends the core assets development. It establishes the reusable platform and consequently the commonalities and variabilities present on the SPL, whereas the second one is responsible for deriving product line applications based on the predefined core assets.

The main benefits of the SPL approach are the reduction of development costs and time-to-market to release new products as many artifacts can be reused for each new product. Besides that, SPL can also lead to a decrease of maintenance effort since changes made on the domain engineering level with the purpose of error correction propagates to all products. Finally, because the artifacts in the platform are reviewed and tested in many products, SPL also indirectly improves the overall quality of the released products.

Figure 2.1 Different models of mobile phones illustrating variabilities.

However, in order to achieve such benefits, managing commonality and variability across the product line in an efficient way is essential. From design to implementation and testing, all artifacts produced on domain engineering level need the ability to represent variability. In this context, according to Kang et al [KCH+90], features can be used to describe commonalities and variabilities since they are prominent or distinctive user-visible aspects, qualities, or characteristics of a software system or systems.

For the purpose of representing variability, features can be organized in diagrams detailing an hierarchical decomposition of features, with a specific notation for each kind of relationship between them. Such diagrams are known as Feature Models defined feature models as a diagram could be used to represent requirements of the product lines.

As an example, Figure 2.2 shows the feature model of the eShop Product Line [Pro12]. As we can oberve, it uses a tree-like notation to describe which features are mandatory, optional or alternative. For example, *Catalog* is a mandatory feature, which means that it is not possible to generate a configuration from this product line without the functionality associated with this feature. On the other hand, *Search* is an optional feature, that may or may not be present in the product. In addition, features can also be gathered in *alternative* groups, where at least one, and only one feature, can be selected, or in *or* groups, where features can be freely combined, but at least one has to be selected. Lastly, one can also represent global constraints using feature models. In the eshop example we can see that the *CreditCard* feature can only be selected if, and only if, the *High Security* feature is selected.

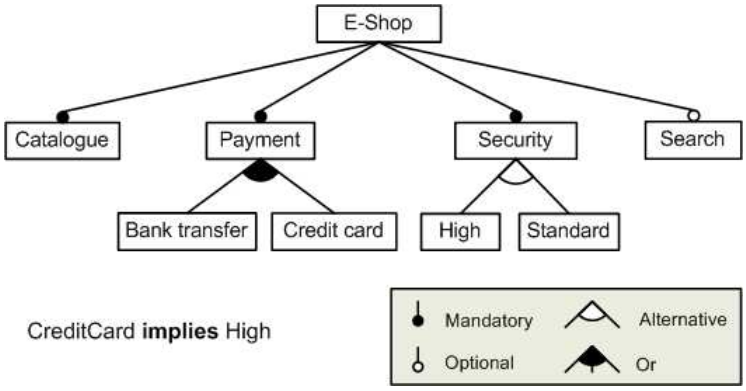

Figure 2.2 eShop feature model.

2.2 Black-box Testing

Myers [Mye04] defined software testing as *"the process of executing a program with the intent of finding errors"*. Software testing is the primary method that the industry uses to improve products reliability and quality. [Bei90]. In fact, software testing is not only a process to find errors but is also a dedicated discipline to evaluate its quality. The discipline of software testing defines two main strategies to test software, the white-box testing strategy and the black-box testing strategy [Bei95].

The white-box testing strategy, also known as structural testing, takes into account the internal mechanism of the system and its components. The tester needs to verify the behavior of classes, methods and its logical code structures. It is common that code developers perform this activity since they need to know what the code looks like to write test cases by choosing inputs that explore paths through the code and to determine the correct output.

On the other hand, the black-box testing strategy, also called as functional testing and behavioral testing, ignores the internal mechanism of the system and focus solely on the overall external system behavior. Testers using the black-box strategy are interested in determining whether or not a program does what it is supposed to do based on its functional requirements. As opposed to white-box testing, testers don't need to have previous knowledge of the system internal mechanisms, in fact they need only to know what the system is supposed to do based on its specification. In this work we focus on the black-box strategy.

In the black-box testing strategy, a primary task is defining test cases that will be able to detect as many defects as possible. Functional test cases are written based on the system use case scenarios specification. The test analyst uses these scenarios specifications as input to derive test cases. The resulting test cases consist in a sequence of steps that perform a task or a group of tasks depending on the test case objective. Furthermore, a test case defines the expected results, that is, the outcomes that the system should provide.

To exemplify how to write test cases based on use cases, let's take a look in the use case described in Table 2.1. It considers the scenario of an user performing the login task in the eShop login screen. The use case describes the main flow, where the user provides the correct values for username and password and the system correctly authenticates the user. Additionally,

Table 2.1 Use case: user login.

Use Case ID:	[UC01]
Name:	User login
Actor:	All
Associated Requirements:	[RF01] User Login
Inputs:	1- Username; 2- Password.
Initial Conditions:	The system must have at least one registered user.
Final Conditions:	The user is logged into the system
Main Flow:	1- User provides the correct values for username and password to the username and the password fields; 2- The system displays correctly the provided information; 3- User presses the login button or presses the enter key on the keyboard; 4- The system authenticates the user correctly and displays the main screen.
Alternative Flow 1:	1- User presses the login button without providing the username or the password; 2- The system presents the alert message “Please provide username and password”.
Alternative Flow 2:	1- User provides wrong values for username or password and presses the login button; 2- The system presents the alert message “wrong or invalid user”.

the use case considers two alternative flows. The alternative flow 1, where the user does not provide one of the required inputs, and the alternative flow 2, where the user provides wrong values for the required inputs.

Based on UC01 specification, the test analyst can generate one or more test cases. Here we describe two possible test cases, TC01 and TC02. TC01, that is described in Table 2.2, explores UC01 main flow. Whereas TC02, described in Table 2.3, explores the use case alternative flows, first, leaving the password field empty and then entering an incorrect password value.

When executing functional tests, testers need to execute the steps described in the user actions column, observing if the software behaves accordingly to the described system responses. If the system outcome does not correspond with its expected behavior, the test has probably revealed a defect of the system. In the given example, if the tester notices that, on TC01 step 2, the system presents a different behavior, an alert message for example, this probably means that the system login functionality is not working as it is supposed to. In this case, the tester needs to report this failure. The defect report is usually called as Change Request (CR). After reporting the CR, the development team evaluates it and correct the defect.

In this section we defined how black box tests are specified for traditional software development when a single system is taken into account. However, in the context of software

Table 2.2 Test case TC01.

Test Case ID:	TC01	
Objective:	Verify if the system login works correctly.	
Requirement:	[RF01] Users Login	
Initial Conditions:	The system must have at least one registered user.	
Step nº	**User Action**	**System Response**
1	User provides the correct values for username and password to the username and the password fields	The system displays correctly the provided information
2	User presses the login button or presses the enter key on the keyboard	The system authenticates the user correctly and displays the main screen
Final Conditions:	The user is logged into the system	

Table 2.3 Test case TC02.

Test Case ID:	TC02	
Objective:	Verify how the system login behaves with wrong/invalid parameters	
Requirement:	[RF01] Users Login	
Initial Conditions:	The system must have at least one registered user.	
Step nº	**User Action**	**System Response**
1	User provides the correct value for username but leaves the password field empty	System presents the alert message: please provide username and password
2	User provides incorrect value in the password field and hit the login button	System presents the alert message: wrong or invalid user
Final Conditions:	The user is not logged into the system	

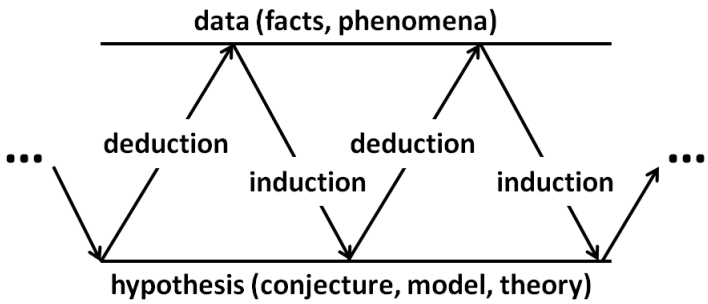

Figure 2.3 The iterative learning process [BHH05].

product lines, some changes need to be considered. In particular, in the domain engineering level, the use case asset needs to assimilate constructs that represent variability. Besides that, there should be a mechanism that generates use case specifications for the products derived from a software product line. We discuss this matter over the next chapter.

2.3 Empirical Sofware Engineering

Box et al. said that *"scientific research is a process of guided learning"* [BHH05]. This happens because learning advances alongside with iteration cycles like the ones illustrated in Figure 2.3. First an initial hypothesis leads, by a process of deduction, to certain consequences that we may compared with data. When consequences and data fail to agree, this divergence can lead, by a process of induction, to a new hypothesis or modification of models and theories. Then a second cycle of the iteration starts to compare the modified hypothesis with the new data.

According to Wohlin et al., *"... carrying out quantitative empirical studies is the opportunity of getting objective and statistically significant results regarding the understanding, controlling, prediction and improvement of software development"* [WRH+00]. This way, before introducing a new method, process or a new way of working, an empirical assessment of the characteristics, pros and cons, of such changes is preferred.

According to Pfleeger [Pfl94] there are three different empirical methods that could be used for assessing a technique in a scientific way, namely surveys, case studies and controlled experiments. These methods, however, are not mutually exclusives, that is, each one serves a particular scenario dependending on what the experimenter wants to investigate. Surveys are used when the use of a technique or tool has already taken place or before it was introduced. They aim at the development of generalized suggestions and these opinions can be collected by the means of interviews or applied questionnaires. One example of the use of surveys in software engineering would be to learn the opinion of some population of potential users of a mobile phone application before releasing it to the market.

Case studies, on the other hand, are conducted to investigate a single entity within a specific time space. The researcher uses different data collection procedures to collect information from a single project during a sustained period of time. Within software engineering, case studies can be used, for example, to compare the implementation of the same asset using different

delevopment environments such as Eclipse and Java Netbeans. However, case studies present a low degree of control of the factors. For this reason, conclusions obtained from case studies could not be generalized.

Lastly, controlled experiments collect metrics that compare two or more treatments controlling other variables that may present influence in the results. In addition, the results of a controlled experiment are expressed in terms of some degree of confidence provided by a statistical test of significance that could be generalized [WRH+00]. For the purposes of this work, controlled experiment is the appropriate method.

When conducting a controlled experiment, the researcher is interested in assessing the outcome response of a certain process while varying some of the input variables [JP05]. In this context, as illustrated in Figure 2.4, we name the outcome variables assessed as metrics and the input variable that vary along the process is the treatment. Nevertheless, besides metrics and treatments definition, the experimenter also needs to pay attention to other factors that can influence the outcome of the process. This kind of unwanted influence is commonly called as noise in empiric studies.

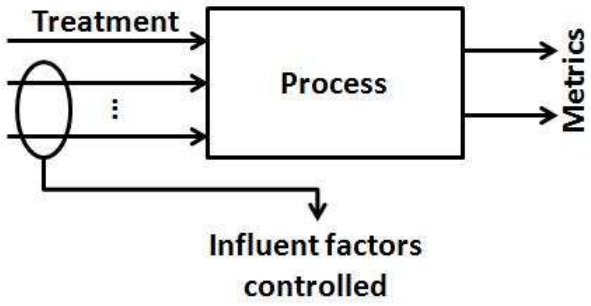

Figure 2.4 Illustration of an experiment [BHH05].

In a controlled experiment the proper design and layout is crucial to the success of the study. The role of the experiment design is to arrange treatments, metrics and factors in a way that the three principles of experiments design are considered. These principles are *Replication*, *Randomization* and *Local Control*. The *Replication* principle allows the repetition of the experiment on many experimental units. Secondly, the *Randomization* tates that there should be a process of assigning the treatments among the experimental units so that every treatment has equal chance of being assigned to any experimental unit. And lastly, the principle of *Local Control* arranges the influent factors in a way that the noise can be controlled during the experiment execution.

There are many ways to arrange the experiment layout. Particularly, in this work, we work with the latin square design [BHH05], which is used when there are two known factors that can influence the experiment outcome. Table 2.4, for example, describes a latin square design for an experiment with three treatments A, B and C. In this arrangement, the lines display the different levels of one controlled variable and the columns display the different levels of the second controlled variable. Also, inside the square, in each given line and in each given column the treatment can appear only once. This arrangement follows the three principles of experiment design because it has local control of two known factors, the randomization is applied inside

the square where the treatments are arranged and finally, this design also allows the replication since the researcher can replicate this square and analyze more than one experimental unit.

Table 2.4 Latin square design.

	I	**II**	**III**
I	C	B	A
II	B	A	C
III	A	C	B

After executing the experiment, the researcher analyzes the assessed metrics in order to answer some research questions. To do that there are a number of descriptive graphics such as box-plot, histograms, dot plots and other graphics that can be used depending on the nature of the data collected. There are also some measures that help to understand the tendency of data, such as average, median and standard deviation. However, when sampling over some population, the researcher might wish to know to what extent his results can be generalized considering the entire population.

The basis for the statistical analysis of a controlled experiment is hypotheses testing. A hypothesis is stated formally and the data collected during the experiment execution is used to test this hypothesis. If the hypothesis can be rejected it can draw to some conclusions that can be generalized under given risks. In the planning phase of the experiment two hypotheses have to be formulated, the *null* and the *alternative* hypotheses.

The null hypothesis, which we use as the basis for the investigation, usually states that there are no real trends and patterns concerning the population. In other words, there is no significant difference between the treatments. On the other hand, the alternative hypothesis is the one in favor of which the null hypothesis is rejected. If the null hypothesis is rejected we can infer that the alternative hypothesis is true. For example, we can denote these hypotheses as follows, considering the average of responses under two given treatments, A and B.

$$H_0 \quad : \quad \mu_A = \mu_B \tag{2.1}$$

$$H_1 \quad : \quad \mu_A \neq \mu_B \tag{2.2}$$

To test the null hypothesis there are a number of statistical tests such as Student test and analysis of variance (ANOVA) [BHH05]. These statistical tests return a value, known as p-value, that represents the probability with which we can affirm that the null hypothesis is true. The smaller the p-value is, there is a higher probability to reject the null hypothesis.

However, testing hypothesis involves two different types of risks, the *Type I error* and the *Type II error*. The *Type I error* occurs when the test rejects a true hypothesis. On the other hand, if the test fails to reject a false hypothesis we have a *Type II error*. To avoid *Type I* errors, it is often used a p-value smaller than 5%. This means that, in order to reject the null hypothesis, the p-value resultant from the statistical test should be smaller than 0.05. In this work we follow this recomendation.

CHAPTER 3

Motivating Issues

This chapter describes some practical examples of how test cases may turn up to be generic (describing the product family overall behavior) or specific (showing the specific steps and parameters suitable to each product). Besides showing this difference, we also explain the consequences of using generic test cases in a test execution environment. The examples described here relates to a toy example of a mobile SPL that manages the interaction of multimedia content (pictures, videos and music), multimedia messaging service (MMS), and some requirements made by a mobile phone carrier. Figure 3.1 illustrates the feature model of this SPL that represents some scenarios that we have observed in a real test execution environment. It is also important to notice that the scenarios described in this chapter are simplified to facilitate the readers comprehension. In a real test execution enviroment the test cases usually contain more details about the product under test.

This feature model contains four main features. The first one called *Media* describes the types of multimedia files that the mobile phone can manage. It consists in an *or* group that can freely combine its elements *Video*, *Music* and *Picture*. The mandatory feature *Camera* contains an optional feature called *Low Resolution* that allows the user to take pictures of make videos using a lower resolution. The feature *MMS* allows the user to send messages with multimedia files attached and it has an optional feature called *Limit Size* that limits the size of the MMS sent. Finally the *Carrier* feature consists of an *alternate* feature group that represents the mobile phone carrier specific requirements. Two mobile phone carriers will sell these products and it is common that they ask for some customizations on their products. For instance, they can add alert messages or display they logo in different screens.

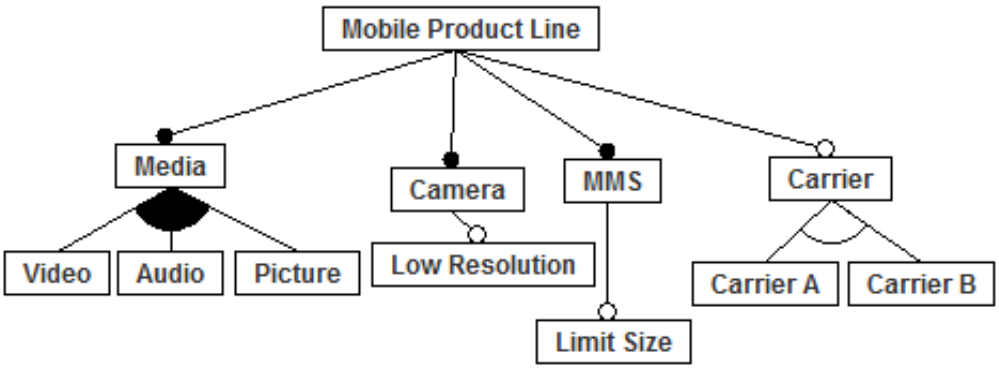

Figure 3.1 Toy example feature model.

Table 3.1 Test case: user sends MMS with picture attached.

Step ID	User Action	System Response
1	Go to Main Menu	Main Menu appears
2	Go to Messages Menu	Message Menu appears
3	Select “Create new Message”	Message Editor screen opens
4	Add Recipient	Recipient is added
5	Select “Insert Picture”	Insert Picture Menu opens
6	Select Picture	Picture is Selected
7	Select “Send Message”	Message is correctly sent

Table 3.2 Specific test case for products with Carrier A feature.

Step ID	User Action	System Response
1	Go to Main Menu	Main Menu appears
2	Go to Messages Menu	Message Menu appears
3	Select “Create new Message”	Message Editor screen opens
4	Add Recipient	Recipient is added
5	Select “Insert Picture”	Insert Picture Menu opens
6	Select Picture	Picture is Selected
7	Select “Send Message”	Dialog appears: “Are you sure you want to send this message? Data transfer shall be charged”
8	Hit “Yes”	Message is correctly sent.

3.1 Test Case: User Sends an MMS with a Picture Attached

Our first example considers the scenario of an user sending an MMS with a picture attached as detailed in Table 3.1. This scenario applies for most products of the SPL in discussion. However, it does not apply for products containing the Carrier A feature, which corresponds to a group of requirements associated to a specific mobile carrier. This carrier specifically requires that, before any data transfer, a message pops up asking the user if he really wants to send that message— since data transfer will be further charged. Differently, Table 3.2 describes the real product behavior when the product follows the Carrier A feature requirements. On steps 7 and 8 we can see the differences from Table 3.1. The Mobile Product Line can generate 112 different products. From this total, 28 products contain the Carrier A feature and need this extra step on the test case specification.

In the generic technique (GT), the test case detailed in Table 3.1 would serve to test all SPL products and the tester would be confronted with an innacuracy while testing products configured with the Carrier A feature. On the other hand, while using the specific technique (ST), there would be two different test cases. The first one, detailed in Table 3.1, would serve to test the products not configured with the Carrier A feature, whereas the second one, detailed in Table 3.2, would serve to test those products that were configured with the Carrier A feature.

In the black-box testing strategy, when the test specification fails to agree with the product

behavior, it probably means that the test case has revealed a defect. However, when testing a Carrier A product, that's not what happens with the generic test case just described. Here, the product works fine according to its configuration. The problem is that the generic test case does not consider all steps required to perform the task correctly. The implementation is ok, but the specification is vague. In this context when the tester is not familiar with the products specificities prior to the test execution, it is likely that he might interpret the test innacuracy as a product defect. This misunderstanding can be solved if the tester finds evidence that the test specification does not apply for that product. He can do that by talking to a requirements analyst or reading the products specification. However, if he can't find this evidence he will report an invalid product defect and he will have spent some os his time and some other person time to analyse this inaccuracy.

Besides reporting invalid defects, a different issue might happen. Let's imagine that the product configured with the Carrier A feature does not present the alert message before sending the MMS. In this case, the product was not correctly implemented, and the specification is vague. There is a product defect that the tester needs to report so that the development team can correct it. However, the tester won't be able to notice this defect because the generic test case does not consider the step that shows the alert message. If the defect is not reported the product will be realeased to the market without considering the requirement made by the Carrier A. In the software testing discipline we call this as an *escaped defect*, that is, defects that are found after the product release. This scenario might be worse than reporting invalid defects because it affects directly products quality, instead, reporting invalid defects only affects the test process productivity.

3.2 Test Case: User Checks Icon and Label on Mobile Phone Main Menu

Our second example considers the test case, detailed in Table 3.3, where the tester has to check the icon and the label for web navigation on the mobile main menu. Again, this scenario applies for the majority of the products except for the ones configured with the Carrier A feature, which exposes the carrier brand logo in a different way. In practice, in a real test execution environment, this test case would contain a link leading to a document that describes the icons that should be displayed, but here we simplified this specification by representing the mobile phone screen in Figure 3.2. This figure displays the proper behavior of the product with the Carrier A logo and the label "Blue Web" instead of "Web". A specific test case for a product configured with Carrier A would have the same steps described in Table 3.3 except for the differences on the logo and the label that would expose the Carrier A brand logo.

When following generic test cases, testers confronted with this kind of differences may be confused about the right product behavior. Again, he may interpret this innacuracy as a product defect since the labels and icons don't agree with the test specification. On the contrary, considering the scenario where the product implementation is wrong, in other words, the product does not present the Carrier A logo correctly, the tester will pass the test case and the product might be released to the market without the Carrier A requirements implemented correctly.

Table 3.3 Test Case: User Checks Icon and Label on Main Menu

Step ID	User Action	System Response
1	Go to Main Menu	Main Menu Appears
2	Check the browser navigation option	World icon appears with the title “Web”

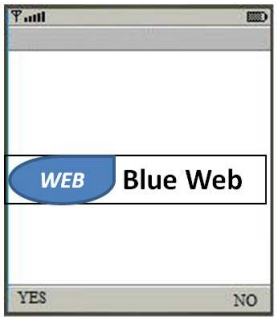

Figure 3.2 Correct behavior for Carrier A products.

3.3 Test Case: User Attaches Video to MMS

Our third and last example contemplates the scenario detailed in Table 3.4. This time, the test case asks the user to make a 5 seconds video and then try to send it with an MMS. This test case aims to test the feature *Limit Size* described in the feature model depicted in Figure 3.1. This feature limits the MMS size so, if the user tries to attach a file that exceeds this limit, a message appears asking if the user wants to resize the file in order to send the message. However, some products are configured with the feature *Low Resolution* that gives the option to take pictures and make videos using a lower resolution. A 5 seconds video taken with low resolution would not exceed the MMS limit size. In this way, steps 5 and 6 don’t apply to these products.

In this scenario, the generic test case failed to specify the interaction between the MMS limit size and the camera low resolution features. This probably happened because when the feature *Low Resolution* was incorporated by the SPL the interaction between these two features was not specified correctly considering this situation. Evolving and maitaining generic specifications is an error prone activity and some spots where there should be a feature interaction the steps are missing. A specific scenario for products configured with both features should specify, on step 5 of Table 3.4, the following system response “Video is correctly attached”. Then the next step would be to attach the message recipient.

In summary, generic test cases may differ from specific test cases in three different ways. First, they might present less steps than the product requires like discussed in our first example. Alternatively, they might present different parameters such as icons and labels as we showed in our second example. Lastly, they might describe more steps than necessary for some products, like presented in our third example.

The problem with the inaccuracies on generic test cases is that testers don’t always know each SPL product specificities prior to test execution. Specially because new features are often incorporated by the SPL and new configurations and features interaction are possible. Consequently, testers get confused when the test cases are not suitable with the product behavior.

Table 3.4 Test Case: user attaches video to MMS.

Step ID	User Action	System Response
1	Go to Main Menu	Main Menu appears
2	Open Camera Application	Camera App opens
3	Make a 5s video	Video is correctly saved to phone memory
4	Select Options	Option Menu appears
5	Select “Send as MMS”	Dialog appears: “Video is too large to attach. Do you want to resize it?”
6	Hit “Yes”	Video is correctly resized and attached
7	Add recipient	Recipient is added
8	Select “Send Message”	Message is correctly sent

This can lead to a decrease of productivity in the test execution process since testers might take longer to execute the test cases and report invalid defects. In addition, when the product implementation is wrong and the specification is vague there might be escaped defects. In the next section we further discuss how these inconsistencies may impact the process of testing those products.

3.4 Consequences

In a black-box test execution environment, testers are not required to have specific knowledge of the application code structure [Bei95]. Only by reading test scripts, they are aware of what the system under test is supposed to do. They follow the user action steps checking if the product behaves according to the described system response. Whenever there is an inconsistency between the test case and the product behavior, testers must investigate whether this inconsistency is a defect. In this context, generic test cases may hamper test execution because when they fail to specify a certain product behavior, testers are not able to identify if there is an issue in the test case. Instead, they might interpret it as a product defect.

The activity diagram described in Figure 3.3 considers the scenarios that tipically happens when the test case is not accurate. We have observed this scenario in a non trivial real test execution environment that uses generic test suites. The activity flow starts when the test case doesn’t specify correctly the behavior of the system under test, presenting an issue similar to those described in Sections 3.1, 3.2 and 3.3. Then the fork on the diagram indicates two possible scenarios. In the left branch the product matches the test case description, in other words, there is a defect but the tester won’t be able to notice, because the test case is also wrong. So the tester passes the test and the consequence is an escaped defect. For instance, in the example described in Subsection 3.2, the mobile company would release the product without the Carrier A brand logo on the Web Navigation option.

In the right branch, the product works fine, so the inconsistency is found and the tester puts the test case on hold to start investigating whether there’s an issue in the test case or in

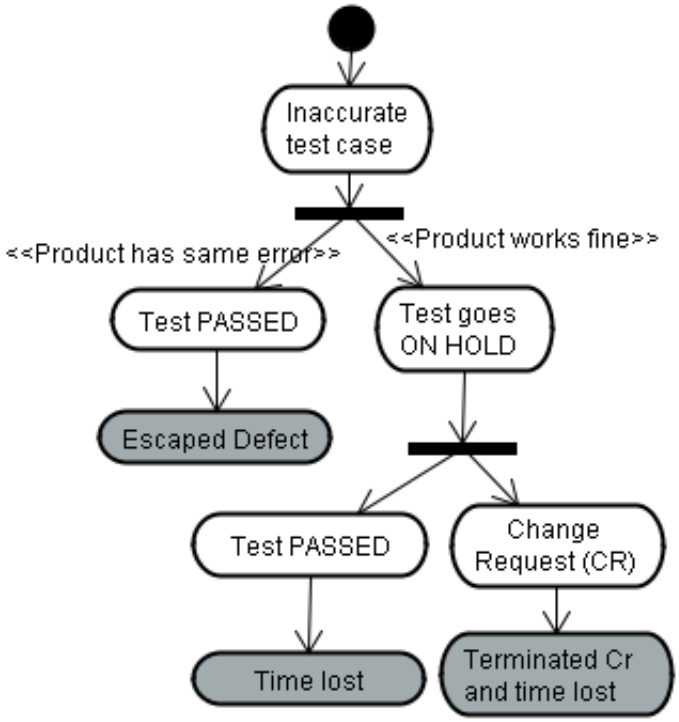

Figure 3.3 Possible consequences of generic test cases.

the product. He might search throughout requirement documents or eventually speak to the requirements analyst, to the development team or to other testers who already executed that test case. Then, if he finds evidence that the product works fine, he passes the test, writing an observation about the test innacuracy, and the consequence is time lost with the investigation.

In our experience, this kind of investigation can take a small amount of time, 1 or 2 minutes if, for example, the tester talks to a technical leader or a requirement specialist available personally or on instant messaging. On the other hand, it can take a lot more time if, for instance, the tester needs to look throughout use case documentation to find out about the right system behavior. Finally, the tester last resource is to contact the development team, having to wait for an answer. For instance, where we observed these scenarios, the development team worked in a different time zone, so the questions took longer than one day to be analyzed. Meanwhile the test case remained on hold and the tester moved on with the test suite.

Either way, if the tester can't find evidence about the correct product behavior he will assume that there is a product defect. He will report a change request (CR) and, when the development team gets to analyze the CR, they will get to the conclusion that the product works fine terminating the CR. In this case the consequences are time lost and a terminated CR which is a negative metric indicating that the tester reported a failure that didn't exist.

Therefore, the GT may impact SPL development with respect to two aspects, quality and productivity. Quality because some defects might escape, and productivity because of time lost during investigations and possible terminated CRs. The more often these inaccuracies appear on the test cases the more significative it is the impact on the test execution process. SPL

that contains more variation points are more likely to present such problems since maintaining and evolving test cases without the adoption of a test case derivation technique that manages variability is tiresome and error prone.

3.5 Specific Test Cases for SPL

To derive specific test cases for a given SPL there are some different possibilities. One alternative is to copy the same test document for each possible product line configuration and manually adjust the differences between them. However, this solution is not really appropriate because the more complex the SPL gets, the harder it is to maintain and evolve each product test document. Making it even more tiresome and error prone then using the single generic test document.

An alternative to obtain specifc tests is to reuse test cases for the different products in a given SPL. This reuse can be done in, at least, two different ways. First, we can use an SPL approach managing test cases variability and generating specific test cases for each product. Some of the existent approaches are PLUTO [BG03] and ScenTED [RKPR05]. The second alternative is to structure use cases using modularization mechanisms so that it is possible to generate specific use cases for SPL products. One existing technique for this matter is MSVCM [BB09]. After deriving specific use cases, these specifications can be used as input to a tool that automatically generate test suites. One example of this tool is TaRGeT [NTS+11].

Althought we didn't perform any evaluation concerning the specification effort to specify test cases using the aforementioned techniques, our natural feelingis that the alternative considering the reuse of test cases or the reuse of use cases takes more initial effort than the manually adjusted documents since the analyst needs to adapt the requirement documents to the new technique model. However, once this initial step is complete, maintaining and evolving specifications might become easier. Althought it would be interesting to compare the effort of generating specific test cases using each possible alternative from the point of view of the test specification process, this is not the focus of our work. Here we consider that the test cases are already specified as generic or specific and compare these techniques using the point of view of the test execution process. Different comparisons are suggested on the future works description in Chapter 5.

Having specific test cases for each product derived from the product line might help to solve the problems just described. This happens because the specific test cases consider the variability present in each product. More accurate test cases might help test execution impacting on both products quality and development productivity. However, organizations cannot decide to introduce new techniques or change their usual methods based only in assumptions. We can't simply assume that specific test cases will in fact bring such benefits. That's when empirical studies take place. They bring evidence that helps the decision-making process in an improvement seeking organization. This work has the main goal of providing this kind of evidence. In order to do that we have executed a serie of experimental studies that we present in the next chapter.

CHAPTER 4

Evaluation Studies

In this work we empirically compare generic and specific test cases evaluating each technique characteristics and implications from the point of view of the test execution process for SPL. Figure 4.1 illustrates this comparison. On the Figure left side, the generic technique provides one single generic suite that testers will use to test two different products (P1 and P2). Meanwhile, on the right side, the specific approach provides two different suites: P1 Suite and P2 Suite, each one specifying its respective product. Normally a SPL can generate much more than two products, but to fit this analysis into our experiment, since we had a limited amount of time to execute it, we chose two products that were the most representative from the chosen SPL considering the number of features and possible variations. In other words, we chose the two configurations with more features and most different from each other concerning alternative features.

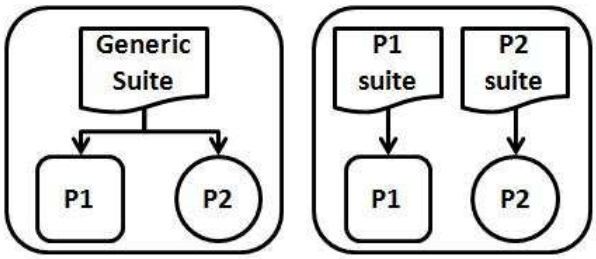

Figure 4.1 Generic and specific techniques.

In order to compare these two techniques we executed 5 experimental studies. Each experiment execution provided data that was compared to our set of hypothesis leading to knowledge achievement. The remainder of this chapter is organized as follows. Section 4.1 states the investigated problem in a more formal way. Next, Section 4.2 presents the raised hypothesis based on the problem statement. Sections 4.3 and 4.4 describe respectively the experiments design and operation. After that, Section 4.5 presents how we wrote the test suites. Section 4.6 presents a summary of the executed experiments. Next, from Sections 4.7 to 4.11, we present each experiment operation and results in more detail. Finally, Section 4.12 makes some considerations about the generalization of the achieved results.

4.1 Experiment Definition

We have structured the problem statement using the goal, question, metric (GQM) approach [BCR94] in order to collect and analyze meaningful metrics to measure the proposed process. The problem is described as follows.

Goal: analyze test execution effort, for the purpose of evaluating the effect of two different test case design techniques for SPL (GT vs. ST), with respect to their efficiency regarding time to execute the test suites and the number of terminated CRs reported during the test execution process. Using the point of view of test engineers and software engineering researchers in the context of experiments done with software engineering students in the environment of universities.

Research Question 1 (RQ_1): Does the ST reduce the test execution effort compared with the GT?

Metric: Test execution time

Research Question 2 (RQ_2): Does the ST reduce the number of terminated CRs compared with the GT?

Metric: Number of terminated CRs.

To evaluate the differences between the GT and the ST we chose test time execution and the number of terminated CRs metrics because, from what we have observed, the GT may decrease test execution productivity since testers might take longer to execute generic test cases besides reporting defects that don't exist. Considering the test execution process, we are interested in collecting time taken to execute the test cases, so, we won't collect, for example, the time taken by testers to set the initial conditions prior to the test case execution. We will also collect other metrics such as valid CRs, that is, CRs that report real product defects because we need to analyze all reported CRs in order to indentify the terminated ones.

Back on Chapter 3 we said that escaped defects could be another consequence of generic test suites. However, we didn't collect escaped deffects because we considered that it wouldn't make sense to inject defects and then specify test cases that described the defect as a correct behavior. Next section describes our experiment hypotheses.

4.2 Hypothesis

The objective of our experiment is to simulate a test execution environment using the GT and the ST in order to collect metrics for further analysis. Our hypothesis are described as follows.

To answer RQ_1 concerning the average of time consumed to execute the test suites:

$$H_0 \quad : \quad \mu_{TimeST} = \mu_{TimeGT} \tag{4.1}$$

$$H_1 \quad : \quad \mu_{TimeST} < \mu_{TimeGT} \tag{4.2}$$

To answer RQ_2 concerning the number of terminated CRs reported during the test suites execution:

$$H_0 \ : \ \mu_{TerminatedCRsST} = \mu_{TerminatedCRsGT} \quad (4.3)$$

$$H_1 \ : \ \mu_{TerminatedCRsST} < \mu_{TerminatedCRsGT} \quad (4.4)$$

Next section presents the design used during the experiment executions.

4.3 Experiment Design

Setting the proper design to the experiment execution is important because it ensures the use of the three basic principles of experiments design: replication, randomization and local control as we discussed over Chapter 2. In our context, the metrics evaluated are execution time and terminated CRs and the treatments are the GT and the ST. Furthermore, there are some factors that we need to control during the experiment execution as discussed below.

The first factor under control is the subjects selected to execute the experiment since different background knowledge and skills can influence directly on our metrics. Secondly, the features under test also need to be controlled because features scattered and tangled may produce more ambiguities in generic test cases than more focused features. In this work, when we say features we mean the artifacts (code, requirements) associated with a feature described in the SPL feature model.

In order to manage the subject and the feature factors we opted for a latin square design [BHH05] that has already been used in other software engineering experiments. For instance, Bonifácio has used this design to measure the effort required to create and maintain SPL specifications considering two different requirement variability management approaches for SPL [Bon10]. To apply this design to our considered factors, we dispose subjects in the rows and the features in the columns of the Latin square (see Table 4.1). In this way, we systematically use the three principles of experiment design as discussed over Chapter 2. Also, in each given row and column, the treatment, the GT or the ST, appears only once. Additionally, in each treatment activity, the subject executes two test suites, one for each product (P1 and P2), as illustrated in Figure 4.1. The order of the products is also randomized. This way, we can test two different products that are the most representative configurations of the SPL.

Considering the product under test, we believe that configurations with different quantities of variation points will not favor one or other technique. To understand this statement let's imagine the more extreme examples, the simplest and the most complex configurations from the same SPL. While using the GT to test the simplest configuration it is most likely to have problems with extra steps on the test case. However, when testing the most complex configuration, the tester will probably be confronted with wrong parameters and missing steps that were not well specified when new features were incorporated by the SPL. However, by choosing two products configurations as the most representative for the SPL, we can fully explore the variation points present on the SPL.

Table 4.1 Layout of experiment design.

	F1	**F2**		**F1**	**F2**	
Subject 1	GT	ST	**Subject 3**	ST	GT	
Subject 2	ST	GT	**Subject 4**	GT	ST	**...**
	square 1			square 2		

4.4 Experiment Operation

Each of the 5 experiments execution lasted three days, each day with a two hour session, as Table 4.2 shows. They took place in computer laboratories with one experiment conductor. We divided day 1 session in two phases. The first phase had the purpose of giving some training about black-box testing, giving a lecture about the SPL that they would test, explaining its functionalities and also a demonstration explaining the test cases format and how the participants should proceed while executing the test suites, collecting metrics and filling the CR templates. The main concern was to instruct the students to follow the test steps correctly, otherwise they might be tempted to explore the tool trying different flows and inputs and this is not interesting since exploratory testing is not the focus of this work. In addition, the experiment conductor required that the students should ask questions whenever they felt like and that they should address their doubts exclusively to the experiment conductor without asking questions to the other students nearby.

Table 4.2 Experiment execution.

Day 1	**Day 2**	**Day 3**
Training and Dry-Run	First round	Second round

The second phase consisted of a dry run activity. We asked the participants to execute a test suite with three simple test cases, collecting metrics and asking out questions. The experiment conductor monitored all the process. After finishing these activities, participants sent their results to the conductor email. We didn't use these results to run data analysis because the purpose of the dry run was to make participants understand the procedure while exercising it. We used this data just to analyze if the metrics were properly collected.

On day 2 and day 3 we respectively ran the latin square 1st and 2nd columns (Feature 1 and Feature 2), that we call here as rounds following Table 4.1 layout. In other words, on day 2 students followed Feature 1 test suites and on day 3 they executed Feature 2 test suites. During this process students weren't aware of the differences between the techniques neither which technique they were working with. We took this decision because instructing subjects about the two techniques could cause some noise since they could infer that one technique was better than the other impacting on the tasks execution. As the dry run activity, by the end of each round activities, participants sent to the conductor an email with their results.

4.5 Test Suites Design

We didn't use an SPL test derivation technique like PLUTO [BG03] or a test case generation tool such as TaRGeT [NTS+11] because we don't focus on studying the benefits of the test derivation techniques, but of the test execution techniques. Because of that, we manually wrote the test suites. For each experiment there were 6 test suites as illustrated in Figure 4.2. First, we produced the generic test suites for F1 and F2 (GS-F1 and GS-F2) trying to simulate the issues presented on Chapter 3. Some test cases presented steps that didn't apply for both products under test to simulate the scenario where the test case has more steps than necessary. Other test cases had some missing steps depending on the product configuration to simulate the scenario when the test case has less step than necessary. And finally some test cases presented wrong/missing parameters trying to simulate the case when the test case fails to specify some parameter value.

Next GS-F1 and GS-F2 were manually adjusted to attend P1 and P2 specificities, generating the suites SP1-F1, SP2-F1, SP1-F2 and SP2-F2. To exemplify the test suites differences, Table 4.3 describes one step of the generic scenario that asked the tester to check whether the options for generating reports appear correctly. However not all products contain these two formats (PDF and Bibtex) so this parameters are wrong for one of the configurations. This step was adjusted to specify the behavior of a product configured to support only PDF reports generation as described in Table 4.4.

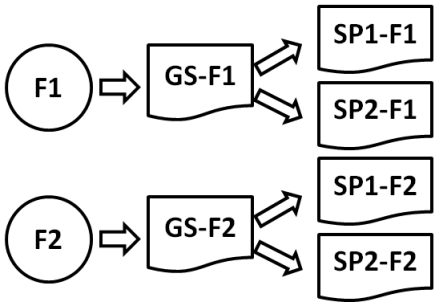

Figure 4.2 Test suites.

The number of test cases in each test suite varied according to the time available in each experiment execution. On the first experiment there were 7 test cases per suite and the time was short to execute this amount of test cases. Because of that, on the remaining experiments each suite contained 6 test cases. These cases cases aimed at testing the features essential functionality and they will be further discussed over this chapter. We chose this number because each latin square round had in average two hour and the testers has to execute two suites giving a total of 12 tests which was a reasonable amount of test to execute in the interval of two hours.

4.6 Experiments Outline

This section describes the summary of the executed experiments. In total we executed 5 experiments. The first one was a pilot study which we used as an experience for further experiments

Table 4.3 Generic test case.

User Action	System Response
Verify the options for the report generation format	The options (PDF, Bibtex) are available.

Table 4.4 Specific test case.

User Action	System Response
Verify the options for report generation format	The option (PDF) is available.

and the last experiments were executed with different purposes aiming at refining the hypotheses and experiment planning. Each study brought a new light on understanding different things about metrics collection methods, noises and the impact that the techniques had on test execution.

However, we believe that some experiments brought more reliable results than others. The first and the third rounds of experiments had some unwanted effects that affected the outcome of the experiment. Nevertheless they were important in the process of experimentation because of the lessons learned that we describe throughout this chapter. On the other hand, the second, fourth and fifth rounds of experiments brought some interesting results that we describe later on this chapter. Also, the material used the the data collected in the executed experiments are available online [Acc12].

4.6.1 First Experiment (Pilot)

We wanted first to experience how metrics would be collected, how participants would behave while completing the tasks and learn about other factors that we would need to control. Consequently, more than achieving results, we needed to gain some experience in order to improve the planning of the next experiments rounds.

The experimental case we selected was the Test And Requirements Generation Tool (TaRGeT)
[NTS+11] that is an SPL whose main functionality is to generate test suites. The participants manually excuted the test suites and collected time by registering time in the test suites spreadsheet before starting a new test case and after concluding it. This approach was not accurate enough because we needed to have a better control of time collection. Besides that we also had some problems related to the selected experimental case. TaRGeT is not an easy tool to understand in such a short period of time and the participants got confused about some of its specifications and features.

Because of these problems we don't extend this discussion to show the analysis of the data collected because the results are not reliable. However the execution and the experience gained in this experiment are further discussed on Section 4.7.

4.6.2 Second Experiment

In the second experiment, we corrected the problems identified during the first experiment execution. We adopted a tool called ManualTEST [AB08] to collect time. In addition, we changed the experimental case to one more adjusted to our needs. We chose to use the Research Group Management System (RGMS) that is an SPL with the purpose of managing researchers information and their associated publications.

Using the ManualTEST students could read test suites and play/pause the test execution whenever they needed. We asked them to start execution by pressing the play button on ManualTEST before starting the test case and pausing only to report CRs. Using this tool we were able to collect metrics in a more accurate way.

We analyzed the metrics resultant from this experiment and they brought some evidence that specific test suites can increase the productivity of an SPL test execution environment by reducing both the time for test suites execution and the number of terminated CRs. Section 4.8 details this experiment execution and results.

4.6.3 Third Experiment

After the second experiment execution, some discussions emerged considering the approach used to collect test execution time. Like explained before, the participants only paused the time to collect CRs but, while executing test cases, they investigated possible defects by asking questions directly to the experiment conductor. Consequently, the participants collected the time for execution and investigation together.

Considering this approach we thought of two arguments that can bring potential *bias* to future replications of the experiment. The first one is that the results achieved in this experiment presented results that could be generalized only to test execution environments where a “test suite specialist" is available at all times to answer testers questions. Back on Chapter 3 we said that the investigation time can vary a lot depending on the structure of the test execution environment. Secondly, we noticed that this approach is not reproducible since the experiment conductor can add *bias* to the results if, for example, he takes longer to answer questions favoring one or the other technique. In the previous experiment we proceeded with caution to mitigate this risk, answering questions right away. Because there were only 9 participants it was easier to manage the questions without compromising time collection, however, when working with a bigger group, 20 participants, for instance, it is harder to manage all the questions at the same time.

Because of these arguments we decided to change the experiment planning so that we could collect execution and investigation time separately. This way we could achieve more reliable results. However, during this new experiment execution we had some inconveniences that prevented us from collecting metrics properly. The main cause of these problems was the lack of the dry run activity on day 1 execution. For this reason we weren’t able to draw valid conclusions from the third experiment execution. However, we discuss in more details the experience gained during this experiment in Section 4.9.

4.6.4 Fourth Experiment

For this experiment we replicated the third experiment planning, but this time we didn't have any inconveniences and the execution flowed normally. After assessing the data, the results indicated that, putting the investigation time aside and considering only the time for test execution, the GT approach can decrease test execution process productivity. The fourth experiment, compared with the other executions was the one that brought the most reliable results. We present its details in Section 4.10.

4.6.5 Fifth Experiment

After the fourth experiment execution, when we gathered evidence about the GT and the ST impact on test execution time, we wanted to collect time differently, considering the time for CR report as well. The rationale for this decision was that, in a real test execution environment, when the tester reports an invalid CR due to the test inaccuracies there is an associated waste of time. During the previous experiments we only raised the number of terminated CRs but we didn't consider the time for reporting invalid CRs since the participants had to pause time for reporting CRs.

To analyze this new approach we planned the fifth and last round of experiment. This time we consider test execution process as a whole, that is, collecting time for test execution and CR reports. Like on the second and fourth experiment the ST presented better results than the GT. Section 4.11 discusses this experiment execution and results.

4.7 First Experiment (Pilot)

To evaluate the techniques, we considered products and test suites from the Test and Requirements Generation Tool (TaRGeT) [NTS+11] SPL. TaRGeT products main goal is to read a formal requirements description, in the layout of use case scenarios, of the system under test and automatically generates test suites based on a set of test generation directives. This tool aims to increase productivity by automating a systematic approach for dealing with requirements and test artifacts in a integrated way. In Figure 4.3 we can see TaRGeT simplified feature model, considering the features involved in this experiment. Next we briefly present these features.

TaRGeT main functionality is the automatic generation of test suites from use case scenarios written in natural language. In addition the test case generation can be done in two different ways. By pressing a button when the product is configured with the *Basic* test generation feature or in a *On The Fly* fashion, that is, users can access the generated test suite at the same time he edits the test generation directives.

The *Use Case Editor* is an optional feature that allows the creation and edition of use cases inside TaRGeT environment, without needing to use another tool for text documentation. This feature contains some important information such as step ID, user action and system response. In addition, use cases also have name, description, initial setup, a list of Requirements to relate to and main and alternative flows.

When requirements evolve it is necessary to regenerate the tests. When this happens, infor-

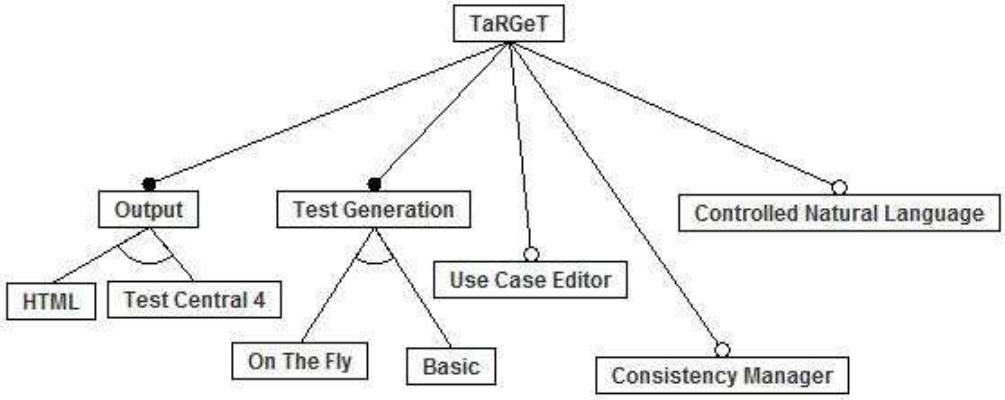

Figure 4.3 TaRGeT simplified feature model.

mations, such as test description, test objective, and the previous selection of test cases get lost. Additionally, the evolution of use cases can generate new test cases with ids that were already in use by other test cases in an old suite. The optional feature *Consistency Management* (CM) was conceived to solve this problem and to maintain the consistency of different test suites generated over the time.

The consistency management starts when the user tries to save the new generated test suite. Then TaRGeT displays a list of test suites that the user generated before to compare with the new one. The user selects one of these tests suites and starts the comparison between them in order to find a relationship between new and old test cases. When the comparison is done the user can visualize test cases that have a certain percentage of similarity (default option is 95%) to associate new and old test cases. Then the system merges the two test cases in the new test suite keeping any information that the user has added in the suite and preserving the tests selection.

TaRGeT uses as input use cases written in natural language, but this notation sometimes can be ambiguous and inconsistent. The *Controlled Natural Language* (CNL) is an optional feature that aims to help the process of text verification and minimizes the possible mistakes in the testing phase. CNL is a subset of an existing natural language. It defines some writing rules and a restricted vocabulary to avoid users to introduce ambiguities into their texts. Also CNL can be useful to define a standard to be followed throughout an organization.

TaRGeT can be configured to generate test case suites in different formats, but for the purposes of this experiment, we used products configurations containing only HTML and XLS formats.

In conclusion, TaRGeT is a software designed to increase productivity by dealing with use cases writing and automatic generation of test suites is an integrated way. Additionally, it also provides test selection according to different criteria filters, guidelines for use cases writing (CNL) and mechanisms that helps to maintain the consistency between test suites.

4.7.1 Products Selection

To execute the experiment, the TaRGeT development team recommended the selection of two products as beeing the most representative configurations available for the SPL. These configurations are presented next.

P1: *Test Generation (On The Fly), Use Case Editor, Output (HTML), Consistency Management, Controlled Natural Language*

P2: *Test Generation (On The Fly), Use Case Editor, Output (Test Central 4)*

To design two different test suites we chose features *On The Fly Test Generation* and *Use Case Editor* features because they include the core of TaRGeT funcionality that is edit use cases and generate test suites based on them. Each test suite had 7 test cases. We chose this number of test cases because each latin square round would take about 2 hours and each participant had to execute two test suites on two products as explained on Section 4.3. That gives a total of 14 test cases in two hours.

We distributed the test suites in a spreadsheet format and participants had to fill in the following information: the time before starting and after concluding each test case, the test case result ("Passed" or "Failed"), and, in case the test case failed, the id of the reported CR. To report CRs we provided a simple template to fill in the following data: CR id, test case id, test case step that failed and a brief issue description.

4.7.2 Participants

In total 7 subjects engaged in this first experiment execution. They were computer science ungraduate students from different universities in Recife, engaged with the Software Productivity Laboratory Network (LabPS), a project with the purpose of complementing students education by providing different software engineering specialization courses including software testing. This project is sponsored by the National Institute of Software Engineering (INES) [oSE12]. The students had different skills in software testing, in fact, some students already had worked with TaRGeT while others didn't.

The 7 participants were randomly assigned in pairs to form 3 replicas of the latin square. The 7th student executed the activities but didn't have its results analysed. To form the first latin square replica, the features (*On The Fly* and *Use Case Editor*) were randomly assigned to Feature 1 and Feature 2 and then, finally, the treatments were randomly assigned inside the square. Then we replicated the first square treatments configuration to the other 2 replicas.

4.7.3 Lessons Learned

This experiment occurred on January of 2011 and we executed it in three days, each day with a two hour session, as Table 4.2 displays. The experiment took place in a computer laboratory with one experiment conductor. One week before the execution we installed TaRGeT products in all laboratory computers so that subjects would have a similar equipment to work with.

In brief, during this first experiment execution we identified a series of factors that had influence on our metrics collection. Althought we don't discuss the experiment results analysis here, the metrics are available online at our website [Acc12]. Next we discuss these factors.

4.7.3.1 Time Collection

When we asked students to collect time by registering the beginning and the ending time of each test case in the spreadsheet, we didn't realize that there were different types of activity related to test execution and that some of them were not interesting for our purposes. We listed these activities as follows:

- Execution: time subjects were actually focused on test case execution following test scripts, observing system behavior and asking questions to the experiment conductor.
- CR report: after finding a defect, subjects had to stop test execution and take some time filling out the defect report before recording the CR id on the spreadsheet, failing the test case and finally registering the end time on the spreadsheet.
- Pauses: even though we recommended that once the subject had started a new test case they shouldn't pause the execution, there was no guarantee that they would strictly follow this recommendation. Because we worked with a small group (7 participants) it was easier to monitor their activities. But in a larger group it would be difficult to notice if, for instance, a participant paused the execution to check emails, answer the phone or if they had any other distraction.

To our next round of experiments we used a tool to collect time properly so that students could play and pause test execution to report CRs and to do other activities.

4.7.3.2 TaRGeT Problems

When we chose the TaRGeT product line as our experimental case, we didn't realize that it served a really specific type of user such as requirements or test engineers. Some features like the *On the Fly Test Generation*, the *Controlled Natural Language* and the *Consistency Manager* were not so trivial to understand after the lecture given in the training session. So, if we wanted to replicate this study using TaRGeT we would have difficulties to recruit and train participants.

In addition to the difficulty of understanding the tool purpose, some students had difficulty with the language because TaRGeT and its specifications are written in English. In fact we observed that during test execution some students were trying to translate the test case on the internet and this time also counted as test execution.

In conclusion, to control those effects in future experiment rounds, we need to choose a different product line with a more general and simple purpose, that the participants can understand its overall functionalities after the lecture and the dry run activities. Also, it is preferable that its test cases are written in Portuguese.

4.7.3.3 Similar Test Cases for Both Features

Finally the last effect that we would like to avoid in our next experiment is the tool learning between rounds. This happens because feature 2 had test cases with a lot of steps in common with the test cases from feature 1. As a result we believe that students may have learned this steps leading to a faster test execution on the second round of the latin square. To control this effect we have to design test suites for each feature with separate flows and steps.

4.8 Second Experiment

Like discussed before, collecting time by registering the beginning and the ending time of each test case is not accurate enough for our purposes, so we adopted a tool called ManualTEST [AB08]. Besides that, we came to the conclusion that TaRGeT product line was not a good tool for the experiment purposes because it is not an easy tool to understand and its specification is written in English. These factor impaired some subjects execution because they were not so familiar with English. To correct these problems we adopted as experimental case the Research Group Management System (RGMS) [cLdPdS12] product line. Over the next sections we describe ManualTEST and RGMS.

4.8.1 ManualTEST

According to Aranha and Borba *"ManualTEST (Manual Test Execution assiSTant) is a tool developed to automate part of the data collection procedures of empirical studies involving manual test execution"* [AB08]. Its main goal is to improve the accuracy of data collection, identifying possible sources of problems occurred during these studies.

Figure 4.4 presents ManualTEST main interface for test execution. On the left side of the screen, testers read the test case steps, moving forward and backward using the keyboard up and down keys to select the step they're executing at the moment. On the right side, the interface works as a control panel for the time collection. Testers fill in the test id and the tester id. It contains a play/pause button which collects the time spent to execute test cases. Finally it has buttons to report the test case result. There are three possible options. Passed, failed and blocked. In this experiment we used only the first two options.

In a real test execution environment the blocked status is used when the task described in the test case can't be performed probably due to a defect that has already been reported. But to simplify the experiment execution, we didn't use this option. The participants had to press the passed button when everything was ok with the test case execution. On the contrary, if they found a defect while executing the test case, they reported the CR, filled the CR id in Observation field and pressed the failed button.

ManualTEST registers the time based on the three following stages: setup, execution and debug. Setup is the time taken to prepare the environment and the system to satisfy the test case pre-conditions. For example, if the test case requires that the user is already logged in the system, the tester uses the setup time while logging in the system. Execution relates to the execution of the test case steps, and that is the interval that we're interested in collecting. Debug it's the stage in which testers report CRs. Also it has a play/pause button so that the tester can interrupt the test execution without compromising time collection. While testers are executing the activities, ManualTEST records their results in a spreadsheet.

4.8.2 Research Group Management System (RGMS)

We chose an SPL called Research Group Management System (RGMS) that was developed by students as part of a project from a software reuse discipline. It consists in a Java desktop product line with the goal of managing members, publications and research lines associated

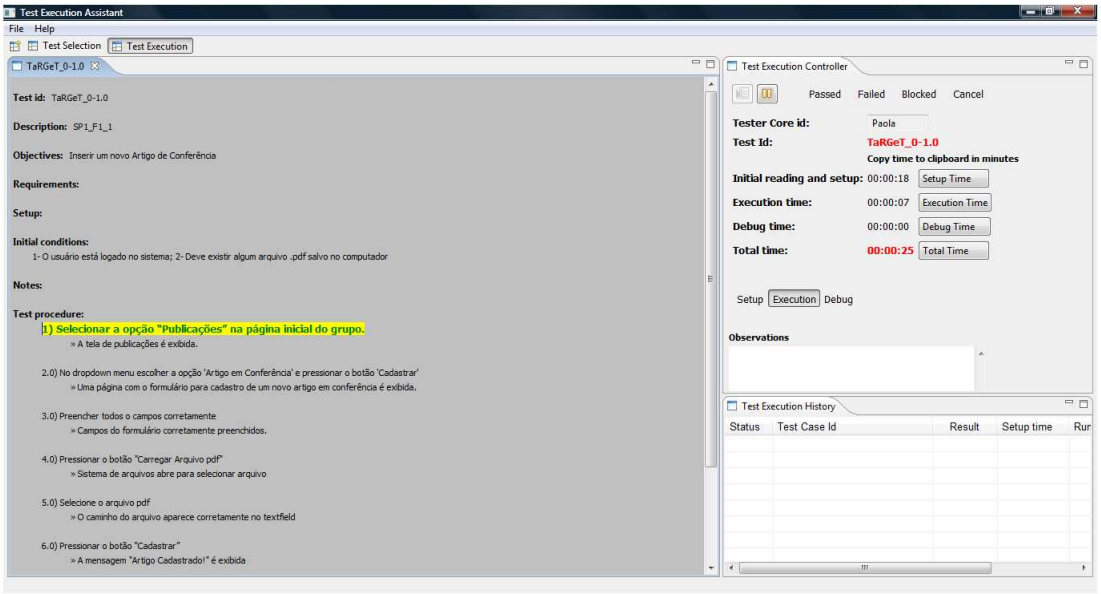

Figure 4.4 ManualTEST interface.

with a research group. It is an information system that registers entities saving then in a database and generate different reports and searches options. In addition the tool and its specification were written in Portuguese. These two characteristics helped to fix the problems that we had with the TaRGeT product line. Figure 4.5 exhibits RGMS feature model.

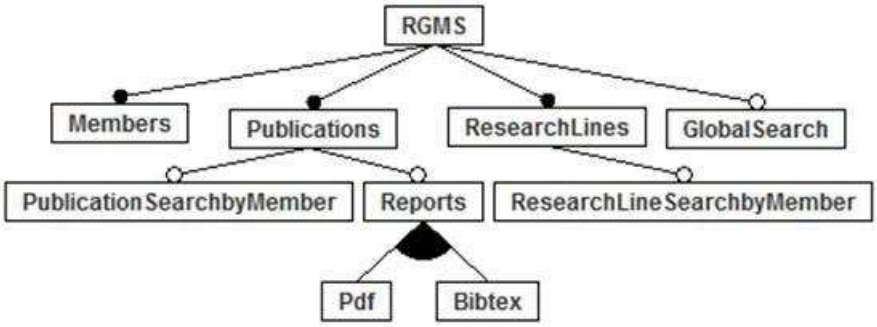

Figure 4.5 RGMS feature model.

Members, *Publications* and *ResearchLines* features are responsible for managing entities insertion, search and deletion from the system. Meanwhile, the *Reports* feature is responsible for generating publication reports in two different formats: PDF or Bibtex. Also *ResearchLineSearchbyMember* feature makes it possible to know which research lines associates with each member registered on the system. Similarly, *PublicationSearchbyMember* retrieves the publications associated with the members of the research group. Lastly, *GlobalSearch* retrieves members, publications and research lines.

In RGMS there are 32 possible configurations. To work in this experiment we chose 2 configurations (P1 and P2) that are the richest (with more features) and most different from

each other (with respect to alternative features). This way we have configurations with different variation points so that we can explore the differences between generic and specific test casese. The products, P1 and P2, were configured as described below.

P1: *RGMS, Members, Publications, ResearchLines, Reports (PDF, Bibtex), ResearchLineSearchbyMember*

P2: *RGMS, Members, Publications, ResearchLines, Reports (PDF), PublicationSearchbyMember, GlobalSearch*

The features chosen to design the test suites were *Publications* and *ResearchLines* because they were rich and contained different flows to explore and also sufficiently independent from each other generating test suites with separate flows. We didn't choose the *Members* feature because it was really tangled in both products and each feature should have different flows of execution, in order to avoid participants from learning the tool.

For designing the generic test suites, we considered the differences between the P1 and the P2 configurations. We wanted to design test cases that explored the variation points contained in both configurations. Each generic test suite contained 3 or 4 inaccurate steps. For example, to specify test cases that had more steps than the necessary, one test case explored a flow where the tester had to use the *ResearchLineSearchbyMember* button, but the P2 configuration doesn't contain this feature. To make test cases with less steps than necessary, one test case asked the user to press the button for reports generation without telling which format to choose. This flow works for the P2 configuration since it has only one possible format for generating reports (PDF), but it doesn't work for the P1 configuration because before pressing the button, the user has to choose between PDF or Bibtex formats using a radiobutton. Finally, one example of wrong parameter considered a test case where the user had to check that the report was successfully generated in the Bibtex format, but the P2 configuration doesn't have the option to generate reports using this file extension. In total, the generic suite for the feature *Publications* contained 4 points of inaccuracies and the generic suite for the feature *ResearchLines* contained 2 points of inaccuracies. The test suites used in this experiment are described in the AppendixA.

4.8.3 Participants

Altogether, 9 students participated in our second experiment. So we randomly arranged 4 latin squares replicas and 1 student performed the experiment as a backup. From this total, 3 of them were ungraduate students involved with the LabPS activities. The other 6 subjects were all computer science postgraduate (doctorate or master) students from the Informatics Center that volunteered to participate.

4.8.4 Experiment Operation

The experiment operation occurred on September of 2011 in a similar way of our first experiment. Since we had 9 subjects, we randomly assigned them to form 4 latin squares and the 9th student executed the activities together with the others and we considered him as a backup. It took place in the same laboratory of the first experiment and it also had three sessions of two hours each. The first session contained the training and the dry run activity and the subjects had to answer a small questionnaire about their previous experience with software testing.

On day 2 and day 3, subjects were asked to execute test cases using the P1 and the P2 and collecting metrics using ManualTEST. To report CRs they used the same template from the first experiment. Testers were instructed to press the debug button on the ManualTEST to fill in the defect report, then put the CR id in ManualTEST observation field and, lastly, fail the test case. Students could also pause the execution at any time to answer the phone, or any other necessary activity, but they couldn't pause to ask specific questions about the test cases because we wanted to collect also the time of making questions and obtaining answers.

4.8.5 Data Analysis

After collecting the metrics, we wanted to draw some valid conclusions based on this data. So, by the end of the experiment execution, we downloaded participants results from the conductor email to start the analysis. However, we noticed that one of the subjects allocated in one of the squares did not collect time correctly. According to his results, he reported a CR in a six seconds which was highly unlikely. We believe he had difficulties using ManualTEST to collect time (this issue will be further discussed on the threats to validity section). Because of that we replaced him for the backup student results which was coincidently assigned with the same activities sequence. After this replacement we initiated data analysis.

4.8.5.1 Time Analysis

We used the R Statistical Software [R D08] to perform the data analysis in our experiment. We took all spreadsheets with students results and manually formatted them summing up execution time (in minutes) for each test case to get total time for both test suites executed in each latin square round. This formatted document was used as input to R software. In Appendix B we present the format of this input document.

In order to interpret data we first carried on a descriptive analysis to observe data distribution based on some characteristics such as data dispersion and the median value. For this purpose, we plotted the box plot graphic [JP05] that appears in Figure 4.6 and, as we can observe, the ST values tend to be smaller than the GT values. The GT median value is close to ST upper quartile (Q3). In addition, the GT values are more dispersed than the ST responses. Also, Table 4.5 describes the average and standard deviation for the execution time observed in both techniques.

Table 4.5 Second experiment average and standard deviation.

	Generic	**Specific**
Average	1534	1315
Standard Deviation	504	440

In addition to the median values, we wanted to compare the observations according to each subject results. Figure 4.7 shows the individual performances according to the technique. From 8 students, 7 finished the specific test suites faster than the generic test suites. Only subject 8 finished the generic suites faster than specific ones.

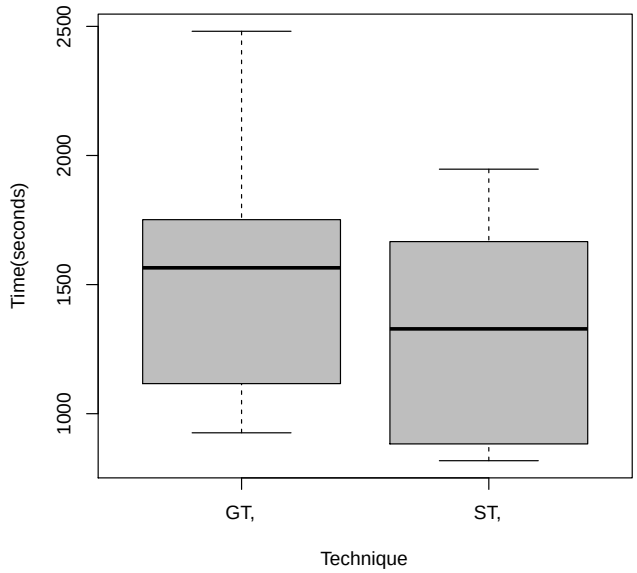

Figure 4.6 Second experiment box-plot graphic.

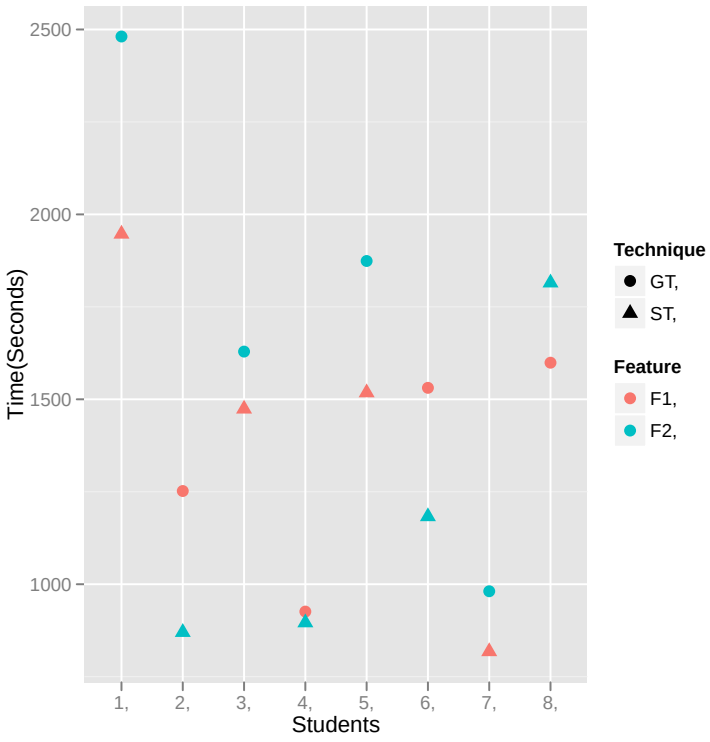

Figure 4.7 Individual results for each technique.

$$Y_{lijk} = \mu + \tau_l + \tau\alpha_{li} + \beta_j + \gamma_k + \varepsilon_{lijk}, \text{ where}$$

Y_{lijk}	response of the l_{th} replica, i_{th} student, j_{th} SPL, and k_{th} treatment
μ	average of the response data
τ_l	effect of the l_{th} replica
$\tau\alpha_{li}$	effect of the interaction between the l_{th} replica and the i_{th} student
β_j	effect of the j_{th} SPL
γ_k	effect of the k_{th} technique (GT or ST)
ε_{lijk}	random error

Figure 4.8 The regression model of our experimental design.

Moving on with the analysis, we used the linear model described in Figure 4.8 that considers the effect that the different factors had on the response variable (time). But before running the statistical test we ran the Box-Cox transformation technique [Sak92] that consists in a parametric power transformation technique that aims to reduce anomalies such as non-additivity and non-normality. The curve obtained by Box Cox is illustrated in Figure 4.9. To reduce such anomalies the curve needs to present its maximum value between 0 and 1 and, as we can see, this constraint is satisfied in this curve. Because of that, there was no need to apply a transformation in our results.

After using Box-cox technique we ran Tukey Test of Additivity [BHH05] that checks if the effect model is additive. An additive model ensures that the interaction between factors displayed on the columns and rows of the latin square won't significantly affect the response. To run this test we have the following hypothesis:

H_0: The model is additive

H_1: H_0 is false

Tukey Test of Additivity returned a p-value of 0.475 which doesn't give us enough evidence to reject H_0. Consequently we considered our model to be additive.

Finally we ran the ANOVA (ANalysis Of VAriance) [WRH+00] test to compare the effect of both treatments on the response variable. This test returned the p-values described in Table 4.6. For the purpose of our analysis, the most important data is the estimated p-value associated with the technique factor. In order to avoid Type 1 errors, which means rejecting the null hypothesis when in facts it is true, we follow the convention of considering a factor as being significant to a response variable when $p\text{-}value < 0.05$ [BHH05].

In Table 4.6 the p-value associated with the technique factor is approximately 0.04, satisfying the aforementioned condition. Consequently we are able to reject our null hypothesis that stated that the techniques didn't have a significant effect on the response variable. Considering that the alternative hypothesis is true (the average time to execute test cases is different for both techniques) we can see from the box plot and from the individual results graphics that there is evidence that specific suites for SPL can reduce time for test execution. In the next section we show the results for the terminated CRs.

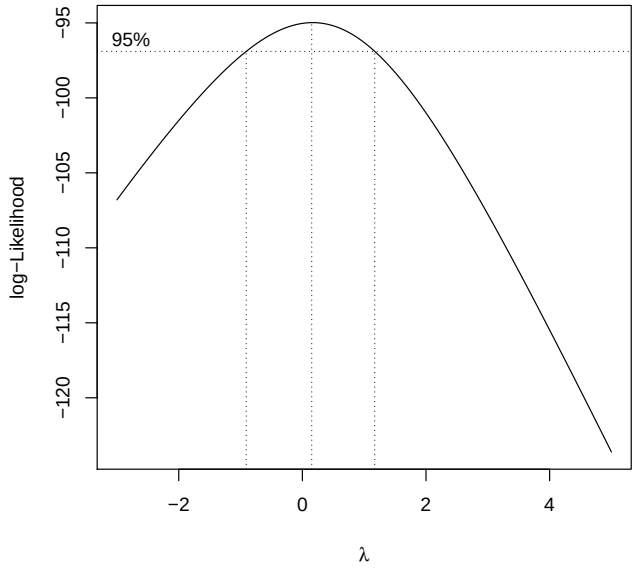

Figure 4.9 Box-cox technique.

Table 4.6 Second experiment ANOVA results.

	Df	Sum Sq	Mean Sq	F value	*p-value*
Replica	3	431280	143760	5.0998	**0.0434014**
Replica:Student	4	2506626	626657	22.2304	**0.0009623**
Feature	1	27556	27556	0.9775	0.3609867
Technique	1	191844	191844	6.8056	**0.0401876**
Residuals	6	169135	28189		

4.8.5.2 Terminated CRs Analysis

Remembering our second research question (RQ2), we wanted to investigate if the GT would increase the number of terminated CRs. This could happen because the tester would get confused interpreting a test inaccuracy as a defect and reporting a CR that, in a real test environment, would be terminated. For the CR analysis there won't be more accurate statistical tests because there were many observations with value equals to zero. This means that, during the test suite execution, the participant didn't report any CR. It is difficult to analyze patterns in a set of observation containing this considerable amount of zeros, so, we will simply compare number differences between techniques.

To evaluate the status of the reported CRs, first we read all CRs descriptions and then classified them into the following categories: valid, invalid and duplicated. A valid CR describes a real defect on a product. Whenever the same participant described the same issue with more than one CR the second one was considered duplicated. In a real test execution environment, before reporting a CR, the tester needs to investigate in the CR system if that defect has already been reported, if he doesn't do that, and report a CR describing a defect that has already been reported, his CR will be classified as a duplicate of the first one. However, in this experiment we didn't have a CR system to report and search reported CRs. Every tester had the liberty of reporting CRs whenever they found a defect, this way we could compare the numbers of terminated CRs individually and considering the different techniques. Finally, an invalid CR represents the scenario that we're investigating: the subject reported a CR that didn't exist because of a test case inaccuracy.

After assessing and classifying the CRs status, they are summarized in Table 4.7. For valid CRs there was the same number for both techniques. However 2 invalid CRs were reported on the GT. They were reported by two different students and reported the lack of the Bibtex report format in a product configuration that didn't contain this option. Besides the fact that there was an experiment conductor available all the time to answer questions, still two testers got confused by test cases inaccuracies and reported product defects that didn't exist.

Table 4.7 Resume of reported CRs

	Valid CRs	Invalid CRs
ST	12	0
GT	12	2

4.8.6 Threats to Internal Validity

In summary, in this second round of studies we were able to avoid some threats learned during the first experiment and to draw some interesting conclusions. Nevertheless some noises were identified.

4.8.6.1 ManualTEST

At first ManualTEST seemed a good solution to collect time while executing test suites. But it didn't turn out this way after the experiment execution. This happened because ManualTEST presented some faults that confused testers. For example, in order do begin a new test case, the student had to close the test execution tab, select a new test case and open the test execution tab again, otherwise the chronometer would continue from when it had stopped on the last executed test case. As a result, subjects sometimes interrupted test execution because they had difficulties using this tool.

We were able to mitigate problems with ManualTEST by giving a more detailed training. In spite of that, ManualTEST turned out to be too complex for our purposes. It collects setup, execution and debugging time when we are only interested in collecting the execution time. Because of that, we decided to implement a simpler tool that would collect execution time according to our needs.

4.8.6.2 Time Collection

We avoided the first experiment time collection approach problem by using an accurate tool to register the execution time. However some new discussions emerged considering the new time collection approach. We instructed participants not to pause execution time for asking questions about the test cases. We wanted to check if the time for execution and investigation of both techniques would influence the results. But two important considerations must be made about this approach.

First, the experiment conductor can easily mask the results to favor one of the techniques. To do that, he can take more time answering students that were executing one technique or the other and there would be no means to prove that. Since we want to replicate the experiment we would need a more sound approach. We were able to mitigate this risk because there was a small amount of participants and also we were careful to answer the questions right away and using uniform answers for all participants. Because we had a small group (9 participants) we didn't have problems with participants asking at the same time, however, with a larger group it would be more difficult to manage the questions and the *bias* would be inevitable. Second, the results would be valid only for test environments that had a "test suite specialist" available all the time to answer the testers questions. This situation is not very common in our experience. To correct these aspects we came up with a new approach for time collection that we describe further in the next section.

4.9 Third Experiment

With the purpose of improving the soundness of our results and implementing a more replicable approach we have planned and executed the third round of the experiment. Basically the main differences between the last study and this one are on how students collected time.

4.9.1 Instrumentation

As discussed over the last section, the students had some difficulties using ManualTEST for collecting time. To begin with, this tool is too complex for our purposes which is collecting only the execution time. Second, it had some defects that impaired some students execution. For this matter we have implemented our own solution for time collection, the TestWatcher tool.

4.9.1.1 TestWatcher

TestWacther is a simple tool for time collection with a single interface as illustrated in Figure 4.10. On the top, the interface shows two fields, the first one should contain the tester ID and the second one the test case ID. Then it displays a play/pause button with a timer and lower there is an observation field, so the tester can fill in the CR id or any other comments concerning the test case. Finally, on the lower part of the interfae there are the passed and failed buttons to record test case results.

Figure 4.10 TestWatcher: time collection tool.

Whenever the tester presses the passed or the failed button, TestWatcher registers a new row in a spreadsheet with the time and date of the test execution, tester name, test case id, execution time (in seconds), test result, how many times he paused test case execution and how long each pause took. Because TestWatcher doesn't show the test case inside its interface, we presented the test suites using spreadsheets like on the first experiment. This approach was not so practical since students had to press play/pause using TestWatcher, read the steps from the spreadsheet and execute the steps using the RGMS product. However, because RGMS interface

didn't take the entire computer screen it was possible to leave the three components open at the same time. So we instructed the students to work like this.

4.9.1.2 Time Collection Approach

Back on Chapter 3, we presented the activity diagram in Figure 3.3 that considers the consequences of generic test cases in a test execution environment. When the tester puts the test case on hold to investigate whether there is a defect on the product or just a test inaccuracy, he does different tasks depending on the environment structure. He may simply go and talk to an immediate superior, such as a technical leader, or contact a developer through instant messenger or he will study throughout some requirement documents.

To consider and simulate all those situations we worked on the following approach to simulate the investigation time: whenever the tester had to investigate if there was a defect on the product, he paused test execution using the TestWatcher and would ask the experiment conductor, who had to know previously the variation points on the test cases that could cause some misunderstanding, this way, we mitigated the risk of the conductor giving a wrong answer. If there was a test inaccuracy, the conductor instructed the tester to ignore the issue, continue with the test execution and write an observation in the TestWatcher, telling what was wrong with the test, for example, if a step didn't apply for the product under test. Otherwise, he would tell the tester to report a CR. After that, the conductor would take note of the interruption. By the end of the experiment, the conductor would have a report about how many times testers interrupted execution because of test inaccuracies.

The purpose of having the number of interruptions is that we could first run the analysis considering only the execution time, without the investigation time. This analysis would bring results that indicated whether the different techniques differ from average test execution time. Then, if the average time was the same for both approaches we could introduce the investigation time by adding small time intervals, 60s for instance, for every interruption caused by participants investigation.

4.9.2 Participants

The subjects to our third study were 20 students from computer science and computer engineering undergraduate program in Brasilia University. They were subscribed on the discipline of object oriented programming. They didn't have any previous knowledge of software testing and would engage in the experiment as an extra activity in exchange of a lecture about black-box testing. Also, they were randomly assigned into pairs to form 10 latin square replicas.

4.9.3 Experiment Operation

Compared with the first two experiments, this one had some operational differences. Since the students didn't have a lot of experience with software engineering, on day 1, instead of having a short training and the dry run, there was a more detailed lecture about scenario specification and black-box testing and then giving a demonstration about how they would conduct the activities step by step, observing system behavior, collecting time with TestWatcher and reporting CRs.

When this lecture finished there was no time left for running the dry run.

Then on day 2 and day 3 the experiment moved on normally except that the students weren't so motivated with the activity, maybe because it was an extra activity that wouldn't count to the discipline evaluation. On day 2, 14 students attended to class and on day 3, 12 students attended. But only 9 students attended to both classes completing the activities. Besides that, we noticed that the lack of the dry run activity made students feel confused at the beginning of day 2 even after a new demonstration of the experiment procedure. Althought we didn't present the results of this experiment in this work, the metrics are available at our website [Acc12].

4.9.4 Lessons Learned

In brief, while running this experiment, we were able to test our new time collection tool with success and also to validate our approach to collect execution and investigation time separately. Nevertheless, there was no time to apply the dry run activity in this experiment. As a result, students didn't get the practice of the procedure prior to the experiment execution. Some subjects presented some difficulties to understand what they needed to do on day 2. These difficulties reflected on the students results since two of them didn't collect time properly on day 2. Because of that, we had to reduce our data set, that already reduced by low student attendance to classes, from 9 students to 7 students.

Besides reducing the data set, we noticed that the lack of the dry run may have had a negative impact on day 2 execution time because students had to take some extra time trying to get familiar with the procedure of the experiment execution. Since they took this extra time on day 2, perhaps they tended to execute day 3 activities faster. Anyway, the lesson learned in this experiment is that the dry run activity is essential to our purposes because subjects can practice what they have to do prior to the metrics collection. This way the experiment conductor can observe if there's anything wrong with data collection previous to the experiment execution.

In conclusion, the lack of the dry run impacted the metrics collection, consequently we don't extend the presentation of this experiment to show the data analysis. However, we planned a new experiment round to correct this mistake and bring reliable results. The results of this new planning are discussed next.

4.10 Fourth Experiment

Over the last experiment we improved the planning to analyze metrics in a more accurate manner by collecting execution and investigation time separately. We did that by registering the number of interruptions that the subjects made to ask questions to the experiment conductor so that, by the end of the experiment, we could add the investigation time on the analysis phase. For this experiment we used exactly the same material that consisted of test suites, RGMS products and the TestWatcher tool that we used over the last experiment. Also the metrics collection approach was the same.

4.10.1 Participants

In total, 20 persons engaged in the experiment. They were all computer science graduate (doctorate or master) students with different levels of knowledge in software testing. From this total, 16 enrolled in the discipline of Experimental Software Engineering and participated in the experiment as part of the discipline. The other 4 students participated as volunteers. The 20 participants were randomly assigned in pairs to form 10 replicas of the latin square in the same way that we did over the last experiments.

4.10.2 Experiment Operation

This experiment had the same configuration of the first and second round of studies. We had three days with sessions of two hours each. The experiment took place in a laboratory at Informatics Center. After the experiment execution we applied a questionnaire to the subjects asking about their level of experience in software testing, about activity execution, if they had any difficulties and what suggestions they could provide to help future replications of this study. Then, since the experiment was part of an Experimental Software Engineering discipline, we brought the experiment planning and its results to a discussion in class in order to get some feedback.

In general the experiment was well accepted. Some of them questioned why they couldn't use the TestWatcher observation field to report the CRs instead of reporting CRs in a file apart. The main reason that we proceeded this way was that we wanted to separate observations from reported CRs. When the test case didn't apply to the product under test we told the students to write an observation reporting that. If CRs and observations were reported in the same place it would be confusing to analyse it afterwards.

4.10.3 Data Analysis

Because one student missed day 1 activity we excluded her results since the lack of the dry run activity. We observed that she had a little difficulty on the latin square first round execution. This could have impaired her results and we would have the same problems of the last experiment. So, in total, we analyzed the results of 18 students, completing 9 latin square replicas. Next sections present the achieved results.

4.10.3.1 Time Analysis

Following the same analysis procedure of the previous experiments, first we plotted a box plot graphic as illustrated in Figure 4.11. This graphic shows that the execution time tends to smaller values on the ST compared with the GT. We can also notice that the GT median almost matches the ST upper quartile (Q3). Also no outliers appeared in the graphic. These results follow the same tendency observed in the second experiment box plot graphic. However, in the second experiment graphic, we notice a larger dispersion. These differences were probably caused by the distinct time collection approaches used in the experiments. Also, Table 4.8 describes the average and standard deviation for the execution time observed in both techniques.

Figure 4.12 depicts subjects individual responses in both techniques and shows that almost

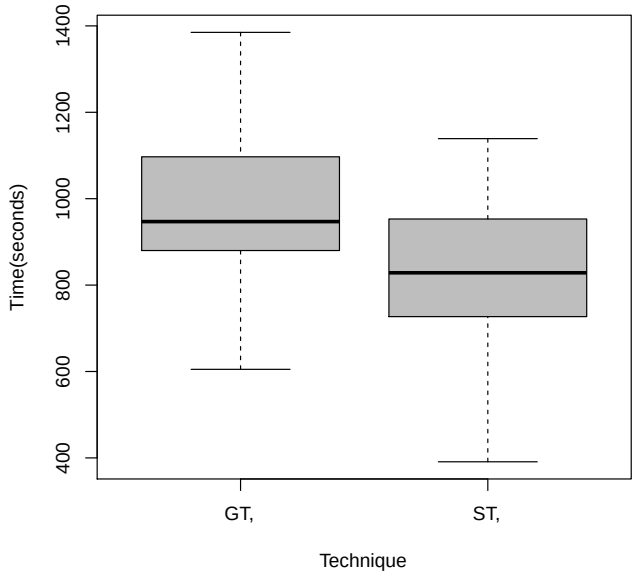

Figure 4.11 Fourth experiment blox plot graphic.

Table 4.8 4th experiment average and standard deviation.

	Generic	**Specific**
Average	975	824
Standard Deviation	192	172

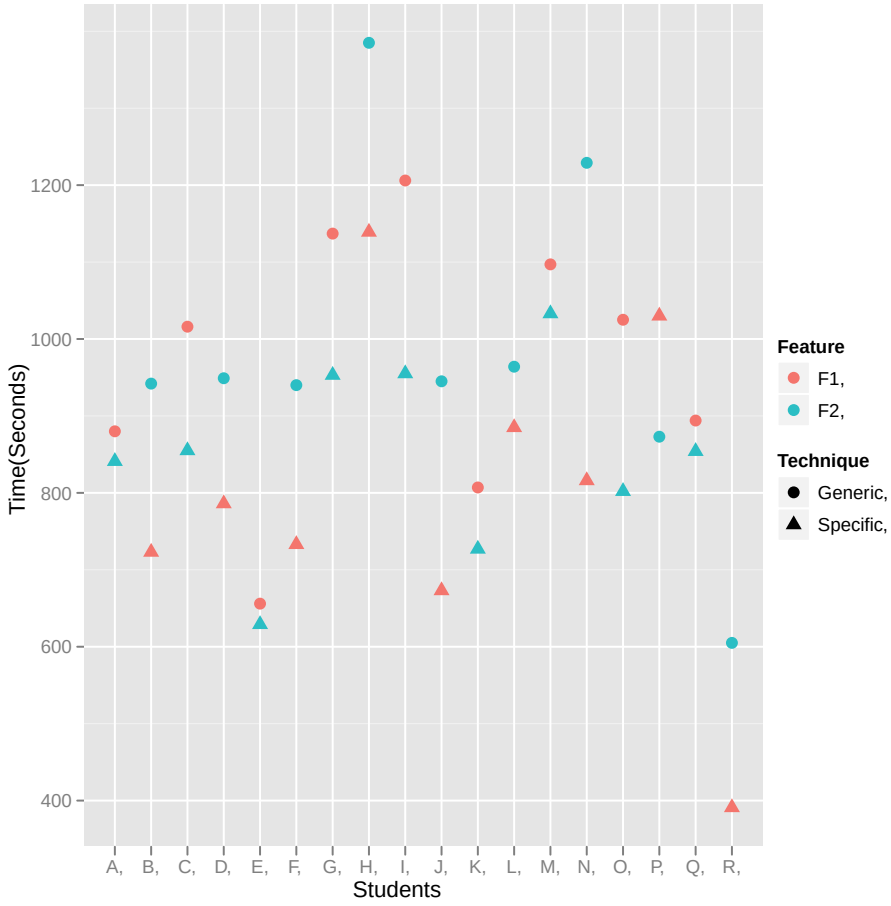

Figure 4.12 Individual results for both approaches.

the totality (94%) of the students finished the specific test suites in less time than the generic ones. Student P was the only one who took more time to execute the ST than the GT. We weren't able to find out if something went wrong to that particular subject.

Moving on with the statistical analysis, Figure 4.13 shows the curve of the Box-Cox transformation technique, but again there was no need to apply any transformation to the data set. Next, Tukey Test of Additivity returned a p-value of 0.57, which doesn't give us enough evidence to reject the null hypothesis that stated that our effect model is additive. Finally, Table 4.9 describes the ANOVA test results leading to a p-value for the factor technique of 0.0001082 which gives us a good evidence that the specific test suites can reduce time for execution even without considering the investigation time.

From what we have observed, we believe that one of the reasons that made subjects took longer executing generic suites was that, when they got to the point of an inconsistency between what the test case described and the system behavior, they tried to explore around the tool looking for missing steps and pressing the return button in order to repeat some steps. Then, when they couldn't find out what was wrong, they paused the execution time for investigation. Perhaps generic test suites may trigger a more exploratory behavior on the subjects. They

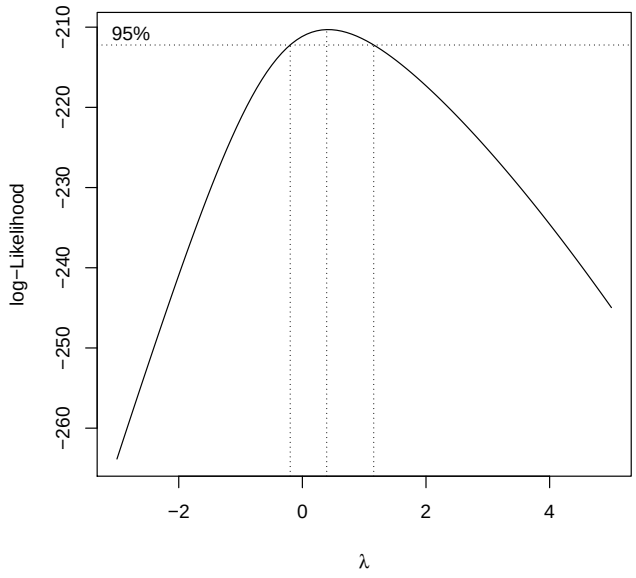

Figure 4.13 4th experiment Box-Cox graphic.

Table 4.9 Fourth experiment ANOVA results.

	Df	Sum Sq	Mean Sq	F value	*p-value*
Replica	8	661378	82672	10.4027	**4.748e-05**
Replica:Student	9	333278	37031	4.6596	**0.0037052**
Feature	1	9571	9571	1.2044	0.2886915
Technique	1	206267	206267	25.9548	**0.0001082**
Residuals	16	127155	7947		

would explore the software more than someone who strictly followed the specific test case.

One of the consequences of this behavior would be finding more defects. Indeed, one of the students, while working with the generic suites, reported a defect that no one else did because he tried to repeat some of the steps to find a button described by the test case that didn't apply to the product under test. When we observed participants acting like that, we thought that this would cause an increase on the number of valid CRs found using the GT because the students who made this work around would go beyond the test case scenario in comparison with the student executing the specific version of the same test case. Nevertheless there was not a significant influence observed in the CR analysis.

In addition, in the Software Testing discipline there is an approach called exploratory testing, where the testers have the freedom to explore various flows at the same time testing more than one feature and performing the steps back and forth. However, when executing script based test cases, the focus should be on the script.

Since we were able to gather evidence showing that the technique has influence on execution time without considering the investigation time, we didn't run the second analysis adding extra time for pauses in test execution since differences would only increase but the scenario wouldn't change. In total there were 19 interruptions caused by generic test suites inaccuracies during the whole experiment execution. Considering 60s as the shortest amount of time for investigation we would have an increase os 19 minutes for the generic suites execution.

4.10.3.2 Terminated CR Analysis

Back on the previous experiments, we classified our CRs into the following categories: valid, invalid and duplicated. But this time we decided to add a new category: irreproducible. We did that because two students reported CRs while using their personal MacBooks during the activities and RGMS product line doesn't have yet a version that supports OSX. Consequently, for them, the interface had some small problems such as labels that were too short for the contained text or fields that were too bright for reading. Because we couldn't reproduce this defects using windows (the system used in the laboratory computer), we considered these CRs as irreproducible considering the environment that we had planned for the experiment execution. It is also noteworthy to remember that the CR report activity didn't affect time collection since, in this experiment, the students paused test time execution for filling the CR template.

Table 4.10 describes the results of the CR analysis. As we can see, there isn't a significant difference concerning valid CRs. Almost every CR that participants reported on the ST was also reported on the GT. This data seems to contrast with our observations that subjects working with GT could find more deffects because of the work around they did while trying to find the missing steps described on the test case.

On the other hand, there was a high number of invalid CRs on the GT compared with the ST. This happened because some subjects didn't investigate the defects even if the experiment conductor encouraged them to ask questions about any misunderstanding that they might have had. However, we observed that the subjects more experienced with test execution tended to ask more and report less invalid CRs.

The only invalid CR reported on the ST was coincidentally reported by the same student that took more time on the ST than on GT. This reminds us that even if the test case is specific

for each product from an SPL it is not guaranteed that there won't be terminated CRs. A poorly specified test case also leads to misunderstandings. That's why specifying detailed and accurate scenarios are important. This CR was reported because the test case asked to check if the publications report was generated and saved correctly, but RGMS saves the report always with the same name and previously she already had generated some reports so she thought that the report wasn't generated because there wasn't a different file name. To avoid this, the test case should specify that all reports are saved with the same file name.

Table 4.10 Reported CRs.

	GT	ST
Valid	15	18
Invalid	20	1

4.10.4 Threats to Internal Validity

Because we controlled most of the unwanted effects learned from other experiments, we were able to achieve more solid results in this experiment. However some considerations must me made about the experiment execution.

4.10.4.1 Configuration of Latin Square Replicas

Like discussed before, to form the latin square replicas, we randomly assigned the subjects in pairs to form the rows of each square, then we randomly assigned *Publications* to Feature 1 and *ResearchLines* to Feature 2 to form the columns of the squares. Then, we raffled the techniques arranging them to form the first replica the same way that Table 4.1 describes. Lastly, we replicated the techniques arrangement to form the other replicas.

A different approach, randomly assigning the treatments for each replica, could perhaps give more solid results because it would be a full randomized configuration. Nevertheless, we believe that this consideration didn't compromise our results because we had really significant differences in individual results and also because we ran tests that excluded non-additivity and non-normality anomalies. However, in a possible replication of this study it's recommended to perform this full randomization.

4.10.4.2 Heterogeneous Environment

During the experiment execution, some students performed the activity using their personal laptops. This resulted in a heterogeneous environment. However, RGMS doesn't have yet versions that support different operational systems. So, when two students used Mac notebooks to execute the activity, they reported 3 defects, related to RGMS interface that had labels that were too short for the text in it, that students who worked with Windows operational system could not find. Because we could not reproduce those defects using Windows, we considered these 3 defects as irreproducible and didn't consider them in the CR analysis. Nevertheless, we ran a second time analysis removing these two students results and there wasn't a significant

change in our results, so this situation didn't affect significantly time metric collection.

To fix this situation for future replications of this study, we could think of two possible solutions. The first one, and more simple, is to use a laboratory where the computers have a similar configuration and not letting subjects use their personal notebooks. The second solution is to create an image using a virtual machine that students could download and work in any computer.

4.11 Fifth Experiment

After the fourth experiment execution there were some discussions about how generic and specific suites could impact the process of test execution as a whole, that is, considering the time for executing test cases and reporting CRs. After all, as discussed over Chapter 3, when the tester reports an invalid defect there's an associated waste of time which leads to a productivity decrease. That's why a high rate of terminated CRs is a bad indicator for a test execution organization.

Over the last experiments we didn't count time for the CR report since the objective was to evaluate the impact of the techniques on the test execution time only. While executing the activities, testers had to pause time before reporting a CR. The results from experiments 2 and 4 gathered some evidence that, considering only test execution time the adoption of the ST can bring benefits to a SPL test execution environment.

Considering this discussion, we planned a new experiment round to register the test execution process as a whole. In this study, testers won't have to pause execution time for reporting CRs. Nevertheless, the other aspects of the experiment planning such as design and instrumentation remained the same. This experiment was performed on February of 2012 at the University of Brasília. Next sections describe its results.

4.11.1 Participants

For this study, we gathered 22 computer science ungraduate students from the University of Brasília who subscribed in the summer course of object oriented programming (OOP). To form the latin square replicas we followed the same procedure of the last experiments to assign the rows and columns, but we also raffled the treatments for each replica instead of randomly assigning the treatments for the first replica and replicating this configuration to the others. We did that to avoid the threat discussed on Section 4.10.4.1.

4.11.2 Experiment Operation

For this experiment we used the same material of the previous experiment. Same RGMS products, same features and test suites and the students used the TestWatcher to collect time. This experiment took three classes of the OOP course. Each class lasted 2 hours. The first class consisted of the training and the dry run following the same procedure of the last experiment except that we instructed the students not to pause time while reporting CRs. However they still had to pause time for asking questions, so that we could treat the investigation time like we

did on experiment 4. The remainder of the experiment ocurred normally with day 2 and day 3 to execute the latin square rounds. By the end of the experiment we excluded the results from two students that didn't attend to all classes and, consequently, didn't complete the activities.

4.11.3 Data Analysis

While assessing the initial results, we noticed that 5 students didn't follow the activity like they were supposed to. Some of them executed the test cases freely exploring the tool, testing different flows and inputs that the test script didn't provide. Consequently they spent more time exploring flows and reporting defects that the students who strictly followed the test script. Besides that, two students apparently didn't collect time properly since their spreadsheet presented strange times. For instance, one of them managed to execute one test case in 1 second, which is highly unlikely. The other one executed the same test case more than once. To mitigate these problems we decided to eliminate the results of these students.

Our initial configuration included 22 students forming 11 latin square replicas. However, by the end of the experiment, we eliminated the results from 9 students. Two of them didn't complete the activities because they missed one ore more classes, 5 were eliminated because they performed exploratory tests and two were eliminated because they didn't collect the metrics properly. At the end, only 13 students presented results that could be analyzed. We were able to preserve two replicas from our initial configuration. The 9 remaining students were randomly rearranged to form three more replicas. In total we executed the analysis considering 5 replicas.

4.11.4 Time Analysis

Analyzing the data set provided by the 5 replicas, we see from the box plot graphic on Figure 4.14 that it follows the same tendency presented by the box plot graphics from the other experiments. One more time the ST presents smaller values then the GT. Also we can see that the GT observations seem to be more dispersed than the ST values. Also the GT median value is located between the ST largest observation and the upper quartile (Q3). Also, Table 4.11 describes the average and standard deviation for the execution time observed in both techniques.

Table 4.11 5th experiment average and standard deviation.

	Generic	**Specific**
Average	1251	1061
Standard Deviation	268	203

Moreover, some interesting results can be noticed on the graphic illustrated in Figure 4.15 showing the individual results associated with the techniques. Students G and I executed the generic suites faster than the specific ones. Besides that, student E executed both suites in practically the same time. This graphic shows more balanced results than the fourth experiment individual results graphic displayed on Figure 4.12 where, from the 18 students who executed the experiment only one executed the generic suites faster in less time than the specific. This was probably caused by the different time collection approaches. Besides that, it's good to

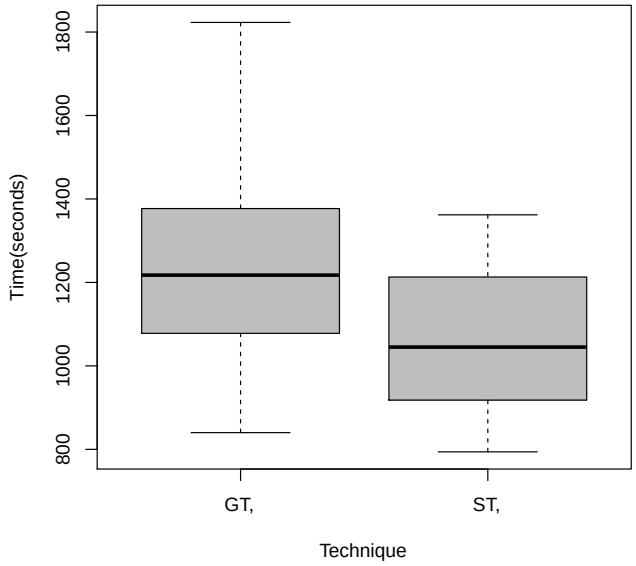

Figure 4.14 5th study box plot graphic.

remember that the students executed each technique using different features (F1 and F2). That's why we can't conclude the technique influence only by looking only at the individual results graphic.

Moving on with the statistical analysis, again we used the Box-Cox transformation technique which resulted on the curve described in Figure 4.16. One more time the curve shows the function maximum value located between 0 and 1, so no transformation was necessary. In order to guarantee that we have an additive model we ran the Tukey Test of Additivity function that returned a p-value of approximately 0.79 so we continued to consider our model to be additive. Finally the ANOVA tests returned the set of p-values described in Table 4.12. The p-value associated with the technique factor is 0.01 which indicates that the technique really influenced this data set. The results achieved by this experiment bring evidence that the adoption of the ST can decrease time for an SPL test execution process considering it as a whole (test execution + CR report).

As a final remark, it is notable the difference of performance in metrics collection between the participants from this experiment and the participants from the second and the fourth experiment. In the second experiment from a total of 9 participants, 8 collected the metrics properly and on the fourth experiment only one student from a total of 20 was eliminated because she missed day 1 activities. In this experiment we lost 7 students because they didn't follow the procedure presented or didn't collect the metrics properly. We believe that this happened because on experiments 2 and 4 the majority of the participants were informatics graduate students with a solid basis on software engineering and more experienced with software testing. In particular, the majority of the participants engaged on experiment 4 were following a course of empirical software engineering where they had classes and projects about planning and executing

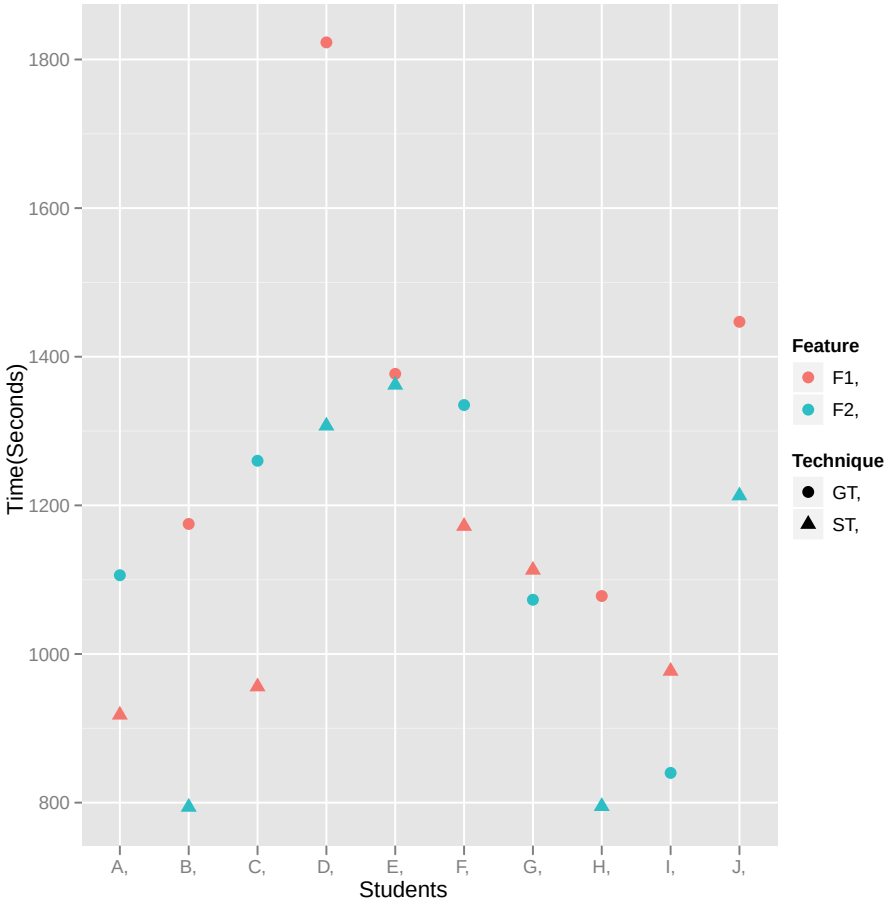

Figure 4.15 Graphic comparing individual results.

Table 4.12 Fifth experiment ANOVA results.

	Df	Sum Sq	Mean Sq	F value	*p-value*
Replica	4	411791	102948	6.1673	**0.01446**
Replica:Student	5	425216	85043	5.0947	**0.02143**
Feature	1	45220	45220	2.7090	0.13840
Technique	1	181832	181832	10.8931	**0.01085**
Residuals	8	133540	16693		

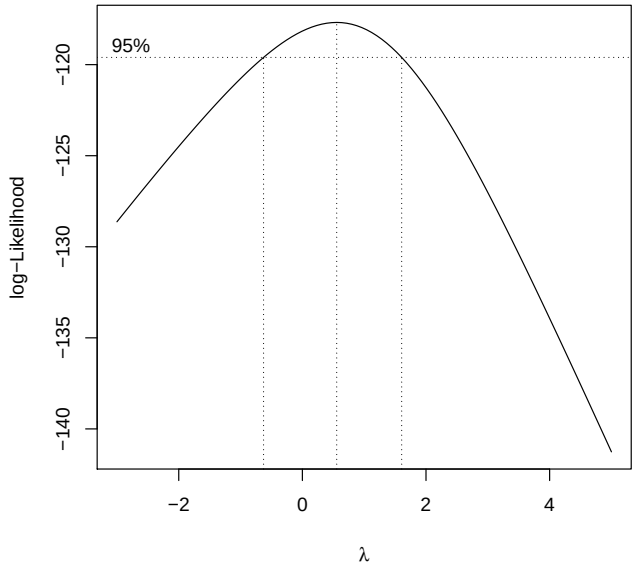

Figure 4.16 Box-Cox curve.

experiments. Perhaps, because of that, they understood better what needed to be done.

On the other hand, the participants of this study were students at the beginning of their graduate school without a good knownledge or experience in software engineering. Perhaps with more training, maybe two or three sessions, we could have increased the number of replicas. Another action taken to prevent this exploratory behavior would be to fix the input values in the test cases. This way students would not have the liberty to explore different input values. Nevertheless, considering the participants who executed the test cases without exploring the tool and properly collecting the metrics, we achieved results consistent with experiments 2 and 4.

4.11.5 Terminated CR Analysis

After assessing each CR and classifying them in valid, invalid or duplicated we have the number described in Table 4.13. Similar to the last experiment we had again a high number of terminated CRs reported with the GT. However, in that study we considered 18 participants and now we analized only 10 participants results. This higher number is consistent with the last section discussion about how the participants that had more experience with software testing reported less invalid CR because they asked more. In this experiment all participants reported at least one invalid CRs while executing the generic suites even if there were encouraged to ask questions about any misunderstanding.

This time we noticed that there were more CRs reported with the GT than with the ST. We believe that this happened because some students were more judicious than others. For example, one of them tried to insert a research line with a very long name and then the system

Table 4.13 Reported CRs.

	GT	ET
Valid	13	9
Invalid	20	1

didn't register the entity correctly. To mitigate this risk we could fix the inputs on the test case like discussed previously. Another student reported a CR because the string described in the test case didn't contain capital letters and the ones presented by the product did. So the test suites should probably be revised more carefully.

4.11.6 Threats to Internal Validity

In brief, this experiment gathers some evidence about the techniques impact on the process as whole, however there are some considerations to be made about the validity of our results. For instance, considering the threat discussed over the last experiment about executing the experiment in a heterogeneous environment, that is, using different computers and operational systems, we had planned to execute the experiment in a laboratory in the University of Brasília where all computers presented a similar environment. However, since it was a summer course and the majority of the students were in vacation the University warned in the last hour that the laboratory would be closed for repairs and we couldn't use it. Since there was no time to prepare a virtual machine, the students had to install and execute the activities in their personal computers.

One of the students, who used OSX reported the same interface problems reported on experiment 4. We would classify this CRs as irreproducible. However he was one of the two students who didn't collect time properly, completing test cases in a few seconds and the students who were eliminated from time analysis were also were eliminated from the CR analysis, so this risk was mitigated.

Another interesting consideration to be made is that since we are collecting the time taken to report the CRs, we need to be careful to control the size and complexity of the CRs reports because more careful participants may take more time to report a CR with more details while others would describe the defect using only one line or two. To report CRs we provided the same template used over the last experiments which had 4 information to fill in: CR ID, Test Case ID, number of the step where the defect was found and a brief issue description. After assessing the CRs we observed that they had in average the same size, the students didn't use more than one paragraph to report CRs.

4.11.7 Selected Experimental Case

To execute the first experiment we chose TaRGeT (which is a real SPL) requirement artifacts and products to compare the techniques. However, at the end of the first experiment, we learned that this choice was not appropriate for our purposes. First the product and the requirements were written in english which made it difficult for some participants to execute the tasks. Second, we needed more than one training session so that the features and software overall func-

tionality could be understood. Perhaps two more training sessions.

To avoid those problems we chose a non-real SPL, the RGMS, to work with on the later experiments. The use of a non-real SPL may be seen as a potential threat to our results. However, this system has been evolved for some years in the discipline of software reuse in the Informatics Center of the Federal University of Pernambuco and it represents situations that are commonly seen in real SPL, for instance the different report format outcomes are also seen on TaRGeT product line.

Besides that, some researchers believe that empiric evaluations are not limited to real projects. In fact, Buse claimed in his paper about benefits and barriers in user evalutions in software engineering [BSW11] that non-real artifacts *"can often allow researchers to more easily translate research questions into successful experiments. For example, rather than a study in which participants must become familiar with a real (and often complex) system to perform some task of interest, one might instead elect to conduct an experiment where the task artifact is artificially simplified. While designing an artificial project may take time upfront, it can have several important advantages. For example, it may reduce training time, simplify recruiting, and, perhaps more importantly, it can allow the researcher greater control over confounding factors"*. We believe that choosing RGMS we gained all these benefits.

4.12 Threats to External Validity

With respect to the external validity, there were some conditions that might limit the generalization of the results achieved by these experiments. First, the subjects involved in this experiment weren't all testers. They were computer science ungraduate and graduate students with different skills on software testing. In the first, third and fifth experiments we used ungraduate students. On the second experiment we used three ungraduate and 7 graduate students. Finally, on the fourth experiment we used 20 graduate students, of which, some of them really worked as testers in different companies but other didn't have the same experience.

Some studies have already addressed the question of the feasibility of conclusions draw from results of experiments made with students and some suggests that, for some software engineering areas, using students as subjects in experiments is often perceived as a good surrogate for using industry professionals [BSW11, Sta07]. Furthermore, Runeson compared the results achieved in one experiment using three different groups: ungraduate students, graduate students and industry people. His results indicated that there was no significant difference between the three groups results, however the ungraduate students took longer to execute the tasks requested, so the data dispersion observed in this group was larger than the other two groups [Run03]. This result is consistent to what we have observed in our experiments. We had similar results for the three experiments however when we used ungraduate students we had more data loss.

Besides that, from that we have observed in the executed experiments, we believe that if we use testers to replicate these studies perhaps we would notice a decrease on the number of terminated CRs for the GT. This happens because some subjects with less experience in software testing, tended to report more terminated CRs than the more experienced students, even if they were all encouraged to investigate the defects with the experiment conductor.

Another matter about the experiment subjects is that testers with more experience testing the same SPL will tend to have fewer problems while executing generic test suites since they are already familiar with each product specificities. However, when new features are incorporated by the SPL, new configurations are possible and the tester again has to take some time to get used to the new features.

The results achieved by this experiment also depend on the SPL selected. Perhaps SPL with more feature entanglement and more variation points might benefit more in adopting the ST. On the other hand, SPL with features that tend to be more independent from each other perhaps won't benefit so much from using the ST. In addition, the frequency of innacuracies present in the the test suites also influence the difference between the techniques. The more inaccuracies, the more time is spent to investigate and execute test cases. In our studies we injected 2 or 3 inaccuracies in each generic test suite. However we don't have evidence about the frequency with which these mistakes happens because we couldn't have access to the test execution reports of the test execution environment that we investigated. In Section 5.2 we propose a study to analyse this frequency.

Finally, one last question that we have discussed during this work is whether the test cases generated by the existing test derivation techniques would be equivalent to the ones that we wrote. In Section 5.2 we propose an evaluation of these techniques that woul help to answers this question.

CHAPTER 5

Conclusions

In this work, we discuss how challenging functional SPL testing can be mainly due to the huge amount of configurations that can be derived from a SPL and also because the requirements changes from one product to another. In this context, it is hard to manage variability in test cases based on use case scenario specifications. However, literature has recently been proposing some solutions to specify test cases representing variability in the domain engineering level and generating specific test cases in the application engineering level. However, this research area still lacks of more careful and sound evaluations about the proposed solutions.

In the industry context, we observed that one real test execution environment uses test cases that describe the overall family behavior including optional and alternative steps in one single specification that is used to test products in the application engineering level. In Chapter 3, we discuss how this test cases can present inaccuracies in respect to the products real behavior and how this situation may hamper the test execution process mainly because testers don't know the products specificities prior to the test execution. As a result, when the tester gets to the point where test cases steps don't apply to the product under test, he may interpret this as a product defect. The main consequences for the test execution process is scaped defects, the increase of time to execute test cases and a higher rate of terminated CRs leading to an overall decrease of software test productivity.

Also on Chapter 3, we propose specific test cases for SPL as an solution for the above mentioned problems since these test cases contain the steps that apply to each products configuration. However we can't assume that specific test cases will solve the problems of the test execution process without a careful comparison between these two techniques (Generic vs. Specific). For this matter we have planned and executed 5 experiments where we simulated a test execution environment with students executing test cases using both techniques. While executing the test suites, the students had to collect the test time execution and products defects. These metrics served as input to our data analysis process. After that we were able to draw some interesting conclusions. We report these experiments in Chapter 4.

Each one of the 5 experiments were important to gain experience and learn about the differences in each technique, however, in the first and in the third experiment executions, we had some problems with the planning and the execution. Because of that we didn't consider these results for analysis. In the first experiment we had problems to collect the execution time and our case study was not so interesting for our needs. In the third experiment we had problems with the operation phase because we didn't have time to execute the dry-run activity.

Nevertheless, the second, forth and fifth experiment rounds were useful to collect and assess metrics. However we can't perform a meta-analysis in these studies because each one of them considered metric being collected in different manners. In the second experiment we gathered

evidence that considering execution and investigation time as a whole the specific technique can reduce time for test execution. In the fourth experiment we collected execution time separately from investigation time. Again the statistical tests indicated that specific test suites decrease time for test execution. Also, in this experiment there was a considerable high amount of invalid CRs reported by the students while executing generic test suites. Finally, in the fifth experiment round we collected time considering the test execution process as a whole, collecting time for test execution and CR report together. The data analysis showed one more time that specific test suites improve test execution productivity by reducing time and the number or invalid CRs reported.

Finally, based on the results achieved in those experiments, we concluded that the test execution process for SPL products can benefit from using specific test cases because there is an increase of productivity by reducing test execution time and terminated CR rates, showing that it is worth indeed to use a derivation technique for generating product specific test cases. The remainder of this chapter is organized as follows. Section 5.1 presents some related work and Section 5.2 suggests future works based on the results achieved in this work.

5.1 Related Work

SPL Testing has been considered a challenging task [PM06, TTK04, KD06, ER11], not only due to the huge number of products that might be generated from reusable assets [PM06, JKB08], but also motivated by the lack of well known recommendations and best practices for testing product lines— actually, most of the research on SPL testing focuses on proposing new approaches and techniques [ER11, dMSNdCMM$^+$11], instead of empirically assessing their benefits.

For instance, Bertolino and Guinesi proposed a text based use case extension tailored for product line functional testing [BG03], whereas other works detail how to derive product specific test cases from activity and sequence diagrams [NPLTJ03, KPRR03, OG05, RKPR05]. Our investigation complements these works, since it shows evidences about the benefits of product specific test cases, which might be generated from any derivation approach.

Regarding empirical studies on SPL testing, in 2010 Ivan et al. [dMSNdCMM$^+$11] conducted a systematic mapping study with the purpose of investigating the state-of-the-art of SPL testing practices and identifying possible gaps in existing techniques. This study illustrated a number of areas in which additional investigation would be useful, specially regarding evaluation and validation research. This work also served as a basis to propose a novel process for supporting testing activities in SPL projects, the RiPLE-TE [dCMdMSNdAdLM11]. In addition, they conducted two experimental studies to evaluate the proposed process.

Ganesan et al., compared the costs and benefits of two approaches for SPL quality assurance: one that does not consider reusable assets (and test each SPL member as an independent product), and another that considers reusable assets among different components [GKK$^+$07]. Their conclusions, based on *Monte-Carlo* simulations, points that it is worth to test the reusable assets of an SPL during domain engineering (using code inspection and static analysis), and test just product specific parts during product engineering (using not only code inspection and static analysis, but also functional tests). Our study has only focused on functional testing during the

application engineering level.

Denger and Kolb compared the effectiveness of code inspection and functional testing to find SPL defects [DK06]. Their findings suggest that the two techniques complement each other, finding different types of defects. Differently, our assessment concerns about the level of details in which test designers specify SPL test cases, and its consequences on both productivity and quality of defect reports.

Empirical studies on software testing is not so common as well, as discussed by Juristo et al., *"over half of the existing knowledge is based on impressions and perceptions and, therefore, devoid of any formal foundation"* [JMV04], and the lack of such an empirical body of evidence is a considerable challenge of the software testing research [Ber07, ESR08].

Nevertheless, empirical comparisons between testing methodologies have been recently reported. For instance, Itkonen et al. compare the effectiveness to find defects with tests based on exploratory tests [IML07]; while Lima et al., compare two test prioritization techniques ($Manual \times Automatic$) also using a latin square design [LISA09].

5.2 Future Work

In this work we discussed and gathered evidence about the benefits of adopting SPL techniques for specifying functional test cases. Therefore, it is now our intent to investigate the existing techniques. To do that, first we can perform a systematic mapping study [KP08] with the objective to know what techniques are there and what are their main characteristics. Then, based on the results of this study, we can select the most representative techniques and use them to perform new empiric studies to compare them.

Concerning the techniques that we listed as related work in this thesis, they all presented annotation-based mechanisms to represent variability [KAK08]. This means that they model variability introducing implicit or explicit annotations on the use cases description. A different way to represent variability is to use composition-based techniques. These techniques implements features and other variations as distinct modules and so, to generate a product line member requirements artifact, there must be a composition of these modules. Bonifacio and Borba defined a composition-based technique to model use cases for SPL called Modeling Scenario Variability as Crosscutting Mechanisms (MSVCM) [BB09]. MSVCM proposes an use case model in the domain engineering level with explicit and separated mechanisms to define variability and express configuration knowledge. Besides that, MSVCM uses a derivation mechanisms that automatically generates the products specifications in the application engineering level.

During the development of this work, there were some discussions about how we could use MSVCM to generate functional test cases. Back on Chapter 4, we presented TaRGeT, a tool that takes use cases as input and generates test suites. One of TaRGeT features is the Use Case Editor that allows the creation and edition of uses cases. We thought about integrating these two approaches. That is, improving TaRGeT use case editor to model MSVCM constructs. Besides that, the user should be able to derive product specifications using TaRGeT and then generating specific test suites in the application engineering level.

Lastly we have discussed how innacuracies appears on generic test suites but we didn't dis-

cuss the frequency with which this happens. Like discussed in Section 4.12 we didn't have access to the test execution reports in the environment where we observed this problem. However it would be possible to analyze this frequency searching through different SPL test specifications available on the internet. This would help to raise more evidence about this problem.

APPENDIX A

Test Suites

Chapter 4 describes the experiments conducted in order to evaluate two different functional test cases design techniques for SPL. In that chapter we also discuss how we wrote the test suites. First we wrote the generic suites for the features under test and then, based on these test suites, we manually adjusted them to generate the specific versions for P1 and P2 respectively. Figure 4.2 describes the test suites.

In this appendix we present three test suites to help the reader understand better the differences between generic and specific test suites. From Table A.1 to Table A.6 we describe the generic test suite associated with the Publications feature of the RGMS product line. From Table A.7 to Table A.12 we show the correpondent suite adjusted for P1 configuration. And, from Table A.13 to Table A.18 describes the correpondent suite adjusted for P2 configuration. Each test suite contains 6 test cases. The remaining suites are available online [Acc12].

Table A.1 SG_F1_1

Id do Teste:	SG_F1_1	
Objetivo:	Inserir um novo membro	
Setup:	Nenhum	
Condições Iniciais:	Existe uma imagem jpeg disponível salva no computador	
Procedimento do Teste:	APERTE PLAY!!	
1)	Inicialize o sistema	O sistema RGMS é iniciado
2)	Pressionar o botão “Cadastro" na página inicial do sistema.	Uma página com o formulário para cadastro de um novo membro é exibida.
3)	Preencher os campos: “Nome”, “Login”, “Senha”, “Email”	Campos do formulário corretamente preenchidos.
4)	Pressionar o botão “Carregar Foto”	Sistema de arquivos abre para selecionar a foto
5)	Selecione a foto (arquivo .jpg)	A foto é corretamente selecionada e exibida no campo de fotos
6)	Pressionar o botão “Cadastrar”	A mensagem “Usuário inserido com sucesso!” é exibida

Table A.2 SG_F1_2

Id do Teste:	SG_F1_2	
Objetivo:	Inserir um novo Artigo de Conferência	
Setup:	Nenhum	
Condições Iniciais:	1- O usuário está logado no sistema; 2- Deve existir algum arquivo .pdf salvo no computador	
Procedimento do Teste:	APERTE PLAY!!	
1)	Selecionar a opção “Publicações” na página inicial do grupo.	Uma página é exibida com todas as publicações do grupo.
2)	No dropdown menu escolher a opção “Artigo em Conferência” e pressionar o botão “Cadastrar”	Uma página com o formulário para cadastro de uma nova publicação é exibida.
3)	Preencher os campos “Título Artigo”, “Nome da Conferência” e “Ano”	Campos do formulário corretamente preenchidos.
4)	Pressionar o botão “Carregar Arquivo pdf”	Sistema de arquivos abre para selecionar arquivo
5)	Selecione o arquivo pdf	O caminho do arquivo aparece corretamente no textfield
6)	Selecione um nome na lista de Membros e clique no botão “»”	O nome do membro aparece na lista da direita
7)	Pressionar o botão “Cadastrar”	A mensagem “Artigo Cadastrado!” é exibida

Table A.3 SG_F1_3

Id do Teste:	SG_F1_3	
Objetivo:	Objective:Inserir uma tese de doutorado	
Setup:	Nenhum	
Condições Iniciais:	1- O usuário está logado no sistema; 2- Deve existir algum arquivo .pdf salvo no computador	
Procedimento do Teste:	APERTE PLAY!!	
1)	Selecionar a opção “Publicações” na página inicial do grupo.	Uma página é exibida com todas as publicações do grupo.
2)	No dropdown menu escolher a opção “Tese de Doutorado” e pressionar o botão “Cadastrar”	Uma página com o formulário para cadastro de uma nova publicação é exibida.
3)	Preencher os campos obrigatórios (campos que possuem “*”)	Campos do formulário corretamente preenchidos.
4)	No dropdown menu “Doutorando” selecione um membro	O Membro do grupo de pesquisa é selecionado
5)	Pressionar o botão “Carregar Arquivo pdf”	Sistema de arquivos abre para selecionar arquivo
6)	Selecione o arquivo pdf	O caminho do arquivo aparece corretamente no textfield
7)	Pressionar o botão “Cadastrar”	O diálogo “Artigo Cadastrado!” é exibido
8)	Pressione “Ok” no diálogo	A tela de publicações é exibida.
9)	Pressione o botão “Busca de Publicações por Membros”	Tela de busca é exibida
10)	Selecione na lista de Membros o membro autor da tese de doutorado recém cadastrado e clique no botão “»”	O membro aparece na lista da direita
11)	Pressione “Buscar”	A tese de doutorado recém cadastrada aparece nos resultados da busca

Table A.4 SG_F1_4

Id do Teste:	SG1_F1_4	
Objetivo:	Gerar Relatório de Publicação	
Setup:	Nenhum	
Condições Iniciais:	1- O usuário está logado no sistema; 2-Deve existir pelo menos uma publicação cadastrada no sistema	
Procedimento do Teste:	APERTE PLAY!!	
1)	Selecionar a opção “Publicações” na página inicial do grupo.	Uma página é exibida com todas as publicações do grupo.
2)	Na parte de Geração de Listas de Publicações, selecione o formato Bibtex e pressione o botão “Gerar Lista”	Um diálogo aparece com a mensagem “Relatório Gerado com Sucesso!”
3)	Após pressionar ok, vá na pasta do sistema e verifique que o arquivo foi gerado em bibtex	O arquivo lista.bib foi gerado (não é necessário abrir o arquivo)

Table A.5 SG_F1_5

Id do Teste:	SG_F1_5	
Objetivo:	Gerar Relatório de Publicação	
Setup:	Nenhum	
Condições Iniciais:	1- O usuário está logado no sistema; 2-Deve existir pelo menos uma publicação cadastrada no sistema	
Procedimento do Teste:	APERTE PLAY!!	
1)	Selecionar a opção “Publicações” na página inicial do grupo.	Uma página é exibida com todas as publicações do grupo.
2)	Na parte de Geração de Listas de Publicações pressione o botão “Gerar Lista”	Um diálogo aparece com a mensagem “Relatório Gerado com Sucesso!”
3)	Após pressionar ok, vá na pasta do sistema e verifique o arquivo gerado no formato correto	O arquivo foi gerado (não é necessário abrir o arquivo)

Table A.6 SG_F1_6

Id do Teste:	SG_F1_6	
Objetivo:	Verificar Formatos de Geração de Relatórios	
Setup:	Nenhum	
Condições Iniciais:	O usuário está logado no sistema;	
Procedimento do Teste:	APERTE PLAY!!	
1)	Selecionar a opção “Publicações” na página inicial do grupo.	Uma página é exibida com todas as publicações do grupo.
2)	Na parte de Geração de Listas de Publicações e verifique os possíveis formatos de geração	Os formatos de geração possíveis são corretamente exibidos

Table A.7 SP1_F1_1

Id do Teste:	SP1_F1_1	
Objetivo:	Inserir um novo membro	
Setup:	Nenhum	
Condições Iniciais:	Existe uma imagem jpeg disponível salva no computador	
Procedimento do Teste:	APERTE PLAY!!	
1)	Inicialize o sistema	O sistema RGMS é iniciado
2)	Pressionar o botão “Cadastro” na página inicial do sistema.	Uma página com o formulário para cadastro de um novo membro é exibida.
3)	Preencher os campos: “Nome”, “Login”, “Senha”, “Email”	Campos do formulário corretamente preenchidos.
4)	Pressionar o botão “Carregar Foto”	Sistema de arquivos abre para selecionar a foto
5)	Selecione a foto (arquivo .jpg)	A foto é corretamente selecionada e exibida no campo de fotos
6)	Pressionar o botão “Cadastrar”	A mensagem “Usuário inserido com sucesso!” é exibida

Table A.8 SP1_F1_2

Id do Teste:	SP1_F1_2	
Objetivo:	Inserir um novo Artigo de Conferência	
Setup:	Nenhum	
Condições Iniciais:	1- O usuário está logado no sistema; 2- Deve existir algum arquivo .pdf salvo no computador	
Procedimento do Teste:	APERTE PLAY!!	
1)	Selecionar a opção “Publicações” na página inicial do grupo.	Uma página é exibida com todas as publicações do grupo.
2)	No dropdown menu escolher a opção “Artigo em Conferência” e pressionar o botão “Cadastrar”	Uma página com o formulário para cadastro de uma nova publicação é exibida.
3)	Preencher os campos “Título Artigo”, “Nome da Conferência” e “Ano”	Campos do formulário corretamente preenchidos.
4)	Pressionar o botão “Carregar Arquivo pdf”	Sistema de arquivos abre para selecionar arquivo
5)	Selecione o arquivo pdf	O caminho do arquivo aparece corretamente no textfield
6)	Selecione um nome na lista de Membros e clique no botão “»”	O nome do membro aparece na lista da direita
6)	Pressionar o botão “Cadastrar”	A mensagem “Artigo Cadastrado!” é exibida

Table A.9 SP1_F1_3

Id do Teste:	SP1_F1_3	
Objetivo:	Objective:Inserir uma tese de doutorado	
Setup:	Nenhum	
Condições Iniciais:	1- O usuário está logado no sistema; 2- Deve existir algum arquivo .pdf salvo no computador	
Procedimento do Teste:	APERTE PLAY!!	
1)	Selecionar a opção “Publicações” na página inicial do grupo.	Uma página é exibida com todas as publicações do grupo.
2)	No dropdown menu escolher a opção “Tese de Doutorado” e pressionar o botão “Cadastrar”	Uma página com o formulário para cadastro de uma nova publicação é exibida.
3)	Preencher os campos obrigatórios (campos que possuem “*”)	Campos do formulário corretamente preenchidos.
4)	No dropdown menu “Doutorando” selecione um membro	O Membro do grupo de pesquisa é selecionado
5)	Pressionar o botão “Carregar Arquivo pdf”	Sistema de arquivos abre para selecionar arquivo
6)	Selecione o arquivo pdf	O caminho do arquivo aparece corretamente no textfield
7)	Pressionar o botão “Cadastrar”	O diálogo “Artigo Cadastrado!” é exibido
8)	Pressione “Ok” no diálogo	A tela de publicações é exibida.

Table A.10 SP1_F1_4

Id do Teste:	SP1_F1_4	
Objetivo:	Gerar Relatório de Publicação	
Setup:	Nenhum	
Condições Iniciais:	1- O usuário está logado no sistema; 2-Deve existir pelo menos uma publicação cadastrada no sistema	
Procedimento do Teste:	APERTE PLAY!!	
1)	Selecionar a opção “Publicações” na página inicial do grupo.	Uma página é exibida com todas as publicações do grupo.
2)	Na parte de Geração de Listas de Publicações, selecione o formato Bibtex e pressione o botão “Gerar Lista”	Um diálogo aparece com a mensagem “Relatório Gerado com Sucesso!”
3)	Após pressionar ok, vá na pasta do sistema e verifique que o arquivo foi gerado em bibtex	O arquivo lista.bib foi gerado (não é necessário abrir o arquivo)

Table A.11 SP1_F1_5

Id do Teste:	SP1_F1_5	
Objetivo:	Gerar Relatório de Publicação	
Setup:	Nenhum	
Condições Iniciais:	1- O usuário está logado no sistema; 2-Deve existir pelo menos uma publicação cadastrada no sistema	
Procedimento do Teste:	APERTE PLAY!!	
1)	Selecionar a opção “Publicações” na página inicial do grupo.	Uma página é exibida com todas as publicações do grupo.
2)	Na parte de Geração de Listas de Publicações, selecione o formato PDF e pressione o botão “Gerar Lista”	Um diálogo aparece com a mensagem “Relatório Gerado com Sucesso!”
4)	Após pressionar ok, vá na pasta do sistema e verifique o arquivo gerado no formato correto	O arquivo lista_publicações.pdf foi gerado (não é necessário abrir o arquivo)

Table A.12 SP1_F1_6

Id do Teste:	SP1_F1_6	
Objetivo:	Verificar Formatos de Geração de Relatórios	
Setup:	Nenhum	
Condições Iniciais:	O usuário está logado no sistema;	
Procedimento do Teste:	APERTE PLAY!!	
1)	Selecionar a opção “Publicações” na página inicial do grupo.	Uma página é exibida com todas as publicações do grupo.
2)	Na parte de Geração de Listas de Publicações e verifique os possíveis formatos de geração	Os formatos de geração possíveis são corretamente exibidos (PDF e Bibtex)

Table A.13 SP2_F1_1

Id do Teste:	SP2_F1_1	
Objetivo:	Inserir um novo membro	
Setup:	Nenhum	
Condições Iniciais:	Existe uma imagem jpeg disponível salva no computador	
Procedimento do Teste:	APERTE PLAY!!	
1)	Inicialize o sistema	O sistema RGMS é iniciado
2)	Pressionar o botão “Cadastro” na página inicial do sistema.	Uma página com o formulário para cadastro de um novo membro é exibida.
3)	Preencher os campos: “Nome”, “Login”, “Senha”, “Email”	Campos do formulário corretamente preenchidos.
4)	Pressionar o botão “Carregar Foto”	Sistema de arquivos abre para selecionar a foto
5)	Selecione a foto (arquivo .jpg)	A foto é corretamente selecionada e exibida no campo de fotos
6)	Pressionar o botão “Cadastrar”	A mensagem “Usuário inserido com sucesso!” é exibida

Table A.14 SP2_F1_2

Id do Teste:	SP2_F1_2	
Objetivo:	Inserir um novo Artigo de Conferência	
Setup:	Nenhum	
Condições Iniciais:	1- O usuário está logado no sistema; 2- Deve existir algum arquivo .pdf salvo no computador	
Procedimento do Teste:	APERTE PLAY!!	
1)	Selecionar a opção “Publicações” na página inicial do grupo.	Uma página é exibida com todas as publicações do grupo.
2)	No dropdown menu escolher a opção “Artigo em Conferência” e pressionar o botão “Cadastrar”	Uma página com o formulário para cadastro de uma nova publicação é exibida.
3)	Preencher os campos “Título Artigo”, “Nome da Conferência” e “Ano”	Campos do formulário corretamente preenchidos.
4)	Pressionar o botão “Carregar Arquivo pdf”	Sistema de arquivos abre para selecionar arquivo
5)	Selecione o arquivo pdf	O caminho do arquivo aparece corretamente no textfield
6)	Selecione um nome na lista de Membros e clique no botão “»”	O nome do membro aparece na lista da direita
6)	Pressionar o botão “Cadastrar”	A mensagem “Artigo Cadastrado!” é exibida

Table A.15 SP2_F1_3

Id do Teste:	SP2_F1_3	
Objetivo:	Objective:Inserir uma tese de doutorado	
Setup:	Nenhum	
Condições Iniciais:	1- O usuário está logado no sistema; 2- Deve existir algum arquivo .pdf salvo no computador	
Procedimento do Teste:	APERTE PLAY!!	
1)	Selecionar a opção “Publicações” na página inicial do grupo.	Uma página é exibida com todas as publicações do grupo.
2)	No dropdown menu escolher a opção “Tese de Doutorado” e pressionar o botão “Cadastrar”	Uma página com o formulário para cadastro de uma nova publicação é exibida.
3)	Preencher os campos obrigatórios (campos que possuem “*”)	Campos do formulário corretamente preenchidos.
4)	No dropdown menu “Doutorando” selecione um membro	O Membro do grupo de pesquisa é selecionado
5)	Pressionar o botão “Carregar Arquivo pdf”	Sistema de arquivos abre para selecionar arquivo
6)	Selecione o arquivo pdf	O caminho do arquivo aparece corretamente no textfield
7)	Pressionar o botão “Cadastrar”	O diálogo “Artigo Cadastrado!” é exibido
8)	Pressione “Ok” no diálogo	A tela de publicações é exibida.
9)	Pressione o botão “Busca de Publicações por Membros”	Tela de busca é exibida
10)	Selecione na lista de Membros o membro autor da tese de doutorado recém cadastrado e clique no botão “»”	O membro aparece na lista da direita
11)	Pressione “Buscar”	A tese de doutorado recém cadastrada aparece nos resultados da busca

Table A.16 SP2_F1_4

Id do Teste:	SP2_F1_4	
Objetivo:	Gerar Relatório de Publicação	
Setup:	Nenhum	
Condições Iniciais:	1- O usuário está logado no sistema; 2-Deve existir pelo menos uma publicação cadastrada no sistema	
Procedimento do Teste:	APERTE PLAY!!	
1)	Selecionar a opção “Publicações” na página inicial do grupo.	Uma página é exibida com todas as publicações do grupo.
2)	Na parte de Geração de Listas de Publicações pressione o botão “Gerar Lista”	Um diálogo aparece com a mensagem “Relatório Gerado com Sucesso!”
3)	Após pressionar ok, vá na pasta do sistema e verifique que o arquivo foi gerado em pdf	O arquivo lista_publicações.pdf foi gerado (não é necessário abrir o arquivo)

Table A.17 SP2_F1_5

Id do Teste:	SP2_F1_5	
Objetivo:	Gerar Relatório de Publicação	
Setup:	Nenhum	
Condições Iniciais:	1- O usuário está logado no sistema; 2-Deve existir pelo menos uma publicação cadastrada no sistema	
Procedimento do Teste:	APERTE PLAY!!	
1)	Selecionar a opção “Publicações” na página inicial do grupo.	Uma página é exibida com todas as publicações do grupo.
2)	Na parte de Geração de Listas de Publicações pressione o botão “Gerar Lista”	Um diálogo aparece com a mensagem “Relatório Gerado com Sucesso!”
3)	Após pressionar ok, vá na pasta do sistema e verifique que o arquivo foi gerado em pdf	O arquivo lista_publicações.pdf foi gerado (não é necessário abrir o arquivo)

Table A.18 SP2_F1_6

Id do Teste:	SP2_F1_6	
Objetivo:	Verificar Formatos de Geração de Relatórios	
Setup:	Nenhum	
Condições Iniciais:	O usuário está logado no sistema;	
Procedimento do Teste:	APERTE PLAY!!	
1)	Selecionar a opção “Publicações” na página inicial do grupo.	Uma página é exibida com todas as publicações do grupo.
2)	Na parte de Geração de Listas de Publicações e verifique os possíveis formatos de geração	Não aparece opção de geração de relatório já que a opção default é PDF

APPENDIX B

Data Analysis Script

Back on Chapter 4 we discuss the statistical tests made to check whether the ST can reduce test execution time compared with the GT. This appendix describes the commands used to execute those tests using the R statitical software [R D08]. The commands described here were adapted from the script provided by Bonifacio in his phd thesis [Bon10]. In Table B.1 we describe the input file format used to the analysis of time in the second experiment. This file is arranged in 5 columns. The first column represents the replica number from 1 to 4. The second column considers the student location inside the replica, that is, 1 or 2, The third and fourth columns represent the feature under test (F1 or F2) and the technique used (GT or ST) respectively. Finally, the fifth and last column describes the the time taken to execute the test suites in second and represents the response variable in our model. The remaining input files of the other experiments are available online [Acc12].

Table B.1 Input file containing the collected data in the second experiment.

replica, student, feature, technique, Time_Seconds
1, 1, f1, ST, 1947
1, 1, f2, GT, 2481
1, 2, f1, GT, 1252
1, 2, f2, ST, 870
2, 1, f1, ST, 1474
2, 1, f2, GT, 1629
2, 2, f1, GT, 926
2, 1, f2, ST, 896
3, 1, f1, ST, 1518
3, 1, f2, GT, 1874
3, 2, f1, GT, 1531
3, 2, f2, ST, 1183
4, 1, f1, ST, 818
4, 1, f2, GT, 981
4, 2, f1, GT, 1599
4, 2, f2, ST, 1815

We start the analysis by giving the following commands to read and save the data contained in the input file *data.dat* as follows.

```
exp1.dat = read.table(file="C:/Users/data.dat", header = T)
attach(exp1.dat)
```

Then, we create the factors that build our effect model with the following command.

```
replic = factor(replic.)
student = factor(student.)
macrofeature = factor(feature.)
technique = factor(technique.)
```

After creating the factors, we plot the box plot graphic using the response variable (time) associated with the techniques with the following command

```
> plot(executionTime~technique,col="gray",xlab="Technique"
,ylab="Time(seconds)")
```

Through the following command we adjust the effect model that will serve as basis for posterior analysis. Notice that the factor *student* is associated with the factor *replica* since for each replica we used a different pair of students.

```
> anova.ql<-aov(executionTime~replic+student:
replica+feature+technique)
```

After creating the effect model we use the following command to run the Box-Cox transformation method. Remembering Chapter 4 discussions, there was no need to apply tranformations in the data set of in any experiment.

```
library(MASS)
boxcox(anova.ql,lambda = seq(-3, 5, 1/10))
```

Next, we ran the Tukey Test of Additivity defining the following function. And than applying this function using the effect model *anova.ql* as parameter.

```
TukeyNADD.QL.REP<-function(objeto1)
{
y1<-NULL
y2<-NULL
y1<- fitted(objeto1)
y2<- y1^2
objeto2<- aov(y2 ~ objeto1[13]$model[,2] +
objeto1[13]$model[,3]:objeto1[13]$model[,2]
+ objeto1[13]$model[,4]+ objeto1[13]$model[,5])
ynew <- resid(objeto1)
xnew <- resid(objeto2)
objeto3 <- lm(ynew ~ xnew)
M <- anova(objeto3)
MSN <- M[1,3]
MSErr <- M[2,2]/(objeto1[8]$df.residual-1)
```

```
F0 <- MSN/MSErr
p.val <- 1 - pf(F0, 1,objeto1[8]$df.residual-1)
p.val
}

TukeyNADD.QL.REP(anova.ql)
```

Finally, we used the following command to analyse the ANOVA test results.

```
plot(anova.ql)
```

Bibliography

[AB08] Eduardo Aranha and Paulo Borba. Manualtest: Improving collection of manual test execution data in empirical studies. In *Proceedings of the 5th Experimental Software Engineering Latin America Workshop , ESELAW 2008, Salvador, Brazil, November, 2008*, 2008.

[Acc12] Paola Accioly. Material used in the experiment, February 2012. `http://goo.gl/Brx3r`.

[BB09] Rodrigo Bonifácio and Paulo Borba. Modeling scenario variability as crosscutting mechanisms. In Kevin J. Sullivan, Ana Moreira, Christa Schwanninger, and Jeff Gray, editors, *Proceedings of the 8th International Conference on Aspect-Oriented Software Development, AOSD 2009, Charlottesville, Virginia, USA, March 2-6, 2009*, pages 125–136. ACM, 2009.

[BCR94] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. The Goal Question Metric approach. In *Encyclopedia of Software Engineering*, pages 528–532. John Wiley and Sons, 1994.

[Bei90] B. Beizer. *Software Testing Techniques*. Van Nostrand Reinhold, New York, 2nd edition, 1990.

[Bei95] Boris Beizer. *Black-box testing - techniques for functional testing of software and systems*. Wiley, 1995.

[Ber07] Antonia Bertolino. Software testing research: Achievements, challenges, dreams. In *2007 Future of Software Engineering*, FOSE '07, pages 85–103, Washington, DC, USA, 2007. IEEE Computer Society.

[BG03] Antonia Bertolino and Stefania Gnesi. Use case-based testing of product lines. In *Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering*, ESEC/FSE-11, New York, NY, USA, 2003. ACM.

[BHH05] George E. P. Box, J. Stuart Hunter, and William G. Hunter. *Statistics for experimenters : design, innovation, and discovery.* Wiley-Interscience, 2005.

[Bon10] Rodrigo Bonifácio. Modeling Software Product Line Variability in Use Case Scenarios— An Approach Based on Crosscutting Mechanisms. Master's thesis, Federal University of Pernambuco, Recife, Brazil, 2010.

[BSW11] Raymond P. L. Buse, Caitlin Sadowski, and Westley Weimer. Benefits and barriers of user evaluation in software engineering research. *ACM SIGPLAN Notices*, 46(10):643–656, October 2011.

[cLdPdS12] Reuso Estratégico com Linhas de Produtos de Software. Research group management system, February 2012. `http://goo.gl/8UOjo`.

[dCMdMSNdAdLM11] Ivan do Carmo Machado, Paulo Anselmo da Mota Silveira Neto, Eduardo Santana de Almeida, and Silvio Romero de Lemos Meira. RiPLE-TE: A process for testing software product lines. pages 711–716. Knowledge Systems Institute Graduate School, 2011.

[DK06] Christian Denger and Ronny Kolb. Testing and inspecting reusable product line components: first empirical results. In *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, ISESE '06, pages 184–193, New York, NY, USA, 2006. ACM.

[dMSNdCMM+11] Paulo Anselmo da Mota Silveira Neto, Ivan do Carmo Machado, John D. McGregor, Eduardo Santana de Almeida, and Silvio Romero de Lemos Meira. A systematic mapping study of software product lines testing. *Information & Software Technology*, 53(5):407–423, 2011.

[ER11] Emelie Engström and Per Runeson. Software product line testing - a systematic mapping study. *Information and Software Technology*, 53(1):2 – 13, 2011.

[ESR08] Emelie Engström, Mats Skoglund, and Per Runeson. Empirical evaluations of regression test selection techniques: a systematic review. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, ESEM '08, pages 22–31, New York, NY, USA, 2008. ACM.

[GKK+07] Dharmalingam Ganesan, Jens Knodel, Ronny Kolb, Uwe Haury, and Gerald Meier. Comparing costs and benefits of different test strategies for a software product line: A study from testo ag. *Software Product Line Conference, International*, 0:74–83, 2007.

[IML07] J. Itkonen, M.V. Mantyla, and C. Lassenius. Defect detection efficiency: Test case based vs. exploratory testing. In *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, pages 61 –70, 2007.

[JKB08] Michel Jaring, Rene L. Krikhaar, and Jan Bosch. Modeling variability and testability interaction in software product line engineering. In *Proceedings of the Seventh International Conference on Composition-Based Software Systems (ICCBSS 2008)*, pages 120–129, Washington, DC, USA, 2008. IEEE Computer Society.

[JMV04] Natalia Juristo, Ana M. Moreno, and Sira Vegas. Reviewing 25 years of testing technique experiments. *Empirical Softw. Engg.*, 9:7–44, March 2004.

[JP05] A. Jedlitschka and D. Pfahl. Reporting guidelines for controlled experiments in software engineering. In *Empirical Software Engineering, 2005. 2005 International Symposium on*, page 10 pp., nov. 2005.

[KAK08] Christian Kästner, Sven Apel, and Martin Kuhlemann. Granularity in software product lines. In Wilhelm Schäfer, Matthew B. Dwyer, and Volker Gruhn, editors, *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008*, pages 311–320. ACM, 2008.

[KCH+90] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. Feature-oriented domain analysis (foda) feasibility study. *Software Engineering Institute, Pittsburgh*, 1990.

[KD06] Timo Käkölä and Juan C. Dueñas, editors. *Software Product Lines - Research Issues in Engineering and Management*. Springer, 2006.

[KP08] S. Mujtaba M. Mattsson K. Petersen, R. Feldt. Systematic mapping studies in software engineering. In *EASE 08: Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, University of Bari, Italy, 2008*, 2008.

[KPRR03] Erik Kamsties, Klaus Pohl, Sacha Reis, and Andreas Reuys. Testing variabilities in use case models. In *5th International Workshop on Product Family Engineering (PF3 2003)*, pages 6–18, 2003.

[LISA09] L. Lima, J. Iyoda, A. Sampaio, and E. Aranha. Test case prioritization based on data reuse an experimental study. In *Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on*, pages 279 –290, 2009.

[Mea12] Meantime. Meantime mobile creations, 2012. http://www.meantime.com.br.

[Mye04] Glenford J. Myers. *Art of Software Testing*. John Wiley & Sons, second edition, 2004.

[NPLTJ03] C. Nebut, S. Pickin, Y. Le Traon, and J.-M. Jezequel. Automated requirements-based generation of test cases for product families. In *Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on*, pages 263 – 266, oct. 2003.

[NTS+11] Laís Neves, Leopoldo Teixeira, Demóstenes Sena, Vander Alves, Uirá Kulesza, and Paulo Borba. Investigating the safe evolution of software product lines. In Ewen Denney and Ulrik Pagh Schultz, editors, *Proceedings of the 10th International Conference on Generative Programming and Component Engineering, GPCE 2011, Portland, Oregon, USA, October 22-24, 2011*, pages 33–42. ACM, 2011.

[OG05] Erika Mir Olimpiew and Hassan Gomaa. Model-based testing for applications derived from software product lines. In *Proceedings of the 1st international workshop on Advances in model-based testing*, pages 1–7, New York, NY, USA, 2005. ACM.

[oSE12] National Institute of Software Engineering. National institute of software engineering, February 2012. http://www.ines.org.br/.

[PBvdL05] K Pohl, G Buckle, and F van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques 2005*. Springer, 2005.

[Pfl94] Shari Lawrence Pfleeger. Experimental design and analysis in software engineering, part 1: The language of case studies and formal experiments. *ACM SIGSOFT Software Engineering Notes*, 19(4):16–20, October 1994.

[PM06] Klaus Pohl and Andreas Metzger. Software product line testing. *Commun. ACM*, 49, December 2006.

[Pro12] Hats Project. eshop product line, 2012. http://www.hats-project.eu/node/206.

[R D08] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. ISBN 3-900051-07-0.

[RKPR05] Andreas Reuys, Erik Kamsties, Klaus Pohl, and Sacha Reis. Model-based system testing of software product families. In Oscar Pastor

and João Falcão e Cunha, editors, *Advanced Information Systems Engineering, 17th International Conference, CAiSE 2005, Porto, Portugal, June 13-17, 2005, Proceedings*, volume 3520 of *Lecture Notes in Computer Science*, pages 519–534. Springer, 2005.

[Run03] Per Runeson. Using students as experiment subjects - an analysis on graduate and freshmen student data. In *Proceedings of the 7th International Conference on Empirical Assessment in Software Engineering. Keele University, UK*, pages 95–102, 2003.

[RW06] Rudolf Ramler and Klaus Wolfmaier. Economic perspectives in test automation: Balancing automated and manual testing with opportunity cost. In Hong Zhu, Joseph R. Horgan, Shing-Chi Cheung, and J. Jenny Li, editors, *Proceedings of the 2006 International Workshop on Automation of Software Test, AST 2006, Shanghai, China, May 23-23, 2006*, pages 85–91. ACM, 2006.

[Sak92] R. M. Sakia. The box-cox transformation technique: A review. *Journal of the Royal Statistical Society. Series D (The Statistician)*, 41(2), 1992.

[Sán11] Iván Sánchez. Latin Squares and Its Applications on Software Engineering. Master's thesis, Federal University of Pernambuco, Recife, Brazil, 2011.

[SLS07] Andreas Spillner, Tilo Linz, and Hans Schaefer. *Software Testing Foundations: A Study Guide for the Certified Tester Exam, 2nd Edition*. Rocky Nook, 2007.

[Sta07] Miroslaw Staron. Using students as subjects in experiments–A quantitative analysis of the influence of experimentation on students' learning proces. In *CSEE&T*, pages 221–228. IEEE Computer Society, 2007.

[TTK04] Antti Tevanlinna, Juha Taina, and Raine Kauppinen. Product family testing: a survey. *ACM SIGSOFT Software Engineering Notes*, 29(2), 2004.

[WRH+00] Claes Wohlin, Per Runeson, Martin Hösrt, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering*. Kluwer Academic Publishers, 2000.