UNIVERSIDADE FEDERAL DA BAHIA
INSTITUTO DE MATEMÁTICA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

COMBINATORIAL INTERACTION TESTING TOOLS FOR SOFTWARE PRODUCT LINES ENGINEERING: A COMPARATIVE ANALYSIS

Denivan do Carmo Campos da Silva

DISSERTAÇÃO DE MESTRADO

Salvador - BA
28 de Novembro de 2018

DENIVAN DO CARMO CAMPOS DA SILVA

COMBINATORIAL INTERACTION TESTING TOOLS FOR SOFTWARE PRODUCT LINES ENGINEERING: A COMPARATIVE ANALYSIS

Esta Dissertação de Mestrado foi apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal da Bahia, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Ivan do Carmo Machado

Salvador - BA
28 de Novembro de 2018

TERMO DE APROVAÇÃO

DENIVAN DO CARMO CAMPOS DA SILVA

COMBINATORIAL INTERACTION TESTING TOOLS FOR SOFTWARE PRODUCT LINES ENGINEERING: A COMPARATIVE ANALYSIS

Esta Dissertação de Mestrado foi julgada adequada à obtenção do título de Mestre em Ciência da Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação da Universidade Federal da Bahia.

Salvador - BA, 28 de Novembro de 2018

Prof. Dr. Ivan do Carmo Machado
Universidade Federal da Bahia

Prof. Dr. Wesley Klewerton Guêz Assunção
Universidade Tecnológica Federal do Paraná

Prof. Dr. Rodrigo Rocha Gomes e Souza
Universidade Federal da Bahia

I dedicate this work first to God, to my family and friends.

AGRADECIMENTOS

Em primeiro lugar, agradeço a Deus, a Ele toda honra e toda a Glória! Muito obrigado meu Deus, por tudo e principalmente por sua presença em minha vida!

Agradeço as minhas duas famílias pela força, pelo apoio dado. Agradeço a minha família Meire Matins, Simone Bandeira e Rose Fernandes pela força, incentivo e pelo cuidado mesmo de longe me amparando com seus carinhos. Agradeço a meu Pai Adaires Dorneles pela força e a preocupação da sua filha estar longe e dando forças para que meu sonho se ralizasse.

Quero agradecer ao meu orientador Ivan Machado, por incentivar para que eu entrasse no mestrado, e em seguida me orientou durante esses dois anos. Agradeço pelo acolhimento e pelas oportunidades concedidas, as experiências transmitidas e pelas suas correções que foram dados, pos me ensinaram a ser uma pessoa melhor.

Quero agradecer a professora Christina, pelo seu apoio e seu carinho de mãe para comigo. Agradeço a todos os professores do programa de pós-graduação em computação PGCOMP/UFBA, que de uma forma ou de outra me ensiram o caminho a ser seguido.

Agradeço a FAPESB - Fundação de Amparo a Pesquisa do Estado da Bahia, pelo apoio financeiro sem a bolsa não seria possível realizar este trabalho.

Não poderia deixa de agradecer aos funcionários do CEAPG-MAT/UFBA pela atenção, disponibilidade e presteza ímpares, ao longo desses dois anos. Meu Obrigado à vocês Davilene e Mayara.

Quero agradecer em especial aos meus amigos mais chegados que irmão, que o mestrado me presenteou: Beatriz Brito, Daniel Amador, Edilton Santos, Filipe Garrido, e agradecer pela força, pelas risadas e por nos apoior uns aos outros como irmãos nas horas dificéis.

Agradeço as meninas que dividiram apartamento comigo, Beatriz Brito, Beatriz Vieira, Brunna Caroline, Maria Clara, Luciana, Lucinéia, Jany e Samia Capistrano que juntas propiciamos uma ambiente familiar e divertido durante os dois anos, meu muito obrigada.

Agradeço a todos os amigos e parceiros que fizeram ou fazem parte do grupo RiSE (Reuse in Software Engineering) e do Laboratório de Engenharia de Software da UFBA pelos boms momentos que convivemos juntos.

ACKNOWLEDGEMENTS

First of all, I thank God, to Him all honor and all glory! Thank you very much, my God, for everything and especially for your presence in my life!

I thank my two families for their support. I thank my family Meire Matins, Simone Bandeira and Rose Fernandes for the strength, incentive and for the care even from afar supporting me with their affections. I thank my Father Adaires Dorneles for the strength and concern of his daughter to be far and giving the strength to make my dream come true.

I want to thank my adviser Ivan Machado for encouraging me to enter the master's degree, and then he guided me during those two years. I thank you for the welcome and the opportunities granted, the experiences transmitted and your corrections given to me, which taught me to be a better person.

I want to thank Professor Christina for her support for me. I thank all the teachers of the PGCOMP/UFBA computer graduate program, who in one way or another teach me the way to be followed.

I thank FAPESB - Foundation for Research Support of the State of Bahia, financial support without the grant would not be possible to carry out this work.

He could not fail to thank the CEAPG-MAT / UFBA employees for their unparalleled attention, availability and promptness over the course of these two years. My thanks to you Davilene and Mayara.

I would like especially to thank my closest friends as a brothers, that the master presented me: Beatriz Brito, Daniel Amador, Edilton Santos, Filipe Garrido, and thank you for the strength, the laughter and the support of each other as brothers in the hard times.

I thank the girls who shared an apartment with me, Beatriz Brito, Beatriz Vieira, Brunna Caroline, Maria Clara, Luciana, Lucineia, Jany and Samia Capistrano, who together provided a family atmosphere and fun during the two years, thank you very much.

Thanks to all the friends and partners that have made or are part of the Reuse in Software Engineering group and the Software Engineering Laboratory of UFBA for the good times that we live together.

Whenever you are asked if you can do a job,
answer yes and then learn how to do it.
—F.ROOSEVELT

RESUMO

Testar um sistema é uma atividade rotineira e desempenha um papel importante no processo de garantia de qualidade do software. Entretanto, o teste de sistemas altamente configuráveis, como as Linhas de Produto de Software (LPS), é uma atividade complexa, devido à presença de variabilidade em seu processo de engenharia, que aumenta o número de configurações de produto a se testar. Caso um defeito afete um (ou um subconjunto) destas funcionalidades, uma gama de produtos (e não apenas um, como é o caso da Engenharia de Software tradicional) será afetada. Tal complexidade implica ainda no aumento significativo do custo da atividade de testes. O uso de técnicas de teste de amostragem apoiadas por suporte ferramental podem trazer contribuições significativas para alcançar reduções de custo. Dentre as técnicas mais eficazes, destaca-se o teste de interação combinatória (CIT), que tem sido usado extensivamente para prover amostras de entradas no teste de sistemas de software altamente configuráveis. O CIT baseia-se na premissa de que muitos erros no software só podem surgir da interação de dois ou mais parâmetros. O CIT toma como entrada um modelo de configuração que define o espaço de configuração válido para o software em teste. Esse modelo geralmente inclui um conjunto de opções de configuração, cada uma delas obtendo um valor de um pequeno número de configurações discretas e um conjunto de restrições de todo o sistema entre as opções de configuração. Dado o modelo, esses métodos calculam uma matriz de abrangência t - um conjunto de configurações, no qual cada combinação válida de configurações de opção para cada combinação de opções aparece pelo menos uma vez. O sistema é então testado executando seu conjunto de testes em todas as configurações selecionadas. Esta dissertação apresenta o MERCI, um método para avaliar técnicas de teste de interação combinatória. O objetivo do MERCI é avaliar a adequação das ferramentas de CIT existentes, amplamente empregadas no teste de software tradicional, para a engenharia de LPS. Neste trabalho, realizamos uma avaliação empírica para comparar quatro ferramentas de CIT: ACTS, CATS, PICTMaster e VPTag. A análise considerou as métricas de detecção de defeitos, cobertura de testes e tempo de execução dos testes. Os resultados mostraram que o método pode servir como um bom indicador para avaliar como as ferramentas CIT poderiam se comportar em um cenário prático de testes de projetos LPS.

Palavras-chave: Linhas de produtos de software, Teste de interação combinatória, Engenharia de Software Empírica.

ABSTRACT

Testing a system is a routine activity and plays an important role in the software quality assurance process. However, testing highly-configurable systems, such as Software Product Lines (SPL), is a complex activity due to the presence of variability in its engineering process, which increases the number of product configurations to test. In case a defect affects one (or a subset) of these functionalities, a range of products (and not just one, such as in traditional Software Engineering, in which each product is built from scratch) may be affected. Such complexity also implies a significant increase in the cost of testing. The use of tool-supported sampling testing techniques could bring significant contributions to achieve reductions in such a cost. Among the most effective techniques, Combinatorial Interaction Testing (CIT) has been used extensively to sample inputs to software, and to test highly-configurable software systems. CIT is based on the premise that many errors in software can only arise from the interaction of two or more parameters. CIT take as input a configuration model that defines the valid configuration space for the software under test. This model typically includes a set of configuration options, each of which takes a value from a small number of discrete settings, and a set of system-wide constraints among configuration options. Given the model, these methods compute a *t-way* covering array - a set of configurations, in which each valid combination of option settings for every combination of t options appears at least once. The system is then tested by running its test suite in all the configurations selected. In this investigation, we aimed to analyze the effectiveness of existing tool support for CIT. To accomplish our goal, we introduced the *MERCI - a Method to Evaluate Combinatory Interaction Testing techniques*, aimed to establish a systematic means to evaluate the adequacy of existing CIT tools for highly-configurable systems testing. In this work, we performed an empirical evaluation to compare four CIT tools: ACTS, CATS, PICTMaster and VPTag. The analysis considered the metrics defect detection, test coverage and test execution length. The yielded results show that the method could be employed as a good mechanism to evaluate how CIT tools could behave in a practical SPL testing scenario.

Keywords: Software Product Lines, Combinatorial Interaction Testing, Empirical Software Engineering.

CONTENTS

LIST OF ACRONYMS

AETG Automatic Efficient Test Generator

ANOVA Analysis of Variance

APFD Average Percentage of Faults Detected

CA Covering Array

CC Cyclomatic Complexity

CIT Combinatorial Interaction Testing

FDC Fault Detection Capability

FM Feature Models

FOSD Feature-Oriented Software Development

FODA Feature-Oriented Domain Analysis

GUI Graphical User Interface

LOC Lines of Code

PICOC Population, Intervention, Comparison, Outcome and Context

PUT Product Under Testing

SLR Systematic Literature Review

SPLCAT Software Product Line Covering Array Tool

SPL Software Product Lines

SQA Software Quality Assurance

UML Unified Modeling Language

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

SPL can be defined as a "set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way" (CLEMENTS; NORTHROP, 2001). In other words, it can be considered as a set of programs that share key functionalities, created from a set of reusable parts.

The software development strategy based on SPL engineering is analogous to manufacturing, such as the automotive industry in which, from a common basis (or platform), it is possible to configure a range of products, which share functionalities, but offer options that make them different. Such options may address particular customer needs, or might even consider the needs of a whole given market segment (APEL et al., 2013).

In SPL engineering, variability management is the key strategy that provides flexibility for product differentiation and diversification. Variability management generally affects the degree to which an SPL project is successful, covering all the activities of variability representing software artifacts throughout the project life cycle (CHEN; BABAR; ALI, 2009).

While variability management enables delivering a range of products with reduced cost and time-to-market, when compared to traditional software development strategies, it is necessary to employ suitable software quality strategies in order to achieve high quality in every delivered product. In this scenario, software testing plays an important role. Software testing is intended to identify faults in the system, and guarantee that they can be corrected in a timely and effective manner, i.e., finding actual defects before delivery to the end user.

Although testing has proved its effectiveness, it is a rather arduous and laborious activity, as there are several input data combinations to test in a software project. In terms of highly configurable systems, such as SPL projects, in which variability increases the number of possible configurations, testing is even more difficult. Such a complexity is mainly due to the amount of variability an SPL must handle, what leads to an increased number of possible combinations to test (MACHADO et al., 2014).

Along the years, several tools have been proposed to handle testing in SPL engineering (LIMA-NETO et al., 2012). There is another large set of tools not developed to handle variability, but that might be suitable to test SPL projects. However, there is a few reports on the analysis of the effectiveness of existing tool support for SPL testing.

Tool support for SPL testing should be focused on reducing the set of possibilities to a reasonable and representative set of product configurations, thus reducing the test configuration space. While keeping a small sample is critical to limit the effort necessary for testing each selected configuration (HERVIEU; BAUDRY; GOTLIEB, 2011), this is particularly a combinatorial problem, that should be handled accordingly.

An affordable strategy to cope with reducing the testing space is the use of CIT techniques. CIT can be considered as a feasible strategy to reduce testing effort by selecting a set of representative products (YILMAZ et al., 2014; LOPEZ-HERREJON et al., 2015). CIT techniques have been the subject of recent studies as they are used in various domains and there is a variety of commercial and free tools to support the testing process (YILMAZ et al., 2014). They mainly leverage test configuration selection or employ test configuration prioritization approaches.

Therefore, CIT techniques play an important role in SPL engineering, as they propose mechanisms to test the interaction between the many features and parameters that compose the configuration space of software systems (PETKE et al., 2015). In this effect, this current investigation aims to ***analyze the effectiveness of CIT tools regarding their adequacy to SPL engineering***. In order to reach such a goal, we performed a set of empirical evaluations to compare CIT tools[1].

1.1 MOTIVATION

Software testing is a technique used to detect faults efficiently in complex software systems (AMMANN; OFFUTT, 2016). Despite its importance, testing software is a non-trivial, exhausting, and costly activity, and it is deemed to be a bottleneck in the software development process. In this effect, there is an enormous attempt to reduce its related costs. Unfortunately, many companies leave test aside, in a *pseudo*-cost reduction strategy. Indeed, it is better to reduce the cost by improving the quality of the software system, and it encompasses fixing any likely issue still during the development.

The above statement still holds true in the SPL engineering context. Due to the degree of complexity variability poses, some authors consider that testing is also a bottleneck in SPL engineering, being even more expensive than testing individual systems (KOLB, 2003).

There are some recent reports discussing the state-of-the-art SPL testing practices (LAMANCHA; USAOLA; VELTHUIS, 2009; NETO et al., 2011; ENGSTROM; RUNESON, 2011; MACHADO et al., 2014). They synthesize available evidence on this field, and discuss existing solutions and leverage the gaps still to bridge. All of them observed an increased interest in improving SPL testing practice, by providing both testing processes and mechanisms to automate such processes.

Although several studies in the SPL engineering field have been proposed along the

[1]Widely used CIT tools are available at http://www.pairwise.org/tools.asp

years, we observed a lack of empirical studies attempting to discuss the benefits and drawbacks of existing tool support, in the light of widely accepted testing measurements.

Lima-Neto et al. (2012) carried out a systematic literature review on SPL testing tools. The main objective was to identify how the available tools could support existing SPL testing processes. The literature review leveraged a set of supported features, and which should be present in all tools based on their priorities. It is worth mentioning that the study was not focused on discussing the tools' effectiveness to any extent.

In this effect, there is a need for more in-depth studies of testing tools for SPL engineering. In this current research, we narrow down the focus to the analysis of CIT tools in terms of their effectiveness to test SPL projects. Tool effectiveness could be observed by analyzing several metrics, which mainly indicate their readiness for finding faults (MACHADO et al., 2011).

In order to reach such a goal, we carried out a set of empirical evaluations to compare CIT tools. Although the reports on the state-of-the-art SPL testing practice (LAMANCHA; USAOLA; VELTHUIS, 2009; NETO et al., 2011; ENGSTROM; RUNESON, 2011; MACHADO et al., 2014) discuss topics that need particular attention from both research and practitioners communities - such as quality attribute testing considering variability, the traceability between development and test artifacts, and the management of test assets throughout the whole SPL development life cycle - it is worth to mention that these are out of scope in this investigation.

1.2 OBJECTIVES

1.2.1 General objective

In this current investigation, we aim to evaluate the effectiveness of the CIT tools regarding their adequacy to the development of SPL projects.

1.2.2 Specific objectives

- Review the literature in order to leverage state-of-the-art practices in SPL testing;
- Identify testing tools applied to the SPL engineering context;
- Provide an empirical comparison among testing tools with respect to the following metrics: defect detection capability, code coverage, and test execution length.

1.3 EXPECTED CONTRIBUTIONS

As a result of the work presented in this dissertation, the following contributions can be highlighted:

- Evidence on state-of-the-art SPL testing practice;
- A method to empirically evaluate CIT tools;
- An empirical study on the adequacy of CIT tools for software testing in SPL projects.

1.4 DISSERTATION STRUCTURE

The remainder of this dissertation is structured as follows:

- **Chapter 2:** This chapter presents the foundations on SPL and Software Testing. Such a discussion is rather important to establish the underlying concepts necessary for the understanding of this dissertation.
- **Chapter 3:** This chapter presents an updated Review on SPL testing, based on a previously published work (MACHADO et al., 2014), which is extended by incorporating studies published from 2014 up to 2016.
- **Chapter 4:** This chapter introduces the MERCI, our proposed method to Evaluate CIT Tools for SPL engineering. The proposed method was empirically evaluated with the ACTS, CATS, PICT Master and VPTag tools, which resulted in a reduction in the number of products.
- **Chapter 5:** This chapter presents the concluding remarks, by including a discussion on the related work, and by pointing out directions for future investigations in the field of SPL testing.

Chapter 2

THEORETICAL BACKGROUND

This chapter presents fundamental concepts which are relevant to follow up this work: SPL engineering, Feature-oriented SPL, software testing, SPL testing, testing tools and metrics related to software testing.

2.1 SOFTWARE PRODUCT LINES

In the early years of software development, the products were commonly built from scratch, and reuse was merely opportunistic. For instance, it is not uncommon for small and medium-sized companies to adopt a *clone-and-own* strategy, by copying, adding or removing functions from existing products (RUBIN; CHECHIK, 2012). This approach leads to building *ad-hoc* product portfolios of multiple yet similar variants (FISCHER et al., 2014). With the growth of such products portfolio, the management of variations among them becomes more complex (SHATNAWI; SERIAI; SAHRAOUI, 2015). Whenever a new customer decided to acquire a software product, it was handcrafted to meet its particular needs. However, with the software evolution numerous people could afford buying several types of products, similarly to what occurred with the automobile domain. Such a domain is widely known by the concept of mass-production, introduced by Henry Ford as a strategy to deliver cars much cheaper than individual creation, in what was coined as ***product line engineering*** (POHL; BOCKLE; LINDEN, 2005).

As with the automobile domain, the software systems domain has also undergone evolution (POHL; BOCKLE; LINDEN, 2005). The mass standardization process, also called as ***mass customization***, i.e., *"the large-scale production of goods tailored to individual customer's needs"*, was an important step to advance Software Engineering practices. Due to mass customization, the software industry is prone to improve quality by adopting strategies that emphasize proactive reuse, as a means to deliver products more efficiently, quickly, reliably, and cheaply.

SPL engineering is a paradigm that has been largely adopted by companies in the last decades. It provides a means to systematically reuse software artifacts, and build sets of products which share functionalities, allowing them to reach improvements in

time-to-market, productivity, quality, and cost reduction, as well as other positive gains (CLEMENTS; NORTHROP, 2001; APEL et al., 2013).

An SPL captures commonalities among a range of software products, so that product development would focus on product specific issues rather than on issues that are common to all products (GURP; BOSCH; SVAHNBERG, 2001). The product specific parts may be referred to as variability, which is a key artifact for SPL engineering (BEUCHE; PAPAJEWSKI; SCHRÖDER-PREIKSCHAT, 2004).

A successful management of variability in software artifacts leads to better customizable software products that are in turn likely to result in higher market success: in the information systems domain, the products are more easily adaptable to the needs of different user groups; in the embedded systems domain, the software can be easily configured to work with different hardware and environmental constraints (GURP; BOSCH; SVAHNBERG, 2001), and so on.

In SPL engineering, variability may be expressed in terms of features, basic building blocks for specifying products (GURP; BOSCH; SVAHNBERG, 2001). A feature can be defined as the distinctive characteristic of a system (LEE; KANG; LEE, 2002). In most trivial cases, a small number of features could result in a small number of possible product configurations. However, the number of product configurations increases as the number of features increases (MACHADO et al., 2014). Exhaustively testing all configurations for SPL projects is not feasible in practice. Applying existing testing techniques to test each product separately is also difficult and may requires enormous adaptations so they could handle variability (GALINDO et al., 2016).

2.1.1 SPL Motivation and Benefits

SPL engineering bring a diversity of benefits for software development, as follows (POHL; BOCKLE; LINDEN, 2005): quality improvement, reduction of time-to-market, reduction of maintenance effort, and reduction of development costs. These are discussed next.

- **Quality improvement.** Many products are tested and analyzed through artifacts and their reusable assets, that have their quality assessed considering different contexts, leading to overall higher product quality.
- **Reduction of time-to-market.** Time-to-market is one of the main success factors for an SPL, which requires a high initial investment, when compared to single systems. However, as the time passes, the time to deliver a product instance to the market is significantly reduced, as several artifacts can be reused in novel products.
- **Reduction of maintenance effort.** Whenever software artifacts are changed or new artifacts are included in the core asset base (for reusable artifacts), those changes are propagated to all products, making maintenance and evolution simpler and cheaper compared to maintaining and evolving products separately.

- **Reduction of development costs.** Cost reduction is an essential reason to introduce SPL. Instead of developing every new product from scratch, the artifacts can be reused in different types of systems, generating benefits such as the reduction of costs.

2.1.2 Essential SPL activities

Figure 2.1 shows the three key SPL engineering activities, according to the SEI/CMU[1]: (i) core asset development, (ii) product development, and (iii) management. It presents the three activities in the form of rotating arrows, each rotating circle represents one of the essential activities. This means that the three activities are connected to each other in a constant movement showing that all of them are essential and highly interactive and that they can occur in any order. The arrows indicate that the product is not only developed by the ***core asset*** activity, but also that revisions of existing core assets or even new core assets can, and do so more frequently, evolve from the ***product development*** activity (CLEMENTS; NORTHROP, 2001).

For Clements and Northrop (2001), core assets can be either developed or acquired, making a strong feedback loop between core assets. Management is needed everywhere. It may control the development of the essential assets and products. Moreover, it offers processes and activities necessary for the three essential activities to work together. Following we describe each of these activities (CLEMENTS; NORTHROP, 2001):

Figure 2.1 Essential Product Line Activities (CLEMENTS; NORTHROP, 2001)

- **Core asset development.** It aims to define commonality and variability by establishing reusable artifacts and production capacity for products. Figure 2.2 shows the core asset development activity together with its outputs and contextual factors. This activity is also commonly referred to as ***domain engineering*** (POHL; BOCKLE; LINDEN, 2005).

[1] ⟨https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=513798⟩

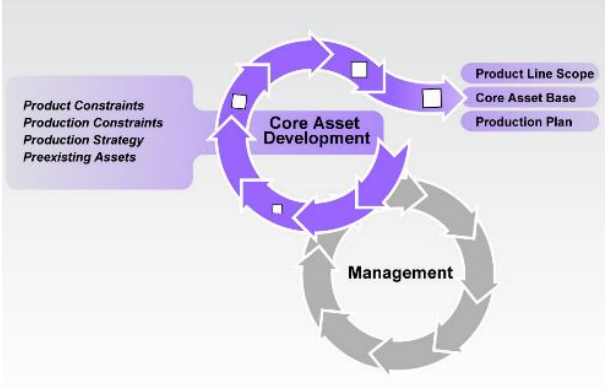

Figure 2.2 Core Asset Development (CLEMENTS; NORTHROP, 2001)

In Figure 2.2, the spinning arrows show that there is no one-way causal relationship from the context to the outputs.

- **Product development.** This activity is responsible for creating individual products by reusing core assets. In addition, it gives feedback to core asset development, as a means to evolve the core asset base, as the products are evolving. Figure 2.3 shows the product development activity along with its results and contextual factors (CLEMENTS; NORTHROP, 2001). This activity is also known as ***application engineering*** (POHL; BOCKLE; LINDEN, 2005).

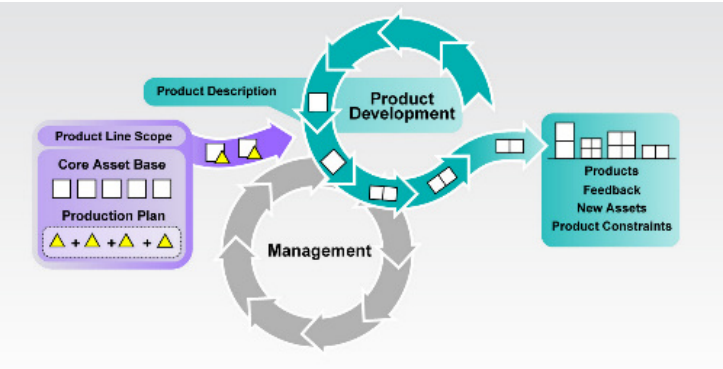

Figure 2.3 Product Development (CLEMENTS; NORTHROP, 2001)

Alike in core asset development, the rotating arrows in Figure 2.3 indicate iteration and relationships involved. In this activity it is mandatory to provide feedback on problems or deficiencies found in core assets (CLEMENTS; NORTHROP, 2001).

- **Management.** It plays a critical role, and includes both technical and organizational management issues. *"Management at the technical and organizational levels should be strongly committed to the effort of the SPL"* (CLEMENTS; NORTHROP, 2001). Technical management is responsible for controlling requirements and for

coordinating asset and product development, ensuring that those who construct essential assets and products participate in the required activities (CLEMENTS; NORTHROP, 2001).

In turn, organizational management identifies constraints and production strategies. Clements and Northrop (2001) defines *"organizational management as the authority responsible for the success or ultimate failure of the SPL effort"*. All in all, it is necessary an adequate organizational planning, investment, direction and strategic thinking that looks beyond a single product.

2.2 FEATURE-ORIENTED SOFTWARE PRODUCT LINES

An SPL aims to assist the software development industry, based on a set of reusable parts, generating a software product based on the requirements of a particular customer (APEL et al., 2013).

As mentioned in the previous section, SPL engineering is a growing paradigm that helps organizations develop their products from reusable core assets. Products are built from the analysis of a domain, using the Feature-Oriented Domain Analysis (FODA) method that supports reuse at the functional and architectural levels (KANG et al., 1990).

In a domain, features are particularities or characteristics visible to the user. They are necessary for the development of a set of products that define the specific domain. Features are also used to define the domain in terms of the mandatory, optional or alternative elements composing those related systems, from which define the common aspects of the domain, as well as the differences between related systems (KANG et al., 1990).

It is natural and intuitive for customers to specify what features products should contain in terms of features, to express possible similarities or variability in terms of features. According to Kang et al. (1990) "a feature-based model thus provides a basis for developing, parameterizing, and configuring reusable assets".

2.2.1 Variability Modeling

Likewise variability management, variability modeling is also an important activity for SPL engineering. The most common strategy to model variability in an SPL is to use a feature model, which is a tree-like structure, whose nodes describe its features, and the edges indicate the different forms of relationships between them (APEL et al., 2013; BENAVIDES; SEGURA; RUIZ-CORTÉS, 2010). A feature model represents the products in an SPL by means of the dependencies among features and the interrelationships among variation points (JOHANSEN; HAUGEN; FLEUREY, 2011). As a matter of fact, feature models have become the *de facto* standard variability model, as they provide compact representations of all products of an SPL in terms of their features (BEUCHE; PAPAJEWSKI; SCHRÖDER-PREIKSCHAT, 2004).

Figure 2.4 shows a sample feature model, which represents the domain of *Mobile Phone* - the root node. It encompasses the representation of four types of features, as

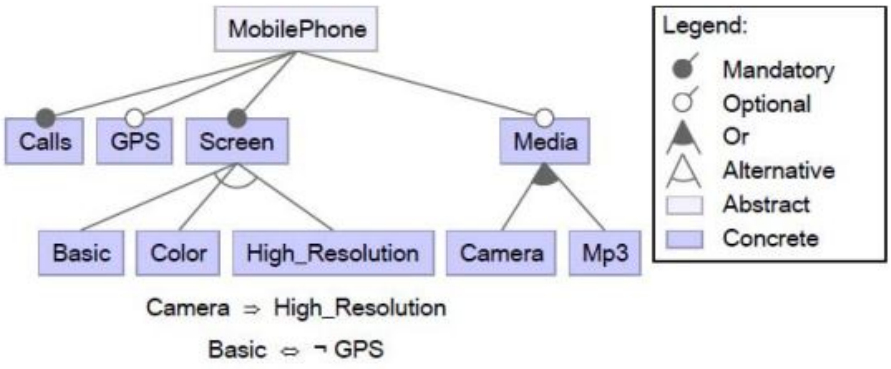

Figure 2.4 Sample Feature Model.

follows:

- **Mandatory** - features that must be present in all product configurations. In the example, they are represented by the features `Call` and `Screen`;
- **Optional** - features that could be either selected or not in a product configuration. In the example, the optional features are `GPS` and `Media`;
- **Alternative** - grouped features which are mutually exclusive. In the sample, the feature `Screen` is a parent-feature of a group of alternative features. In this case, only one feature in such a group could be selected, either `Basic, Color` or `High_Resolution`.
- **Or-features** - in this group of features, at least one of them must be selected. In the sample feature model, the feature `Media` is a parent-feature, from which at least one of the sub-features `camera` or `MP3` must be selected.

Besides, a Feature Model also includes **Cross-Tree Constraints**, i.e., a feature model may define which features are selected and their dependencies, that is, the constraints. These constraints are called cross-tree constraints, but can be just as a table of possible combinations.

2.3 FUNDAMENTAL CONCEPTS OF SOFTWARE TESTING

Any software may contain several types of problems, one of which is to deliver a product different from what it was expected. This is a common effect of human errors, namely because the development activities depend on the ability of interpretation and execution of the people who construct them (DELAMARO; MALDONADO; JINO, 2007).

In this effect, Software Quality Assurance (SQA) plays an important role at defining a set of strategies to reduce the probability of failure of a system. SQA is “a set of activities that define and assess the adequacy of software processes to provide evidence that

establishes confidence that the software processes are appropriate and produce software products of suitable quality for their intended purpose" (BOURQUE; FAIRLEY et al., 2014).

Some concepts and terminologies are important to understand and apply in a software testing process. They are discussed along this section.

Validation and Verification. These activities are intended to ensure that the software works as specified. The **verification** is intended to explore whether the software conforms to the requested specifications, and **validation** is the process of confirming that the system is appropriate and consistent with the requirements. Software testing is part of a broader topic that is often referred to as validation and verification. In other words, if the products of a given stage of the software development process meet the requirements established during the previous phase (AMMANN; OFFUTT, 2016).

Error, Fault, and Failure. The distinction between error, fault, and failure is an important concept related to software testing. According to Burnstein (2006), **error** refers to a misconception or misunderstanding carried out by a developer, a manifestation of some failure. **Fault** is a software anomaly that can cause it to malfunction, and not according to its specification, a static defect. It is a result of an error. **Failure** comprises the inability of a system or a component to perform its required functions as set forth by the requirements.

In SQA, there are two major categories of activities (LUO, 2001):

- **Static analysis** - focuses on the range of methods that are used to determine or estimate software quality without reference to actual executions. Techniques in this area include code inspection, program analysis, symbolic analysis, and model checking;

- **Dynamic analysis** - deals with specific methods for ascertaining and/or approximating software quality through actual executions, i.e., with real data and under real (or simulated) circumstances. Techniques in this area include synthesis of inputs, the use of structurally dictated testing procedures, and the automation of testing environment generation.

There are several well-acknowledged definitions in the literature for software testing. According to Graham (1993), to deliver a product in good quality, it is necessary to test it. Testing is the only way to assess and preserve software quality in a real environment. Ammann and Offutt (2016) states that "testing is the main method the industry uses to evaluate software development".

Pohl, Bockle and Linden (2005) argue that software testing is "the process that generates evidence of defect discovery in software systems. Its main goal is to reveal the existence of faults in the Product Under Testing (PUT) during its different development phases". In any quality assurance process, testing is a necessary part.

In order to test a project, it is necessary to run a set of test cases, where the result will be evaluated to determine whether the performance conforms to its specification and that it meets customer expectations (JALOTE, 2012).

Test cases are sets of input values, execution conditions, and expected results developed for a determined test goal. Test cases executions can be either positive (or valid) or negative (or invalid). The positive test cases are those that confirm that the software does what it should do, while the negative ones are those that the actions performed are unforeseen, i.e., the software does not do what it should do (IEEE, 2008).

2.3.1 Test levels

Several techniques and methods are used to test a software, at different levels. A *test level* can be referred to as the granularity of the items to be tested and the requirements used as the test reference (POHL; BOCKLE; LINDEN, 2005). The most widely accepted test levels are unit, integration, and system test. These are defined next:

- **Unit test** - It has as objective to identify faults directed to the logic and implementation in each unit, ensuring that its algorithmic aspects are implemented correctly. The behavior of a component, method, or class is validated through the unit test.

- **Integration test** - It aims to validate the behavior of two or more units to check whether they also work properly when combined.

- **System test** - Its main objective is to run and validate the behavior of the system as a whole. The system tests validate the implemented system against the specification.

  The desired behavior of the system is defined by the requirements, where each test has a distinct purpose, all work to verify that the elements of the system have been properly integrated and perform the functions assigned to them.

2.3.2 Testing Techniques

There are different testing techniques, these techniques reveal the quality aspects of the software systems. In this section, we discuss *functional* and *structural* testing techniques, the most widely accepted ones in both academia and industry (LUO, 2001; DELAMARO; MALDONADO; JINO, 2007).

2.3.2.1 Functional testing

This is also known as ***black box*** testing. It is intended to create a set of data inputs and evaluate the outputs to test all the functional requirements of a program.

The functional test emphasizes the external behavior of the software entity. As a "black box", the tester is not aware of the system implementation and the software is classified according to the user's view. The following are some black box criteria:

Equivalence class partitioning: Is the representation of a set of valid or invalid states for input conditions, and are defined according to some guidelines, such as a valid class or two invalid classes (DELAMARO; MALDONADO; JINO, 2007).

Boundary value analysis: Is a criterion used with the equivalence partitioning set that works correctly for a set of values in an equivalence class, which exploit the boundary conditions to find the largest number of defects. The data is selected in the threshold form of each class that the equivalence class is expelled from the point of view of input and output (DELAMARO; MALDONADO; JINO, 2007; JALOTE, 2012).

2.3.2.2 Structural testing

This is also known as ***white box*** testing. Unlike the preceding one, this is an implementation-based technique, as it emphasizes the internal structure of the software entity.

The goal of structural testing is to select the test cases and specific points to run in the software entity, such as specific instructions, program branches, or paths (PRESSMAN, 2005; SOMMERVILLE, 2007). This makes the white box testing rather important, as it explores logical paths of the program, where logic errors might occur.

Structural tests may be applied at different stages during the software development process, generating an expected result. These results are evaluated in a set of coverage criteria.

2.3.3 Mutation testing

Mutation testing is a fault-based structural test technique that uses the code structure to guide the testing process to evaluate the quality of the tests that will be applied to a system (DEMILLO, 1980; OFFUTT; UNTCH, 2001).

The mutation analysis process is the process of rewriting the source code in small ways, that is, it causes software failures creating multiple versions of the original software, where each version created contains a failure. After the creation of the mutants, the existing test cases are used to perform the defective versions (mutants) to highlight the defective ones (to kill a mutant) of the original software. The test designer aims to achieve a mutation score of 100 percent, that is, that all mutants (ie, all failures) were detected (OFFUTT; UNTCH, 2001).

According to Jia and Harman (2009), we can highlight several types of mutants that can be created. The simplest and most common are the first-order mutants, that is, those that have only one introduced fault.

One of the fault-based testing strategies is mutation testing. There are many variations of mutation testing such as weak mutation (HOWDEN, 1982), interface mutation (DELAMARO; MALDONADO; MATHUR, 1996) and specification-based mutation testing (MURNANE; REED, 2001).

2.3.4 Combinatorial Interaction Testing (CIT)

CIT is a recognized software testing technique, which aims to test interactions between values of software parameters (COHEN et al., 1997). CIT works by modeling a system under test as a set of factors to detect faults caused by the interactions of various input parameters of the system in an efficient and effective way (YILMAZ et al., 2014; LOPEZ-HERREJON et al., 2015).

CIT is a black-box sampling technique derived from the statistical design field of experiments (COHEN et al., 1997). It works by generating samples that cover a representative set of all possible value combinations between any set of parameters. In order to accomplish its goals, the CIT technique must comply with coverage criteria and the sample must contain some specified combinations, with the purpose of reducing the final number of test cases (COHEN; DWYER; SHI, 2008).

In CIT, a Covering Array (CA) must be built. It is a two-dimensional array, where each column represents a feature and each row represent a test configuration. The strategy is to construct the CA based on `t`-wise strength, where `t` indicates the coverage strength `(1,2,3,..,n)` and it will determine the number of feature combinations that should appear at least once in the CA (KUHN et al., 2009).

When observing its adequacy for SPL engineering, CIT could enable testing several product configurations, by leveraging a representative set of products (or even subsets of features) to test. A generally accepted idea is to select a small subset of products where possible feature interactions are most likely to occur. This is the main principle behind the CIT techniques, given that not all input or configuration options contribute to every fault in a system. In practice, most failures are caused by interactions of `n` factors (KUHN et al., 2009). In this effect, several tools handle CIT by employing a factor `n=2`, in what is called **pairwise testing**, from which the interaction of pairs is more cost-effective at finding important issues in a software system (BACH; SCHROEDER, 2004).

CIT has successfully been applied in test input generation and parameter combinations of single systems development (ENGSTROM; RUNESON, 2011). CIT enables a significant reduction of the number of test cases without compromising functional coverage. There are several studies in the SPL field applying CIT techniques to reduce the overall testing effort (GALINDO et al., 2016).

Despite CIT could be deemed as an affordable strategy to reduce test sets, there are several issues that prevent its larger application in practice. In this effect, we should move to a new understanding in which CIT is used in new ways, being it for alternative notions of input, such as event sequences and SPL, among others. Researchers and practitioners have been working towards facilitating the application of CIT in practice (YILMAZ et al., 2014).

In order to improve such understanding Yilmaz et al. (2014) formalized the CIT concepts, by presenting a set of phases the technique should encompass. In their standpoint, the CIT can be divided into four main phases, as Figure 2.5 shows.

The first group represents the word **WHAT**, which includes two phases: modeling and sampling. What does “test” mean - what are the characteristics of the PUT and what are the inputs against which it should be tested?

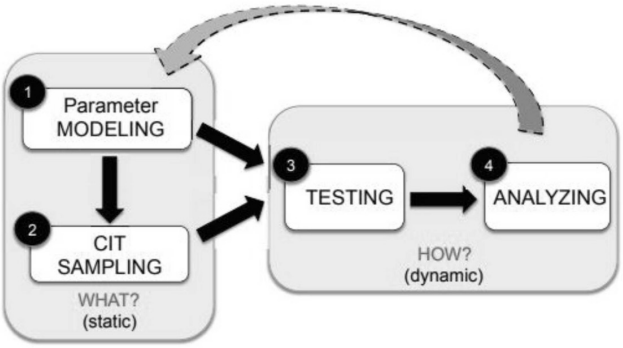

Figure 2.5 Four Phases of CIT (YILMAZ et al., 2014)

Modeling involves the determination of the characteristics of the model to be modeled, that is, inputs, configurations, and sequences of operations. **Sampling** refers to the process or algorithm by which it is possible to determine a means to cover the model generated in the first phase, that is, all pairs or factors. Sampling and modeling are typically static phases, performed at the beginning of the process.

The second group includes the **testing** and **analyzing** phases, which together address the **HOW** tests must be performed - which runs the tests and then examining the yielded results, so it is possible to understand whether a test case has either passed or failed. Such phases are driven by several processes, and are usually carried out over a longer period of time, when compared to the preceding group. The tests can be carried out in batch mode, incrementally, or in an adaptive way. In some eventualities, developers can use the testing and analysis phases to provide feedback, and refine subsequent modeling and sampling activities.

2.4 SOFTWARE PRODUCT LINES TESTING

Software testing is still a widely-accepted quality assurance technique. As mentioned earlier in this work, software testing encompasses a set of important activities to identify the defects and ensure that the finished products can function properly as specified, both in terms of specified functionalities and with a minimum probability of occurrence of an undesirable outcome.

SPL testing is aimed at handling tests from two main perspectives, namely, is must examine the core assets, the individual parts, and the interaction between them, which are shared in products derived from an SPL (NETO et al., 2011). The SPL testing activities embrace activities from validating initial requirements to activities performed by customers in the acceptance of a product, and confirms that testing is still the most effective method of quality assurance. Next, we discuss each of these perspectives, hereinafter called **SPL testing interests** (MACHADO et al., 2014)..

The first interest deals with the **selection of products to test**. It seeks to understand how products are selected from a large set of possible products, and how each selected product should be tested. Although it is necessary to test and analyze all combinations of possible features, this is not feasible for large-sized SPL projects. The ultimate goal of such an interest is to reduce the set of possibilities for a reasonable and representative set of product configurations.

The second interest seeks to guarantee that **end-product functionalities would work as specified**. This interest deals with the systematic reuse of test artifacts and test results as well. This is an affordable means to reduce the overall test effort and avoid retesting every already tested features, for each new product configuration. Test assets are designed to test the functionalities of features that will compose the products.

In either one or another interest, there is a particular problem, namely the number of test inputs to consider, which could increase exponentially as the number of features increases. Designing and selecting effective test sets, considering the likely amount of test inputs, play an important role for SPL testing.

2.4.1 Testing in core asset development

When we refer to customization, we are emphasizing the concept of variability, that is, we use common artifacts and differences in applications in terms of requirements, architecture, components, and test artifacts in an SPL. Variability, a key element in SPL engineering, is also the source of complexity, and the variation points is one of the sources of faults (POHL; BOCKLE; LINDEN, 2005).

In core asset development, the test artifacts are created so that they can be reused in product testing (POHL; BOCKLE; LINDEN, 2005). Therefore, core asset testing is an opportunity to save effort, as it is not always necessary to (re-)test everything again, but only reuse the previously obtained results instead (MACHADO et al., 2011).

Notwithstanding, there is a big challenge concerning core asset testing, given that several products would reuse the core assets in a range of distinct product configurations. Hence, not only the individual core assets but, to a certain extent, their integration, should be tested as well (MACHADO et al., 2011). Otherwise, a defect would be spread over a range of products, what may directly impact the promised SPL benefits.

2.4.2 Testing in product development

Carrying out tests exclusively on either core asset or product development phases is not always safe. An important assumption is that the core assets were previously created, documented, and tested during core asset development, and as such they may be reused seamlessly, and no testing is required as well. However, in practice, even if the products are derived from the same core asset base, it is necessary to make sure that all the features are working properly in the integrated environment, and also that all expected requirements are working according to the product specifications (MCGREGOR, 2001). Therefore, it is strongly recommended to carry out tests in both SPL phases.

2.5 SPL TESTING TOOLS

The tools are used to verify and validate the software, which helps in the detection of errors. If the defect is detected and solved during the development phase, many faults can be avoided. By using the tools, the reduction of test costs can be quite significant (SOMMERVILLE, 2007).

There are different types of testing tools available for different purposes. In SPL engineering, they require specific tools to assist in the management of reusable tests and automation of test execution (LIMA-NETO et al., 2012).

The tools discussed in the work of Lima-Neto et al. (2012) are designed to support only one specific test level, and that none of the tools support all the functionality of a general SPL life cycle testing process.

If compared to other systems, an SPL may be a huge system, as it encompasses not only one but a range of distinct products due to the degree of variability. In this effect, it is prominent to count on support tools to both developers and testers could scan the large volume of source code of SPL projects in a straightforward and cost-effective manner. In practice, test tools can support testing large-scale SPL to achieve their goals (EDWIN, 2007; LIMA-NETO; ALMEIDA; MEIRA, 2012).

For Tevanlinna, Taina and Kauppinen (2004), SPL requires support for automated tools, but also requires support for robust and specific tools. The use of test tools is relevant because it decreases the effort when reusing test assets and makes the complex test process more manageable in the construction of products that contain variability. We next describe some of the widely-reported software testing tools in the literature.

- *MoSo-PoLiTe-Model-based Software Product Line Testing framework:* is a set of tools that contains pairwise configuration selection component based on feature model. MoSo-PoLiTe is a framework that provides a test framework for SPL (OSTER; MARKERT; RITTER, 2010; OSTER et al., 2011).

- *SPLCAT-Software Product Line Covering Array Tool:* It is a tool that generates a set of T-wise tests. It is a tool that implements various algorithms for covering features models arrays in SPL (JOHANSEN, 2013).

- *CASA - Covering Arrays by Simulated Annealing:* is a Simulated Annealing algorithm that is designed to generate T-wise coverage matrices for SPL projects. It is a tool in an improved metaheuristic search for restricted interaction tests (GARVIN; COHEN; DWYER, 2009).

- *ACTS - Automated Combinatorial Testing For Software:* is a test-generation tool for constructing t-wise (or pairwise) combinatorial test sets with input parameter values, including support for constraints and variable force tests. The tool provides command line interfaces and Graphical User Interface (GUI) (BORAZJANY et al., 2012).

- *PICT Master - Pairwise Independent Combinatorial Testing tool* is a tool that implements an algorithm similar to an optimized for speed Automatic Efficient Test Generator (AETG) (CZERWONKA, 2006).

- *CATS - Constrained Array Test System* is a tool for generating test cases, aiming to improve the coverage and efficiency of the test. The tool uses pairwise to generate the test cases by providing complete statistical test plans with a small number of test cases (SHERWOOD, 1994).

- *Pairwiser:* The pairwise test tests all possible combinations of a set of variables. In the pair test, a set of test cases is generated that covers all combinations of the selected test data values for each pair of variables (BACH; SCHROEDER, 2004).

2.6 SOFTWARE METRICS

When it comes to measurements in software engineering, we are dealing with metrics to measure the intermediate or final product. They measure the characteristics of the software development process (TRAVASSOS; GUROV; AMARAL, 2002).

According to (PRESSMAN, 2005) the metrics are intended to construct indicators that facilitate decision making, and are used to "degree" specific attributes of the software. Quality metrics are applied to software development, and it becomes complex because of the nature of the development activities and the range of measurement techniques that are used in the marketplace.

Test metrics are used to evaluate the effectiveness of the tests. Coverage metrics aim to provide coverage for the program, while defect metrics focus on errors found instead of the tests themselves (PRESSMAN, 2005).

2.6.1 Code coverage

Test coverage metrics that check, in different ways, the percentage that your code is testing. These metrics are intended to show which part is not covered. It is very difficult to say how much software is "well tested" (YANG; LI; WEISS, 2009).

Code coverage is a criterion used by test coverage. Test coverage is a quantitative only metric, i.e., it is not used to measure the quality of a set of tests. The objective of this metric is only to demonstrate by percentage, the amount of code that was executed during the test with effectiveness, taking into account points with no or almost no test performed (DELAMARO; MALDONADO; JINO, 2007).

Code coverage ensures a quantification of test development related to coverage. To select the tests that require greater incremental gain in coverage is a way to prioritize testing. Code Coverage says which code is not tested, but tells you precisely what code is actually tested (JONES; HARROLD, 2003).

The measure taken for a set of tests results in the percentage of code, and thus one or more coverage criteria is used. They are usually defined as rules or requirements to meet the coverage criteria (DELAMARO; MALDONADO; JINO, 2007).

2.7 CHAPTER SUMMARY

This chapter presented the fundamental concepts and definitions about SPL engineering and its essential activities, the motivations and benefits of using SPL, managing

variability and quality of SPL.

We also presented fundamental concepts about SPL testing, where an overview of software testing was presented, discussing the main concepts, processes, technical strategies, approaches, and so on, in order to define the conceptual base for this dissertation.

Chapter 3

AN UPDATED REVIEW ON STRATEGIES FOR SPL TESTING

In this chapter, we present an updated and extended systematic literature review on SPL testing strategies, carried out to synthesize currently available evidence on test strategies, metrics, techniques, and tools. This work was based on the study performed by Machado et al. (2014) and included primary studies published from the years 2014 to 2016. Despite the characteristic of an extension study, it is worth mentioning that this study was focused on understanding the role of tools and metrics for the SPL testing field, and how they have been employed along the last years. In the former study, the authors did not include any discussion surrounding such important issues.

3.1 SYSTEMATIC LITERATURE REVIEW METHOD

A Systematic Literature Review (SLR) is a means of identifying, evaluating, and interpreting all available research relevant to a particular question (PETTICREW; ROBERTS, 2008). This Chapter reports on the SLR carried out to update and extend the preceding investigation of Machado et al. (2014), which analyzed an initial set of two hundred seventy-six studies, published between the years 1998 and 2013. In this study, we analyzed primary studies published between the years 2014 to 2016, so we could have a current view on the SPL testing field. Besides, a new set of research questions was defined, so we could focus our investigation on topics not addressed in the preceding study. To carry out this work, we followed the procedures of (KITCHENHAM, 2004).

The study focused on SPL Tests and has the following objectives: investigate test strategies, test metrics, synthesize available evidence and identify gaps between required techniques and existing approaches available in the literature (MACHADO et al., 2014)

3.1.1 Research questions

We followed the PICOC - Population, Intervention, Comparison, Outcome and Context - criteria (WOHLIN et al., 2012) to define the research questions addressed in this study, as Figure 3.1 shows. They are described next:

Table 3.1 PICOC

Criterion	Description
Population	Published studies in the SPL testing field
Intervention	Studies with empirical evidence
Comparison	Not applicable to this investigation
Outcomes	Evidence on metrics, techniques, tools and strategies suited to SPL testing
Context	Empirically assessed studies in the SPL testing field

RQ1. What SPL testing strategies are available to handle the selection of products to test?

RQ2. What SPL testing strategies are available to deal with the test of end-product functionalities?

RQ3. What are the commonly used test metrics in SPL testing practice?

RQ4. How does SPL practice handle variability testing?

RQ5. What are the available SPL testing tools?

RQ1 and RQ2 were incorporated from Machado et al. (2014). The former was aimed at leveraging primary studies dealing with the selection of product instances for testing. The later was defined to leverage the test strategies for the test of product functionality. The remainder are novel questions, especially designed for this update study. RQ3 was defined to identify which test metrics are applied in the SPL testing practice. RQ4 aimed to leverage testing techniques available to test software variability, and RQ5 was defined to identify the SPL testing tools used in SPL projects, and also to investigate whether support information was available.

3.2 DATA COLLECTION

3.2.1 Identification of relevant literature

The procedure to collect and select the most relevant studies to be used for data extraction was developed in two distinct phases, as Figure 3.1 shows.

In Phase 1, we conducted an automated search for studies published between the years 2014 and 2016. The collection phase consisted of a screening process aimed to exclude studies that are not relevant to answer the research questions. Phase 2 consisted of a manual search of the relevant studies in conference proceedings. In addition, snowballing gave through all the selected papers from both automated and manual phases (WOHLIN, 2014). After reading the studies in each automated and manual phase, a total of 33 studies were selected.

In both phases the inclusion and exclusion criteria were used to select the studies during the search. The inclusion and exclusion criteria were applied in the first and

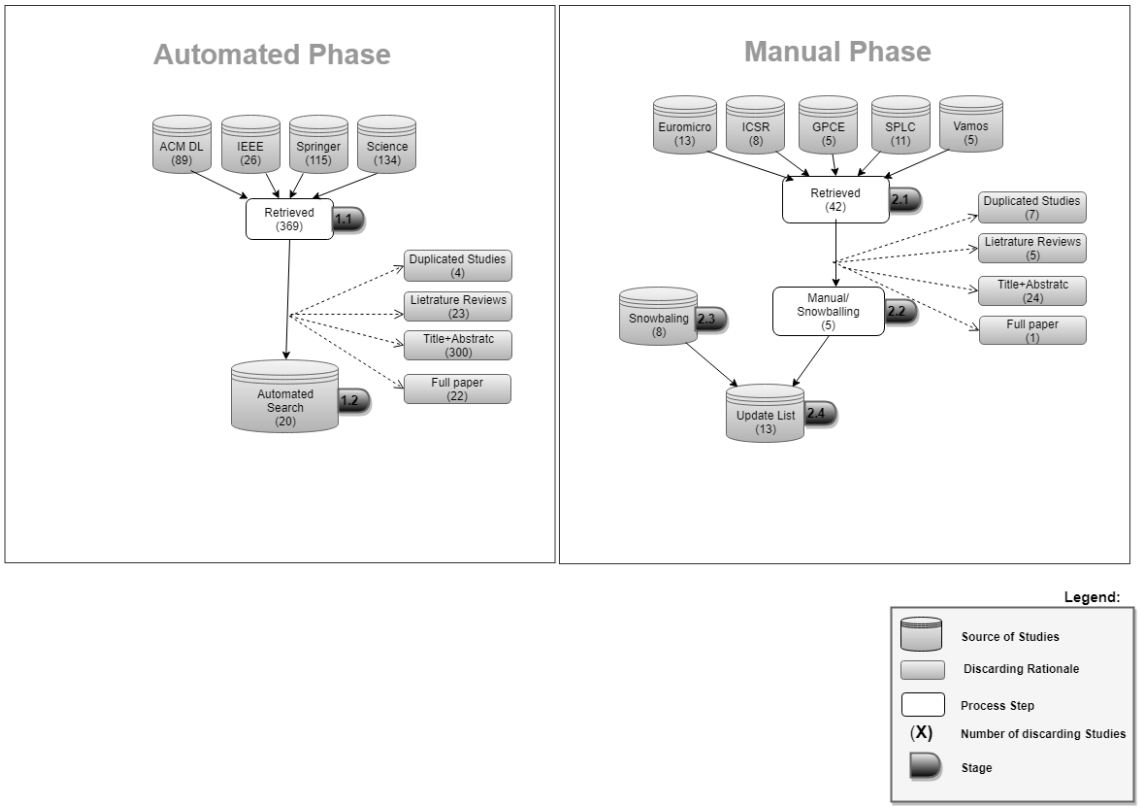

Figure 3.1 Stages of the selection process (adapted from Machado et al. (2014)).

second filter in the study titles and then in the abstracts. Next, we provide more detailed information about each phase.

3.2.1.1 Phase 1: Automated Search

In the first phase, we performed an automated search in the search engines of the following digital libraries: *ScienceDirect, ACM Digital Library, IEEE Xplore*, and *SpringerLink*. These are deemed as the most important ones in the field of Software Engineering, as they together index all relevant journals and conferences in the field.

In order to perform the search, we defined a set of keywords, reflecting the established goals and research questions of this study. Therefore, the following terms were applied: *"Test Technique"*, *"Testing in Software Product Line"*, *"testing strategies in SPL"*, *"metrics"*, *" variability"*, *"Tools"*. For each digital library, a string was created to meet their syntax requirements and capabilities. Table 3.2 shows all the search strings used.

From the automated search, we retrieved a set of 369 primary studies. We eliminated the duplicated studies, i.e., retrieved by more than one search engine. We also excluded retrieved SLR.

The next task consisted of reading the primary studies' titles and abstracts and applying the inclusion and exclusion criteria, to filter out uninteresting studies. Table 3.3 shows the set of inclusion and exclusion criteria used in this SLR. On the basis of such

Table 3.2 Search strings used in the digital libraries.

Databases	Search String
IEEE	(test OR testing) AND ((“software product line” OR “software product family” OR “configurable systems” OR “domain engineering” OR “variability”) AND (metric OR measure))
SPRINGER	(software AND product AND line AND test AND testing AND variability AND metric AND measuring AND family)
ACM DL	(test OR testing) AND ((“software product line” OR “software product family” OR “configurable systems” OR “domain engineering” OR “variability”) AND (metric OR measure))
SCIENCEDIRECT	((“test” AND “software product line” AND “metric”)) or (“testing” AND “software product family” AND “configurable systems” AND “domain engineering” AND “variability” AND “measure”)

Table 3.3 Inclusion and exclusion criteria.

Inclusion criteria	Exclusion criteria
IC1. All publications addressing testing strategies, metrics or tool of SPL	EC1. The publication is gray literature - technical report, short paper, article without empirical evidence, magazines, and extended abstracts
IC2. All publications in English.	EC2. The publication is a secondary study
IC3. Period: 2014- 2016	EC3. The publication is an opinion paper
IC4. In case of extended studies, only the most complete would be selected	EC4. The study does not address SPL
	EC5. Excluded after re-reading of the title and abstract - Papers are not related to testing in SPL.

analysis, another 300 studies were excluded, as they were not considered as relevant to our research.

The remaining studies were subject to full-text reading, so actual data could be extracted from them. After carefully reading every study, 22 of them were excluded because they did not address either strategies for SPL or techniques and metrics.

At the end of the automated search (Phase 1), a final pool of **twenty** studies was included.

3.2.1.2 Phase 2: Manual search

In the second phase, we conducted the manual search for primary studies. To accomplish this task, we only considered the conferences relevant to the field of SPL, as follows: *International Systems and Software Product Line Conference (SPLC), International Conference on Software Reuse (ICSR), International Workshop on Variability Modelling of Software-intensive Systems (VaMoS), International Conference on Genera-*

tive Programming: Concepts & Experience (GPCE), and Euromicro Conference on Software Engineering and Advanced Applications (SEAA).

The search retrieved 42 studies after reading titles and abstracts. Seven of them were excluded as being duplicate studies. Another five studies were secondary studies (i.e., SLR), as such they were excluded as well.

After applying the inclusion and exclusion criteria, another 22 studies were excluded, as they did not address any topic related to this investigation.

Alike in the previous phase, we also carried out a full-text reading in the remainder set of studies, aiming to get actual data, important to answer the research questions. After reading the complete papers, **five** studies were considered as being relevant to our investigation.

Next, we carried out a snowballing, in an attempt to include studies from the references of the primary studies selected up to this point. In this task, we retrieved another **eight** relevant studies. Therefore, there were **thirteen** studies included in final pool from the manual search.

After merging the results from Phases 1 and 2, we had a set of **thirty-three** studies included in this updated and extended SLR.

3.2.2 Data extraction

After selecting the primary studies, we performed the data extraction, a mandatory task to leverage data necessary to answer the research questions. From each study, we extracted the following data:

- Study title, venue, and publication year;
- Testing strategies: selection and prioritization, and design;
- Test metrics and tools;
- Empirical assessment methods employed in the study;
- Domains and platforms in which the studies have been applied;
- Level of evidence, classified and evaluated according to the hierarchy of evidences proposed by (KITCHENHAM, 2004): (i) no evidence; (ii) obtained from demonstration or working out toy examples; (iii) obtained from expert opinions or observations; (iv) obtained from academic studies; (v) obtained from industrial studies; and (vi) obtained from industrial practice.

After extracting all relevant data from each primary study, an analysis was carried out so that we could group the main test strategies, metrics, and tools.

3.3 RESULTS OF THE SLR

In this section, we used the data extracted from the primary studies to answer our research questions. We provided an overview of the selected studies and their relation to their publication sites. The studies are listed in Table A.1 in Appendix A.

3.3.1 Characteristics of the studies

Table 3.4 lists the venues and the number of publications where the studies were published.

Table 3.4 Study distribution per publication sources.

Source	Count
Conferences	
International Software Product Line Conference - SPLC	7
Workshop on Variability Modelling of Software-intensive Systems - VAMOS	3
International Conference on Generative Programming: Concepts and Experience - GPCE	2
International Conference on Software Reuse - ICSR	2
International Conference on Software Testing, Verification and Validation - ICST	2
International Workshop on Feature-Oriented Software Development - FOSD	2
Brazilian Symposium on Software Engineering - SBES	1
Congress on Evolutionary Computation - CEC	1
Conference on Genetic and Evolutionary Computation - GECOO	1
International Conference on Computing, Control, Networking, Electronics and Embedded Systems Engineering - ICCNEEE	1
International Workshop on Search-Based Software Testing - SBST	1
Journals	
Journal of Systems and Software - JSS	3
Software and Systems Modeling - SoSyM	2
IEEE Transactions on Software Engineering - TSE	1
Information and Software Technology -IST	1
Journal Applied Software Computing - ASC	1
Journal Automated Software Engineering - ASE	1
Journal of Logical and Algebraic Methods in Programming - JLAMP	1

Through the analysis of extracted data, we grouped the studies according to their central purpose. Figure 3.2 shows the number of studies addressing each SPL testing strategy, namely *(i)* strategies handling the selection of product instances to test (i.e., this **first interest** is focused on handling *core asset testing*, concept earlier discussed in Section 2.4), and *(ii)* strategies handling the actual product test (i.e., this **second interest** is focused on handling *product development testing*). Table 3.5 shows the studies addressing each SPL testing interest. It is worth to mention that we gathered some studies which address both interests. Besides, Figure 3.2 also shows a combination of the studies retrieved in Machado et al. (2014) - period from 2003 to 2013, and the set of studies retrieved in this current work - period from 2014 to 2016.

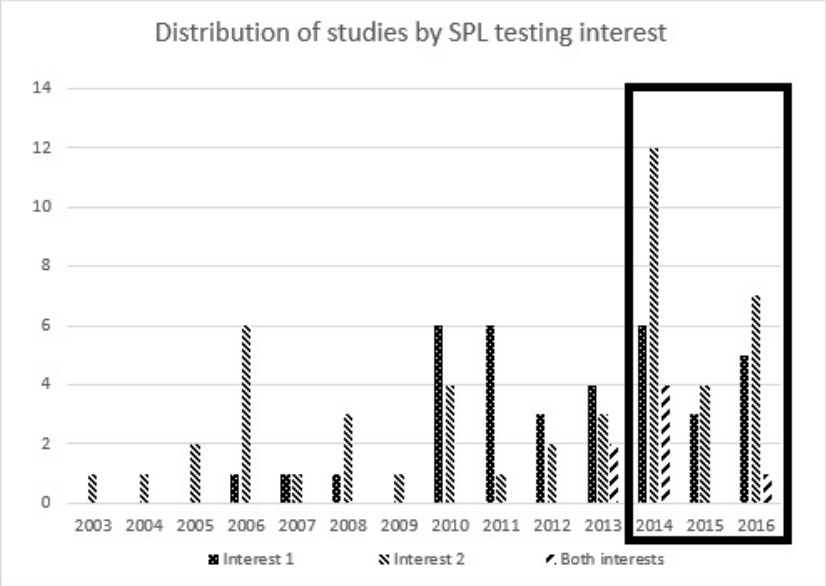

Figure 3.2 Distribution of studies by SPL testing interest and publication year

Table 3.5 Selected studies vs. SPL testing interest addressed.

Interest	Studies	Count
1	[P1], [P2], [P3], [P6], [P7], [P8], [P10], [P11], [P12], [P15], [P17], [P19], [P22], [P23], [P24], [P27]	16
2	[P2], [P4], [P5], [P6], [P7], [P9], [P10], [P13], [P14], [P6], [P18], [P20], [P21], [P22], [P25], [P26], [P28], [P29], [P30], [P31], [P32], [P33]	22
Both	[P2], [P6], [P7], [P10], [P22]	5

3.3.2 Strategies to handle the selection of products to test (RQ1)

Testing an SPL could become impractical due to the huge number of possible product configurations. The primary challenge in SPL testing is therefore to reduce the set of possibilities to a reasonable and representative set of product configurations, thus limiting the effort required to test each selected configuration.

One affordable strategy to cope with such a challenge is to employ CIT techniques to reduce the number of products to handle. This is a low-cost engineering strategy for SPL testing (MACHADO et al., 2014).

In the search for primary studies, we found several ones that are based on CIT techniques, in which the main objective is to select a group of products where feature interaction failures can occur. A feature interaction is a functional behavior where two or more features are selected and enabled for the derivation of an SPL and act unexpectedly, but when they are enabled separately, work in isolation from other behavior (OLIMPIEW, 2008; APEL; KASTNER, 2009). It is possible to detect faults that are triggered by

feature interaction erroneously between the number of features, by employing CIT techniques. We next summarize the main primary studies addressing this first SPL testing interest.

P19 proposed an incremental approach to product sampling for pairwise interaction testing called IncLing Incremental sampLing. They applied CIT to reduce the test effort of the SPL by selecting a minimum but sufficient subset of products, thus restricting the number of products to be tested.

P05 used similarity-based CIT as a surrogate metric for t-wise coverage, that is, if the size of test suites is unfeasible due to technological problems or budget constraints.

P10 used the similarity technique to compare two configurations. They claim that model-based tests have greater failure detection power than similar ones. Due to the application of the two configurations, they used the direct distance metric, that is, a set of distance metrics to evaluate their degree of similarity.

In P10, the main focus of SPL testing has been on the use of pairwise testing, where the coverage takes place through combinations of features, aiming to select a set of products.

P11, P12, P19, and P23 presented approaches in which the pairwise technique was used as an heuristic for the design, selection and coverage of test cases.

The pairwise testing technique aims to select a set of products to test where possible combinations of two features are covered by at least one selected product, i.e., pairwise has focused on obtaining full coverage of all combinations of features with the minimum number of products to be tested.

In order to improve the coverage of the test, only two out of the selected studies did not present any algorithmic implementation to generate all possible connections. According to the selected studies, a range of tools were used, but only 26% (P2, P10, P11, P17,21, and P28) of the selected studies used the SPLCat tool to generate all possible pairs of features Feature model.

Sanchez, Segura and Ruiz-Cortes (2014) explains that the SPLCat tool uses an implicit prioritization criterion, where products that cover the most discovered pairs of features, get first in the list.

In this first interest, we also address software quality metrics. Four out of the primary studies addressing the first interest (P1, P2, P22, and P27) highlighted the use of metrics. The most commonly used metrics are: *Average Percentage of Faults Detected (APFD), Fault Detection Capability (FDC)*, and *Test Suite Fault Detection Capability (TSFDC).*

3.3.3 Strategies to handle the test of end-product functionalities (RQ2)

This second research question is mostly interested in leveraging the testing practices handling problems regarding the actual testing of functionalities, a common activity of *product development.*

For each feature, there must be (a set of) test cases present in the test suite to validate whether the characteristic has been properly implemented. There are some characteristics intrinsic to this SPL testing interest a technique should address, as follows: *(i) Variability* - SPL test assets are explicitly designed and modeled making their variation points explicit, *(ii) Test asset reuse* - Test assets from domain engineering may be sys-

tematically reused in application engineering, as assets from a product instance may serve as input for a next instance, *(iii) Test Automation* - employing an automated strategy for generating SPL test assets leads to significant effort reduction, *(iv) SPL process* - tests can be performed in core asset development, in product development, or in both processes. Table 3.6 lists the studies addressing each of these characteristics.

Table 3.6 Issues addressed by each primary study.

Study	Variability		Asset reuse				Automated gen.		
	TC	TS	TC	TS	TR	TD	TC	TCS	TI
P1	·	·	·	·	·	·	●	·	●
P2	●	·	●	·	·	·	·	·	·
P3	●	·	·	·	·	·	●	·	·
P4	●	·	·	·	·	·	·	·	·
P5	·	·	·	·	·	·	·	·	·
P6	●	·	●	·	·	·	●	·	·
P7	●	·	·	·	·	·	●	·	·
P8	⊙	·	·	●	●	·	·	·	·
P9	●	·	·	·	·	·	●	·	·
P10	·	●	·	·	·	·	·	●	·
P11	·	●	·	·	·	·	·	·	·
P12	·	·	·	·	●	●	·	·	·
P13	·	·	●	·	·	·	·	·	·
P14	·	·	●	·	●	·	·	·	·
P15	·	●	·	·	·	·	·	·	·
P16	·	●	·	·	·	·	·	·	·
P17	·	●	·	·	·	·	·	·	●
P18	·	·	●	·	●	·	·	·	·
P19	·	·	·	·	·	·	·	●	·
P20	·	·	·	·	·	·	·	·	·
P21	·	●	·	·	·	⊙	·	·	·
P22	●	·	·	·	·	·	·	·	·
P23	·	●	·	·	·	·	·	·	·
P24	·	·	·	·	·	·	·	●	·
P25	·	·	·	·	·	·	●	·	●
P26	·	●	·	·	·	·	·	·	·
P27	●	●	·	·	·	·	·	·	·
P28	·	●	·	·	·	·	●	·	·
P29	·	·	·	·	·	·	●	·	·
P30	·	●	·	·	·	·	·	·	⊙
P31	●	·	●	·	·	·	●	·	·
P32	●	·	·	·	·	·	·	·	·
P33	●	·	·	·	·	·	·	●	·

Legend: [●] Characteristic clearly addressed by the study, [⊙] The study encourages the use of such characteristic, but do not provide any implementation (e.g., it states an external tool is used, but no detail is provided), [·] Characteristic not mentioned in the study, [TC] Test case, [TI] Test input, [TD] Test data, [TCS] Test case selection, [TR] Test result, [DE] Domain Engineering, [AE] Application Engineering, [TS] test Scenarios.

We analyzed how the studies address the automated generation of test cases (TC), test cases selection (TCS) and test inputs (TI). In 10 out of the 16 studies, the authors

presented a description of how to perform the automated tests by offering tool support to deal with the mentioned characteristics.

Moreover, we analyzed the level of evidence gathered from each study, as Table 3.7 shows. They were mostly evaluated through academic studies (Lev4) with 63% of the total, followed by the evidence obtained from industrial studies (Lev5) with 31%, and one study obtained evidence from industrial practice (Lev6) and another from demonstrations or toy samples (Lev2), representing around 3% of the studies each.

Table 3.7 Evidence level addressed in each primary study.

Study	Lev1	Lev2	Lev3	Lev4	Lev5	Lev6
P1				•		
P2				•		
P3				•		
P4				•		
P5					•	
P6				•		
P7					•	
P8				•		
P9						
P10				•		
P11				•		
P12					•	
P13					•	
P14					•	
P15		•				
P16				•		
P17				•		
P18					•	
P19				•		
P20					•	
P21				•		
P22						•
P23					•	
P24				•		
P25				•		
P26					•	
P27					•	
P28				•		
P29				•		
P30				•		
P31				•		
P32				•		
P33				•		

3.3.4 Testing metrics suitable for SPL testing (RQ3)

When we deal with software, we always keep in mind that the software complies with the requirements of the customer. In this context, according to Pressman (2005), the quality of software is a compliance of functional requirements requested by the customer

and documented according to development standards.

Metrics are a quantitative measure of the degree to which the system, component, or process has a given attribute (IEEE, 2008). In other words, metrics are the means that we can measure a system, giving it due confidence in the product. There are several software quality metrics.

By analyzing gathered data, we could observe that the studies P1, P2, P14, P16, P18, P21, P22, P27, P28, and P33 (30% out of the total) used the following metrics: *Mean Number of Failures Fetected, Fault Detection Capability (FDC) and Average Percentage of Faults Detected (APFD)*. These metrics are related to the quality of the test, to evaluate how quickly the failures are detected during testing.

Another set of metrics, although not as commonly used as the aforementioned list, could also be observed. In Lopez-Herrejon et al. (2014a), the authors employed the metric Hypervolume (HV) and Generational Distance, which are quality indicators used in the multi-objective community to compare the approximate Pareto fronts of various algorithms. Among other authors who have used the Distance Metrics, Distance Learning, Euclidean Distance, Jaccard, Dice, Anti-dice distance, where they represent 19% of the selected primary studies (P3, P10, P11, P24, P25 and P27). These measure the diversity between two products and the authors used distance metrics to compare the similarity of the sampling set.

Devine et al. (2016) used the change and static code metrics. Change metrics were used for classification of fault-proneness at a file level. In particular, for each file we extracted the same set of seventeen change metrics, and Static code metrics that capture information pertaining to the source code. They range from simple metrics, such as Lines of Code (LOC), to metrics that measure structural intricacy, such as Cyclomatic Complexity (CC).

3.3.5 Variability testing techniques (RQ4)

Of the 33 selected studies, 66% (22) of the studies present several types of testing techniques to handle variability. The studies were categorized in three dimensions: Design, Selection/Prioritization or both. The test Selection/Prioritization dimension were considered in 50% of the identified studies, followed by 32% of the design tests, and the remaining 18% covered both dimensions.

The studies P2, P4, P10, P11, P16, P17, P21, P22, P28, P30 and P31 present prioritization testing techniques. The studies P4, P10, P11, P28, and P30 used the Pairwise test techniques. P4 presented a parallel genetic algorithm to generate a prioritized pairwise test sets for SPL called *Parallel Priorized Genetic Solver*. Authors compared this with the Covering Array Generation Algorithm for SPL prioritized-ICPL Algorithms. The prioritized ICPL does not calculate coverage matrices with full coverage, it covers only the `n`-wise combinations among the features that are present. In this context, they used Prioritized Pairwise Covering Arrays in SPL.

The techniques proposed by P10 are effective and scalable search-based approaches capable of generating product configurations for large-sized SPLs. Authors used prioritization algorithms for any set of product configurations.

The techniques employ a similarity heuristic evaluating empirical studies through a comparison with last generation tools and focusing on generating the configuration of the product as in the aspects of prioritization. Prioritization techniques based on interaction coverage that is commonly used in CIT studies and based on `t`-wise interaction coverage. In the comparison of prioritization approaches, the area under curve is evaluated and represents the weighted average of the percentage of failures detected over the life of the test set.

P21 developed three fault injection strategies and identified differences in performance between algorithms by improving search results. These techniques that drive knowledge of the problem domain, in which the authors measured the impact of performance.

The studies P3, P32, and P11 used techniques such as mutation operators, multi-objective evolutionary techniques and fault-based testing technique.

The authors P32 presented a mutation system that evaluates the quality of the SPL tests through the ability to detect failures. The authors used the test techniques: fault-based testing technique and mutation analysis. The model-based mutation operators are adapted from automated and executed model testing cases and automated mutation analysis.

The slicing technique is an analysis technique to automate the retest decision. P6 propose an automated analysis of the impact of the change based on the incremental slice of the model for the incremental test in SPL. Authors used the techniques of Change Impact Analysis that captures all changes between the test models, and Retest Coverage Criterion which represents a pi cut criterion for which there is a slice regression, while the Retest Test-Case Selection and Generation Validate the changes do not unintentionally influence the behavior already tested.

The selected studies P3, P15, P23, P26, P27, and P30 present an algorithm-oriented approaches. P15 propose and evaluate a hyper-heuristic approach (HH) to derive a set of Feature Models product tests. Hyper-heuristics is a methodology that automates the design and configuration of algorithms, whose purpose is to solve computationally difficult problems. P23 propose a method and a tool that exploits and helps engineers to validate and manage in an automated, scalable, and efficient way. The approach is based on scheduling constraints which allow engineers to increase the effectiveness of configuration tests. This technique covers the modeling of software variability using a feature model and specifying the dependencies between them.

P26 used two Block-based Algorithm and Variable-based Algorithm to optimize the high coverage of the common code base in terms of preprocessed C ++ configurations with a limited set of actual configurations selected for testing. The two algorithms automatically detect the settings by maximizing preprocessor-level code coverage in C++ based projects.

P33 evaluate black box testing techniques in variability in the Drupal framework, using CIT. To perform the test, the authors used the combinatorial algorithm as an example ICPL that is used to generate an pairwise suite for the Drupal feature model, using different criteria based on functional and nonfunctional information to derive a request from test cases that allows to detect failures.

3.3.6 SPL Testing tools (RQ5)

Aiming at greater agility in the activities of the test process, we present some tools to support SPL testing. Test support tools can contribute to reduce time, and improve productivity, reliability, and especially quality for each stage of the test life cycle.

Of the 33 primary studies selected, 68% are proposing as tools, while only 16% are proposing algorithms and 16% of the studies are proposing both.

With the extraction of the data, we observed that the SPLCAT tool was used by 18% of the selected studies - P2, P10, P11, P17, 21, and P28.

P2 used the SPLCat tool to select the test pairs and implement a fault generator, the authors also used the Average percentage of faults detected (APFD) metric to aid in the speed of fault detection. P2 points out that "SPLCAT uses an implicit prioritization criterion by first listing the products that cover the most discovered pairs of features".

P5 calculated and compared the number of failures that were lost by the products generated using CASA covering array, whereas P21 measured the metric values Average percentage of faults detected (APFD) of both prioritized suites generated by a non dominated sorting genetic algorithm II (NSGA-II) adaptation and the initial set of pairs generated by the CASA algorithm in each execution.

3.4 ANALYSIS AND DISCUSSION

In the preceding work (MACHADO et al., 2014), the selected studies were separated into two large groups, which were referred to as first interest and second interest. For approaches in both interests to be concerned with minimizing the test effort, there is little evidence on the subject of obtaining higher rates of defect detection.

Machado et al. (2014) state that each interest has a specific purpose in the SPL test task, knowing common faults can be useful, and can guide an engineer to better identify the faults that are most likely to occur. In addition to the above-mentioned observation, we compared and discussed the main open-ended questions we found by each of the interests and our findings. This work was carried out from two feature-based (first interest) or product-based (second interest) perspectives. The studies selected by Machado et al. (2014) in the period from 2003 to 2013 remained at the same level in relation to this work carried out in the period from 2014 to 2016, as presented in section 3.3.1 in Figure 3.2.

- *First SPL testing interest*

The main problem of the first interest in SPL testing is to reduce the set of possibilities for a reasonable and representative set of product configurations, thus limiting the effort required to test each selected configuration (LOCHAU et al., 2012).

In their study, Machado et al. (2014) show the current techniques are manipulated the combinatorial problem and they can be categorized as sampling and exhaustive exploration techniques. While the former emphasizes the selection of configurations, as well as pairwise coverage, to reduce the combinatorial space involved, the latter intends to consider all possible configurations, forming a flexible and scalable alternative to current techniques in the behavioral aspects of SPL.

Several approaches were found in our work. The ones that stood out most were based on the CIT, which the main objective is to select a group of products where the interaction failures can occur. To address the problem, an incremental approach to product sampling for paired interaction testing called IncLing Incremental sampLing was proposed by Hajjaji et al. (2016a). Authors performed a combinatorial test to reduce the SPL test effort by restricting the number of products to be tested for a minimal subset of products, while the authors Fischer et al. (2016) analyze how the faults are distributed between resources and how effective it is for detect failures and how the similarity compares to the classic coverage `t`-wise.

In the studies selected by Machado et al. (2014), we observed the use of several optimization algorithms and the applicability of prioritization techniques from test cases to test and, finally, to evaluate the quality of the product line test of through the ability to detect faults as quickly as possible. In this sense, one of the concerns is which strategy to use to test all feature interactions in variability. It is necessary to test and analyze all possible combinations of existing features, even in reasonable size variability models.

In P10, authors applied the strategy to evaluate real FM and randomly generated. They investigated the likelihood of encountering an interaction failure using a randomized, search-based approach that works with constraints capable of generating and prioritizing product configurations. When there is a problem in generating SPL configuration, it can be solved by a CIT tool if it handles constraints. CIT targets sample configurations to reduce the size of test suites (HENARD et al., 2014).

Another highlight was the generation of sets of test cases using tools with combinatorial algorithms (P11, P12, P19 and P23). Wang, Ali and Gotlieb (2015) emphasizes minimizing testing for product lines by reducing the total number of test cases to be performed, thereby improving test efficiency. In that sense, they investigate the fitness function with three Genetic Algorithms based on weight and seven multi-objective search algorithms using an industrial case study and 500 artificial problems. While other authors, such as Devroey, Perrouin and Schobbens (2014), focus only on the creation of algorithms to construct a set of tests that satisfies the criterion of coverage of all states at the SPL level with focus on the behavioral aspect using failure injection.

- *S*econd SPL testing interest

The second interest is composed by most of the approaches that deal with the specification of variability in test assets. According to the variable scenarios, they can include specifications for the entire set of possible products, depending on how the variant features are instantiated. In this instance, not all possible test cases can be derived, but are derived from the SPL test specification. Through a specific product the test cases will actually be derived by instantiating the tags in each SPL use case.

As the number of products increases, the number of test cases would also increase. In this sense, the search for the solution to minimize the test suites for a specific product is essential. In addition, an efficient testing process should systematically explore the reuse of test artifacts among products under test (MACHADO et al., 2014).

The analysis of uniformity and variability is the domain engineering, which the process occurs and defines the reusable test assets. In this study it was identified that 31% of

the studies selected for the second interest. Most of the analyzed approaches deal with the specification of variability in the test assets, and explore the capability of variability models to propose strategies to define reusable tests.

Lochau et al. (2014b) uses delta-oriented architectural test component, which enables the generation and reuse of test artifacts between different system variations, while Lochau et al. (2014b) uses an efficient integration test approach for SPL based on delta modeling, focusing on prioritizing test cases by highlighting the differences between variants and identifying the altered parts, which can be retested.

To analyze the computation, Lackner and Schmidt (2014) use model-based mutation operators and adapt them to test cases, automated modeling and execution of automated mutation testing and analysis. The authors developed several mutation operators, and use state machines from the Unified Modeling Language (UML) to reduce costs and efforts in test cases. They evaluated the mutation techniques against the tests generated from the specifications by transcribing the conclusions about the effectiveness of operators mutations.

3.4.1 Threats to Validity

We observed some threats that might affect the validity of our this empirical study. We next discuss them.

Internal validity. The first threat refers to the subjective decisions we take to either include or not a primary. To mitigate this threat, the study was carried out with two researchers and a specialist from the area to consolidate the differences in a collaborative way and to evaluate the primary studies following a pre-defined protocol.

Next, when considering data extraction from primary studies, some of the studies did not provide a clear description or objectives and results. To mitigate the threat, it was necessary to incorporate the most complete primary studies possible to increase the reliability of the conclusions.

External validity. Threats to external validity are the ability to propagate the achieved results outside the observed environment. In this SLR, we selected representative results of the research questions, and selected only four digital libraries according to the Table 3.2. Although we have considered the most important venues, whose are most likely to find SPL-related studies, such a limited size may hinder generalizations of findings.

Construct validity. The threat to construct validation refers to the relationship between theory and observation. We attempted to leverage all relevant studies in the topic under investigation, combining automated and manual researches, to increase coverage.

Conclusion validity. The validity threat to the conclusion describes a relationship between treatment and outcomes, that is, it addresses the issues that affect our ability to draw correct conclusions about an experiment and whether the operations of a study can be repeated with the same results. In this study, we defined search strings and procedures so that other researchers could replicate it directly and objectively and they could find the same set of primary studies.

3.4.2 Related work

In 2004, Tevanlinna, Taina and Kauppinen (2004) published a research on SPL testing. They discussed established practices and challenges that surrounds the SPL testing field. The focus of the discussion was on the methods developed to be applied to test product families, ignoring particular features of a particular method. The paper served for a long time as a good road map for researchers to investigate the field.

Neto et al. (2011) carried out a systematic mapping study to analyze important aspects that should be considered when adopting test tools. They evaluated 33 primary studies from the period 1999 to 2011. Of the 33 studies, 24 described single-system test tools and the other 9 SPL test tools described. Tools are usually developed to support a specific testing level, under the justification that there are no tools supporting the whole software testing process. We identified the possibility to use single system testing tools such as JUnitMX and CodeGenie to support the SPL testing process.

3.5 SUMMARY

Machado et al. (2014) presented a SLR on SPL testing strategies, covering the period from the year 2008 to 2013. In this current investigation, we elaborated on such a study, and expanded the year range, to include studies published up to the year 2016, as well as included some more research questions, to focus on aspects not covered in previously published literature reviews in this field.

This chapter reported an extended and updated review, based on this former study. To assist us in our research questions in the next chapter, the result of the first interest presents the understand how products are selected from a very large set of possible asset testing products and how each selected product is tested. Of the 33 studies selected, 16 provided material for the discussion of the first research question, while 22 articles described strategies that matching the second interest.

Chapter 4

MERCI - A METHOD TO EVALUATE CIT TOOLS FOR SPL ENGINEERING

Testing SPL projects is a rather complex activity, due to the presence of variability in its engineering process, which increases the number of product configurations to test. The underlying idea to make testing feasible in SPL engineering is to select a small but representative subset of products to test, by employing techniques such as CIT.

CIT can be considered as an affordable strategy to reduce the overall testing effort by selecting a small but representative set of products to test (LOPEZ-HERREJON et al., 2015; YILMAZ et al., 2014). CIT techniques have been used in several domains and there is a variety of tools to support the testing process (YILMAZ et al., 2014). They are mainly concerned about providing feasible test configuration selection or employing test prioritization approaches. However, selecting either a CIT technique or support tool suitable to SPL engineering is not straightforward.

In order to facilitate the process of selecting support tools which both implement CIT techniques and may be suitable for SPL engineering, we designed **MERCI**, a method to evaluate the adequacy of CIT techniques to test SPL projects. This Chapter discusses the proposed method in details.

4.1 THE MERCI METHOD

In this section, we describe our proposed method to evaluate the adequacy of CIT tools for SPL testing. Figure 4.1 shows an overview of the method, comprising the set of steps to take in order to assess a given CIT technique. The steps are discussed next.

1. **Generate a Reduced Set of Product Configurations**

   In this first step, the goal is to generate a reduced set of product configurations. Firstly, we need to select some features to generate the reduced set.

   To achieve this goal, we use a Feature Models (FM) to calculate a coverage matrix, using a pairwise technique, which will serve as input and generate the possible combinations of features that could satisfy the n-wise coverage.

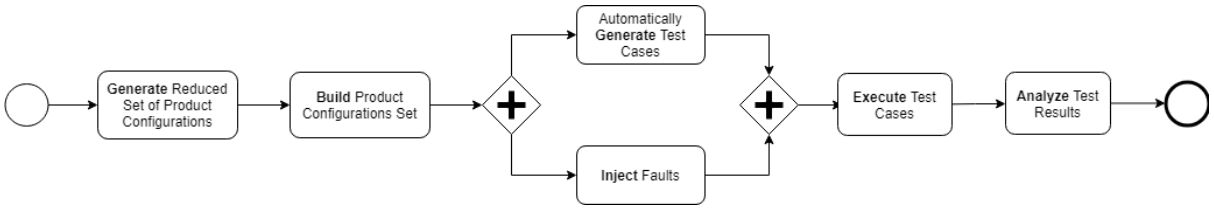

Figure 4.1 Overview of the MERCI.

The presence of variability may add significant challenges to the development of SPL projects. Through a FM, as Figure 4.2 shows, it is possible to generate an enormous amount of products. For example, the BankAccount system, a small SPL project used in this investigation, could generate around 40 products. Whether we considered larger projects, testing all combinations of features could be an exhausting activity. Therefore, the idea is to cover all combinations of parameters, but without necessarily testing all of them.

To generate the reduced set of product configurations, that are representative of all combinations, we select a set of features and their respective constraints, be they mandatory, optional or alternative, that will serve as input for each tool.

The features input in the CIT tools is as follows: each feature is launched as a parameter and each feature has its hierarchical value. When a feature is mandatory, it receives only a true value, when the feature is optional, it may receive either true or false values, and then the constraints between the features are inserted.

After the input of the features in the CIT tools, the tool is executed, and the expected result is the configuration of the reduced products set, which is capable of satisfying the all n-wise coverage.

2. **Build Product Configurations Set**

   After generating the Reduced Set of Product Configurations, this would serve as the input for building the reduced products, that is, to generate the products with their source code. This is enabled with the support of an automated tool to generate product instances, such as the FeatureIDE (THUM et al., 2014), a toolset supporting the Feature-Oriented Software Development (FOSD) paradigm. In the FeatureIDE, it is possible to select the *Generate Reduced Set of Product Configurations from a feature model* option, which makes it possible to automatically generate distinct products.

3. **Automatically Generate Test Cases**

   The goal of this step is to generate the automated test cases. The source code of the reduced product set will be used as input to this phase. The automatic generation of test cases could result in an increased test coverage.

Such a task depends upon tool support. The Randoop tool[1] is a good choice, for the valuable results it may yield. It generates tests for Java code in JUnit, and optimizes all test suites to satisfy a code coverage criterion. The expected results of this phase are the automated test cases.

The Randoop tool generates two types of unit tests: Error-revealing tests and Regression tests. Error-revealing tests aim to fail at the time of execution, indicating a possible error in one or more test classes, and regression tests are tests that pass when executed. Randoop selects a method to call. The method selected will be the last method call in the test. Randoop chooses arguments for the method call, among the values that were calculated by previous tests. No preconditions are thrown for each method, so by default it assumes that any exception thrown by a method and is a correct behavior in response to values that were passed (ABDELGHANY MICHAEL ERNST, 2010). For the execution of the experiment, the regression tests were used.

4. **Inject Faults**

In order to evaluate the effectiveness of the generated test cases, we need to seed some faults in the source code. Fault injection could be supported by the mutation testing technique (JIA; HARMAN, 2011). It is a technique to improve the coverage of a test, in which faults are inserted into the source code to test software paths.

Likewise the preceding steps, it is necessary to count on automated tool support, in order to reach better coverage. There are several automated tools to inject faults in Java-based programs. PIT [2] provides test coverage for Java and reports a set of metrics. Faults are inserted into the source code of the products through automated PIT tools.

Next, it is necessary to introduce random mutants that lead some test cases to fail. The quality of the tests can be evaluated by analyzing the percentage of dead mutants, ensuring that such mutations are contained only in variants in certain combinations of features (COLES et al., 2016).

PIT works by randomly inserting mutants in an automated form and the tests occur on the units modified in the project code. As a result, we get a set of mutated source code classes.

According to Coles et al. (2016), mutants aim to measure how good the tests are, observing and comparing the runtime behavior of non-mutated and mutated programs. Mutants are artificially introduced, measuring the suitability of the test, operating quickly and acting directly on the bytecode by optimizing the executed mutants, and supporting a small number of mutation operators.

[1] Randoop is a unit test generator for Java. It automatically creates unit tests for Java classes, in JUnit format. Available at ⟨https://randoop.github.io/randoop/⟩.

[2] PIT is a state of the art mutation testing system, providing gold standard test coverage for Java and the jvm. Available at ⟨http://pitest.org/⟩.

This is aimed to limit the number of mutants and the execution time (AMMANN, 2015). A list of mutants handled by the PIT tool is available at (COLES et al., 2016).

5. **Execute Test Cases**

   The application of the MERCI method requires the execution of the tests. For this implementation, we must select a set of variants to test. The test cases were included in the Eclipse projects.

   Then, we run the automated tests using the PIT tool. PIT automatically inserted the mutants into the .class files.

   Each of these mutated versions of the selected variants is then tested and the test results compared against each other.

   Therefore, the assertions in the unit test may not be valid in certain variants, and we would mistakenly assume that the mutation was detected.

   The result for our MERCI Method is a set of mutants detected for a given product.

6. **Analyze Test Results**

   The generated tests are applied throughout the SPL, taking into account different combinations of features selected in the SPL.

   After running the tests, the results are then evaluated, and compared against each other. Such a comparison takes into account measures like the number of defects detected - based on the analysis of mutants detected for each system -, code coverage, and also the information about how long it took to run the test set, in each CIT tool.

   This analysis makes it possible to define the effectiveness of each CIT tool.

4.2 DEFINITION OF THE EXPERIMENTAL STUDY

The SLR findings reported in the previous chapter led us to observe a lack of empirical studies concerning the analysis of the effectiveness of existing SPL testing tool support. In this effect, we planned to assess the MERCI method aiming to ***analyze the effectiveness of CIT tools regarding their adequacy to SPL engineering projects***. The following research questions drive this investigation:

- ***RQ1. Which tool uncovered the largest number of defects?***
- ***RQ2. Which tool obtained the highest code coverage test?***
- ***RQ3. Which tool required the shortest execution time?***

We selected four CIT tools: ACTS, CATS, PICT Master and VPTag. These are aimed at reducing input data for single systems. Our main intention was to evaluate their adequacy to generate reduced input data for SPL projects.

Table 4.1 Selected SPL projects.

System	Domain	NF	NOF	NC	NM	NP
BankAccount	Bank	6	5	2	12	24
DiGraph	Library	4	3	8	64	8
ExamDB	Database	4	3	4	54	8
PokerSPL	Game	11	5	8	63	28

Legend: [NF] Number of Features, [NOF] Number of Optional Features, [NC] Number of Classes, [NM] Number of Methods, [NP] Number of Products

Based on an automatically generated set of products considering the SPL projects, we analyzed whether the CIT tools were capable to build a reduced, but representative and effective test set. Table 4.1 shows raw data of the selected SPL projects. These were implemented for different domains and they have different sizes.

For each CIT tool, we followed the MERCI method steps. To accomplish the overall goal of this empirical evaluation, we ran product configurations generated with the support of the FeatureHouse composer (APEL; KASTNER; LENGAUER, 2009) - with the support of the FeatureIDE. To further strengthen the study, random sets of tests were generated automatically with the support of the Randoop tool.

Next, we introduce the subject SPL projects used in this study and discuss the reduced products set generated from the analyzed CIT tools.

4.2.1 Tools

With the choice of CIT technique, the 4 tools were selected for implementing pairwise techniques. In addition, we look at their availability (license + access to tutorials, etc.), and recent reports in the literature. After an analysis in each one of the tools the ones that most fit and were available were the tools ACTS, CATS, PICT Master and VPTag.

Next, we present the main characteristics of the four tools designed to handle the paired tests.

- **ACTS - Advanced Combinatorial Testing System**. The tool generates test sets that ensure `t`-way coverage of input parameter values. The ACTS supports `t`-way combinatorial test generation with several advanced features such as mixed-strength test generation for up to a 6-way coverage. The ACTS supports three types of values: Boolean, Integer (`Int`) and Enumerative (`Enum`). The test generation algorithms implemented in ACTS are: `IPOG, IPOG-D, IPOG-F, IPOG-F2` (YU et al., 2013).

- **CATS - Constrained Array Test System**. It is a tool for generating test cases with the objective of improving the coverage and efficiency of the test. The tool provides statistical test plans. It is an online tool, which generates unique test cases to improve test productivity and reduces the number of test cases performed. The

results of the generator are presented in Hyper Text Markup Language (HTML) tables accessible from a browser.

- **PICT Master**. This is a free software that generates combinatorial interaction test cases by using the Pairwise method. It is a CUI-based application (Character User Interface) which runs at the command prompt. PICT Master Master overlays the CUI-based PICT Master with an Excel-based GUI (graphical user interface) shell. The main input for the PICT Master Master is a plain text file. The core generation algorithm uses a greedy heuristic, similar to the `AETG` algorithm. The tool might generate test cases for several types of tests.

- **VPTag - Visual Pairwise Test Array Generator**. This is a tool aimed to automatically deliver test combinations based on the pairwise approach. It is a free software application and uses the Tai-Lei algorithm (TAI; LEI, 2002) to create the tests. VPTag uses the input parameters in pairs to perform the combinations (VISUAL..., 2010). The algorithm that has been applied includes the constraints on the inputs, ensuring that no invalid tests are generated and the constraints are respected.

4.2.2 Subject Programs

We selected SPL projects developed in Java whose source code was available to be used in the FeatureIDE, with the FeatureHouse composer, and the projects developed in Java and that the source code was available. The projects were selected according to the availability of these characteristics.

Although the projects were developed in the same programming language and were available in the same repository, they were developed by different teams. The source code is a mandatory asset for our investigation. Next, we present the four projects that were selected as subjects of this work.

- **BankAccount:** It provides a means to generate distinct projects for managing bank accounts, in terms of the individual features they provide. Figure 4.2 shows the feature Model of this project. It contains six features, five of which are optional. When considering the CIT tools, the CATS tools generated a reduced set of 7 products, while the ACTS, PICT MasterMaster and VPTag tools generated 6 products;

- **DiGraph:** It simulates a library for representing and manipulating directed graph structures. Beside basic graphs, it also supports various operations such as removal, traversal, and transposition, implemented as optional features. The project has four features, from which three are optional ones, as Figure 4.3 shows. In this project, the ACTS, CATS, PICT MasterMaster and VPTag tools generated a reduced set of 4 products each;

- **ExamDB:** This is a tiny SPL project in the Database domain. Figure 4.4 shows the FM of the project. The project can deliver products that manages students' exams,

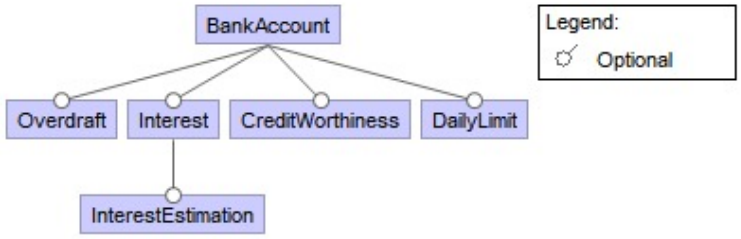

Figure 4.2 Overview of the Feature Model BankAccount.

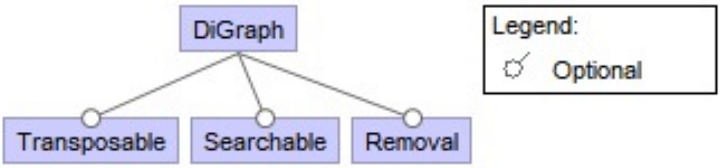

Figure 4.3 Overview of the Feature Model DiGraph.

including features for subscription and backouting, bonus points, and statistics. In this project, the ACTS, CATS, PICT Master and VPTTag tools generated a reduced set of 4 products;

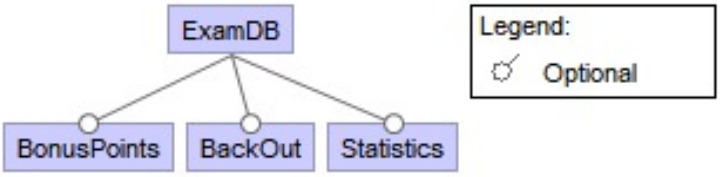

Figure 4.4 Overview of the Feature Model ExamDB.

- **PokerSPL:** This is a project in the Games domain. Figure 4.5 shows the project feature model, which comprises eleven features, from which five are optional ones. The ACTS, PICT Master and VPTag tool generated a reduced set of 8 products, while the CATS generated a reduced set of 6 products.

4.2.3 Hypothesis

Based on our goals, we designed the following hypotheses:

- (H_0) All tools have similar results;
- (H_1) At least one tool stands out from the others.

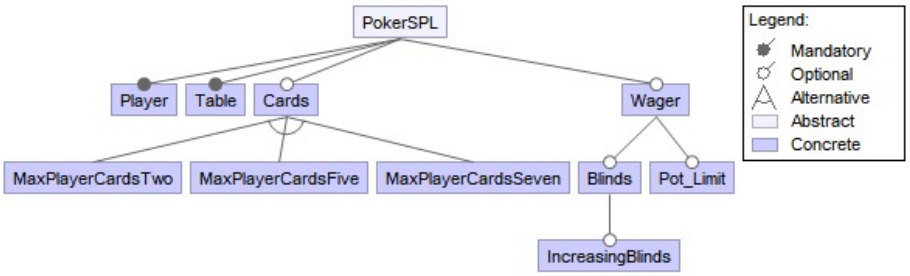

Figure 4.5 Overview of the Feature Model PokerSPL.

Such hypotheses correspond to all research questions, **RQ1**, **RQ2** and **RQ3**.

4.2.4 Statistical Tests

This section discusses the statistical tests used to analyze collected data in relation to all research questions.

Analysis of Variance (ANOVA) is a statistical test used to validate the hypothesis tests. According to Kim (2014), ANOVA is used for a comparison of more than two groups, unidirectional ANOVA is the appropriate method in place of the t-test. For Wohlin et al. (2012) ANOVA is a family of parametric tests that can be can be used to analyze experiments from a number of different designs, and ANOVA can compare projects with one factor, with more than two treatments.

ANOVA aims to observe the total variability of the data and to compare the variability due to the treatment and random error.

Before applying ANOVA, it is necessary to make sure that the following assumptions are respected:

1. The populations have distributions that are approximately normal. If the population does have a distribution that is far from normal, it is recommended to use the Kruskal-Wallis test.

2. The populations have the same variance σ (or standard deviation σ).

3. The samples are simple random samples, that is, samples of the same size have the same probability of being selected.

4. The samples are independent of each other, that is, the sample are not matched or paired in any way.

5. The different samples are from populations that are categorised in only one way.

   The first step to interpret the ANOVA results is understanding that a small p-value (such as 0,05 or less) leads to rejection of the null hypothesis of equal means while a large p-value (such as greater than 0,05), fails to reject the null hypothesis of equal means (TRIOLA, 2006).

Tukey's test: This statistical test is generally used in conjunction with ANOVA to find which means are significantly different from one another. The Tukey's test, also known as Tukey-Kramer method or Tukey's Honestly Significant Difference (HSD) test, is a single-step multiple comparison procedure.

The test compares the means of every treatment to the means of every other treatment; that is, it applies simultaneously to the set of all pairwise comparisons and identifies where the difference between two means is greater than the standard error would be expected to allow (NATRELLA, 2010).

4.3 RESULTS AND DISCUSSION

This section presents the data extracted from the empirical evaluation, used to answer the research questions.

4.3.1 Sampling

Figure 4.6 presents the sample size without product reduction. BankAccount, Digraph, ExamDB, and Poker projects have the size of 24, 8, 8 and 28 products respectively.

After the reduction, the sample size for the **ACTS** tool using the BankAccount, Digraph, Poker, and ExamDB project were respectively 6, 4, 8 and 4 reduced products representing the sample.

The sample size for the **CATS** tool using the BankAccount, Digraph, Poker, and ExamDB project were respectively 7, 4, 6 and 4 reduced products representing the sample.

The sample size for the **PICT Master** tool using the BankAccount project, Digraph, Poker, and ExamDB were respectively 6, 3, 8 and 4 products the sample.

The sample size for the **VPTag** tool using the BankAccount project, Digraph, Poker, and ExamDB were respectively 6, 4, 3 and 4 products the sample.

4.3.2 Defect detection capability (RQ1)

In this research question, quality control is the medium from which it is used to evaluate the quality of the software. Defect detection capability is one of the evaluation metrics based on the history of defects detected during the system test phase.

Data presented in Tables B.6 B.7 B.8 and B.9 in Appendix B, show that the tools generated a subset of reduced products for each tool and for each product. The BankAccount system together with the ACTS tool generated a subset of 6 products, and will be named P1, P2, P3, P4, P5 and P6 and so on for other tools and projects. For the analysis of the results of the experiment, we performed the sum of all the products (P1 + P2 + P3 + P4 + P5 + P6) and applied the statistical tests.

We created sets of automated test cases based upon the four selected SPL projects. After creating the test cases, we seeded mutants in the projects through the PIT tool. Table 4.2 details the number of mutants per project. Moreover, it presents the amount of total mutants and the number of detected mutants (dead mutants) by the tools.

By observing data from Table 4.2, the BankAccount project obtained the smallest number of mutants inserted. This is likely due to the project size. The BankAccount

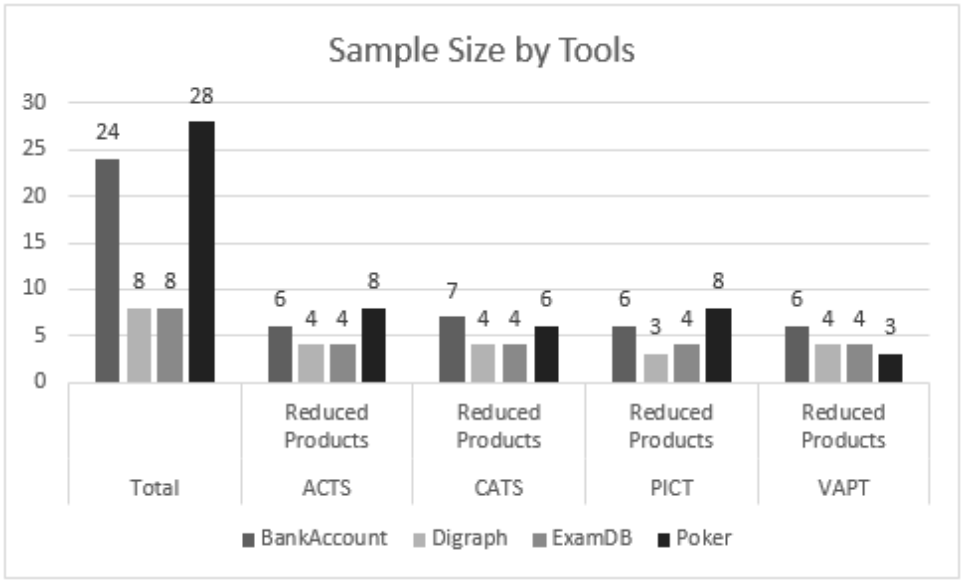

Figure 4.6 Products' Sample Size by Tools

Table 4.2 Number of mutants killed by number of mutants inserte

	ACTS		**CATS**		**PICT Master**		**VPTag**	
	Mutants		**Mutants**		**Mutants**		**Mutants**	
Project/Tool	Killed	Insert	Killed	Insert	Killed	Insert	Killed	Insert
BankAccount	88	237	128	313	99	265	109	291
Digraph	543	1080	507	1080	543	1080	507	1080
ExamDB	737	1247	797	1246	765	1246	797	1246
Poker	412	735	374	670	412	735	159	288

project contains 6 features, but each feature consists of only two classes, one interface (not all features contain interface). And each class has ranged from 15 to 35 lines of codes. Therefore, it obtained the lowest number of mutants inserted.

Figure 4.7 shows the boxplots indicating that the ACTS, CATS, PICT Master and VPTag tools obtained rather similar results regarding the number of dead mutants. On the other hand, Figure 4.8 shows the number of live mutants. The boxplots indicate that the results were similar and the CATS tool presented slightly different values.

In order to evaluate if any of the tools is significantly different from the others, we tested the results statistically through ANOVA.

In this observation, the *p-value* for the ANOVA was 0.936. In other words, it was not possible to reject the null hypothesis (all tools have similar results).

Moreover, to identify different means, we applied the Tukey's test.

Table 4.3 shows the *p-values* of the correlation between the tools. We identified that none of the tools presented statistically significant differences.

Figure 4.9(a) shows the boxplots for the tools' **defect detection capability**. The

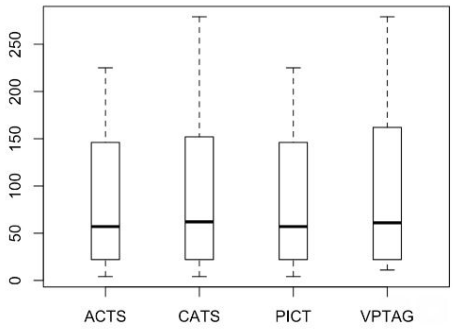

Figure 4.7 Number of dead mutants.

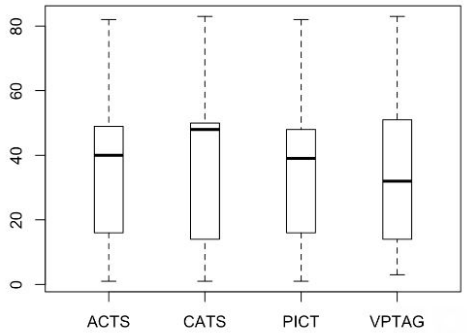

Figure 4.8 Number of live mutants.

horizontal lines crossing the boxplot indicates the median, while the dots show how the values spread over the Y-axis. The tools yielded similar results.

4.3.3 Code coverage (RQ2)

Regarding this research question, we complied the sums of all lines of all subsets, as detailed in Appendix B. The code coverage analysis was performed through the analysis of LOC traversed by the PIT tool, for the mutants detection.

Regarding test coverage, it was not possible to reject the null hypothesis, due to the

Table 4.3 P-value results for defect detection capability.

	ACTS	**CATS**	**PICT Master**
CATS	0,978	-	-
PICT Master	0,999	-	-
VPTag	0,950	-	-
PICT Master	-	0,983	-
VPTag	-	0,998	-
VPTag	-	-	0,958

(a) Defect detection capability (b) Code coverage (c) Test execution length

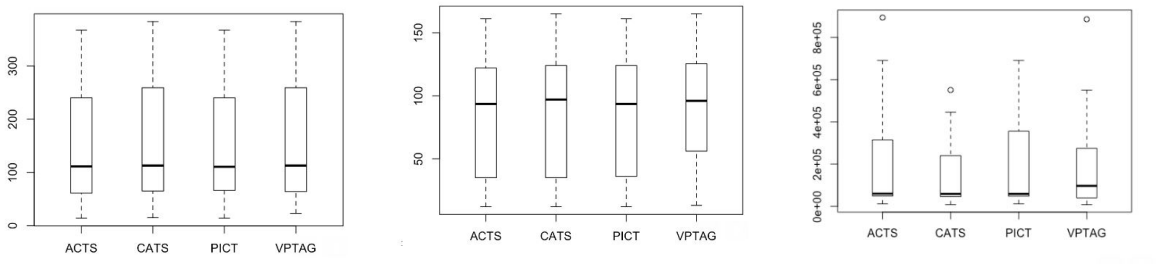

Figure 4.9 Results by tools

value of the correlation between the tools *p-value* was 0.995. To verify the result, we also performed the Tukey's test. Table 4.4 shows the results of such a comparison.

Table 4.4 P-value results for test coverage.

	ACTS	**CATS**	**PICT Master**
CATS	0,999	-	-
PICT Master	0,999	-	-
VPTag	0,994	-	-
PICT Master	-	0,999	-
VPTag	-	0,999	-
VPTag	-		0,996

There is no significant difference between the tools concerning test coverage. Figure 4.9(b) presents the boxplots of test coverage by tools. The tools presented similar results.

4.3.4 Test execution length (RQ3)

In this research question, the execution time was measured in minutes and seconds. Test teams should be able to properly plan their schedules, resources, and estimates for the test run effort, because the effort may be restrictive in practice (ARANHA; BORBA, 2007).

The test run works as follows: (i) the Randoop test generation tool considers minimum requirements coverage and maximum execution effort; (ii) it is analyzed the set of test cases using the PIT mutation tool, where the estimated effort is calculated based on the time that the PIT tool was able to detect the defects.

Once again, it was not possible to reject the null hypothesis, given that *p-value* was 0.882. To check the result, we performed a multiple comparison procedure using Tukey's test. Table 4.5 shows the comparison results.

Figure 4.9(c) presents the boxplots representing the test execution length. We identified outliers in ACTS, CATS, and VPTAG. Moreover, the tools execution length were similar.

Table 4.5 P-value results for test execution length.

	ACTS	**CATS**	**PICT Master**
CATS	0,970	-	-
PICT Master	0,999	-	-
VPTag	0,979	-	-
PICT Master	-	0,960	-
VPTag	-	0,852	-
VPTag	-		0,985

4.3.5 Analysis and Discussion

Among the selected SPL projects (see Table 4.1), we carried out a strategy to reduce the number of products. As a result, we achieved a significant reduction in the number of representative products in Digraph and ExamDB projects. We generated 8 possible products. However, the reduction led to the number of 4 representative products in each project. In addition, the Bank Account project has reasonable average overall, as they generated on average 6 to 7 reduced sets out of 24 possible ones.

With these reduced sets of products, we performed mutation testing, using defect detection capability, test coverage, and test execution length measures. Then, we validated the results of the tools by making a comparison between them.

After analyzing the results, we could observe that the tools yielded very similar results. With a too high *p-value*, it was not possible to reject the null hypothesis, concerning their defect detection capability.

Regarding RQ2, test coverage, we applied the Tukey's test, which indicated that it was not possible to reject the null hypothesis.

Concerning the RQ3, test execution length, we could not reject the null hypothesis as well. Such a study yielded very similar results, when considering distinct CIT tools, and the observed measures: Defect detection capability, Test coverage, and test execution length. Therefore, we could not reject any of the stated null hypotheses. No statistically significant differences could be found between the tools (ACTS, CATS, PICT Master, and VPTAG). This may be an indicator that all the analyzed tools could fit in the SPL engineering context, and not only in the single-systems development.

4.3.6 Threats to Validity

Internal Validity. Because SPL projects acted from different perspectives, such as size, domains, during the inspection, the systems obtained similar results. It might be explained due to the small size of the systems. We understand that larger-sized systems must be included in this analysis.

External Validity. No industrial SPL projects were used in this study; only open source SPL projects created for educational and research purposes were used. To minimize such a threat, we analyzed widely-accepted projects by the SPL research community, which have been used as testbeds for empirical evaluations.

Construct Validity. The MERCI method was developed ye only considering projects developed in Java. This may hinder larger inferences on the basis of our observed results.

Conclusion Validity. We used statistical tests recommended by the empirical software engineering community (WOHLIN et al., 2012). This procedure aimed to minimize issues regarding the conclusions we drew.

4.4 CHAPTER SUMMARY

This chapter presented our proposed method for evaluating the adequacy of CIT tools for testing SPL projects, named MERCI. Besides, we presented an empirical study performed to evaluate four well-accepted CIT tools for single-systems development, and how they could behave when testing SPL projects. The observed results indicate that the tools could be used for both single-systems development and SPL engineering. Indeed, more in-depth studies, with larger samples must be carried out, so we could strengthen the confidence in both the evaluation method and on the adequacy of CIT tools.

Chapter 5

CONCLUSION

There are several testing tools available in the market that may be unable to directly support the SPL testing process. Some tools proposed for SPL testing are neither available nor implemented as a ready-to-use tool. This is a "must go" direction for the SPL research community. In order to make SPL testing a feasible approach in practice, it is necessary to have adequate tool support.

A particular step we took in this dissertation was to analyze existing tool support for single-systems development, in order to evaluate whether they are also suitable for SPL engineering testing. We proposed a method to support such an evaluation. Our results indicate that existing tool support, considering CIT testing, a good approach for reducing test sets, could be adequate to SPL engineering as well.

Whereas we understand this study is a tiny step towards improving SPL testing practice, we believe it could be an interesting starting point to enhance comprehension of the role of existing tool support for testing SPL engineering projects.

According to the data collected and analyzed in the experiment, the method presents indications of its effectiveness. We believe this dissertation is yet another step towards the maturity of testing tools in SPL testing.

We next discuss related work and pinpoint opportunities for further research in this field.

5.1 RELATED WORK

Regarding the empirical evaluation carried out in this dissertation, we identified three similar studies. These are discussed next.

The study conducted by Accioly, Borba and Bonifacio (2012) carried out an empirical evaluation to compare two manual black-box testing techniques, namely a generic technique that was observed in an industrial test environment, and a specific product technique whose functional test cases could be derived using any SPL technique that considers variations in functional tests. The comparison was performed through a controlled experiment with students simulating a real-world test environment. After the

experiment, they reported the differences between the two techniques for testing SPL products.

Another similar study to ours is the work of Perrouin et al. (2012), who performed an experiment using the t-wise technique for SPL and applied them in two sets of independent tools developed by the authors. The authors compared two approaches (CSP-dedicated and Alloy-based) for reducing test cases, both based on the t-wise interaction testing. In order to evaluate the approaches, they used measures such as the number of generated test configurations, and the similarity between them. Both approaches were considered functionally equivalent from the t-wise testing perspective.

Fischer, Lopez-Herrejon and Egyed (2018) presented a benchmark to evaluate the failure detection capabilities of SPL testing approaches. Due to the lack of publicly available SPL tests and actual SPL failures, the authors designed a process to automatically generate tests and failures, consisting of a set of Java-based SPLs for which they generated test sets, and introduced mutations to simulate failures. The work performed through a set of variants to test based on some coverage criteria. The authors claimed that the biggest problem to evaluate such test approaches is to correctly compare them against each other.

All these works present same empirical evaluation to compare existing SPL testing techniques. We consider these as important empirical studies, but they do not discuss how real-world support tools could behave when testing SPL projects. In this effect, we could establish that a key difference between these and our proposal is that our work aims at a comparative analysis of support tools. In practice, any SPL project should count on support tools for generating its test assets, and we believe that our study could provide the SPL research and practitioners community with an affordable strategy to select adequate tools.

5.2 OPEN ISSUES AND FUTURE WORK

We consider this study as an initial endeavor towards establishing a means to make SPL testing an easier activity. One of the most important open issues is how to select a software testing tool, which is both reliable and ready to use. In this sense, we contribute to such advances in SPL testing.

However, we believe that a more in-depth empirical evaluation must be performed so that we could have stronger confidence on the tools to select to real-world SPL projects. We next describe some steps to take in future investigations.

The present dissertation only considered three metrics that were implemented at PIT tool to evaluate the effectiveness of the proposed method. For this reason, more metrics may be included as well.

The design of this study should be used in some replications, which consider larger samples, and not only academic-purposes SPL projects, such as those used from the SPL2GO repository, but also real-world SPL projects. This is a good strategy to mitigate the issues concerning external validity of this study.

We used several tools to derive test sets, to inject faults, and to analyze the results. It could be interesting to have an integrated approach, with an automated support, so

that it could be easier for any ordinary SPL engineer to use the approach and select a given testing tool, based on a set of inputs.

REFERENCES

ABDELGHANY MICHAEL ERNST, W. H. L. I. A. *Randoop Automatic unit test generation for Java Home Page*. 2010. Available in: ⟨https://randoop.github.io/randoop/⟩. [Last access on August 24, 2018].

ACCIOLY, P.; BORBA, P.; BONIFACIO, R. Comparing two black-box testing strategies for software product lines. In: IEEE. *6th Brazilian Symposium on Software Components Architectures and Reuse (SBCARS)*. [S.l.], 2012.

AHMED, A. H.; SIDAHMED, A. A. A.; ELTOUM, R. B. Automation of test scripts in software product line using model driven architecture. In: IEEE. *International Conference on Computing, Control, Networking, Electronics and Embedded Systems Engineering (ICCNEEE)*. [S.l.], 2015.

AL-HAJJAJI, M.; THUM, T.; MEINICKE, J.; LOCHAU, M.; SAAKE, G. Similarity-based prioritization in software product-line testing. In: *18th International Software Product Line Conference - Volume 1*. [S.l.]: ACM, 2014. (SPLC '14).

AMMANN, P. Transforming mutation testing from the technology of the future into the technology of the present. In: *International conference on software testing, verification and validation workshops. IEEE*. [S.l.: s.n.], 2015.

AMMANN, P.; OFFUTT, J. *Introduction to software testing*. [S.l.]: Cambridge University Press, 2016.

APEL, S.; BATORY, D.; KASTNER, C.; SAAKE, G. *Feature-Oriented Software Product Lines*. [S.l.]: Springer, 2013.

APEL, S.; KASTNER, C. An overview of feature-oriented software development. *Journal of Object Technology (JOT)*, 2009.

APEL, S.; KASTNER, C.; LENGAUER, C. Featurehouse: Language-independent, automated software composition. In: IEEE COMPUTER SOCIETY. *31st International Conference on Software Engineering*. [S.l.], 2009.

APEL, S.; KOLESNIKOV, S.; SIEGMUND, N.; KASTNER, C.; GARVIN, B. Exploring feature interactions in the wild: The new feature-interaction challenge. In: *5th International Workshop on Feature-Oriented Software Development (FOSD)*. [S.l.]: ACM, 2013.

ARANHA, E.; BORBA, P. An estimation model for test execution effort. In: IEEE. *1th International Symposium on Empirical Software Engineering and Measurement*. [S.l.], 2007.

ARRIETA, A.; SAGARDUI, G.; ETXEBERRIA, L. Test control algorithms for the validation of cyber-physical systems product lines. In: *19th International Conference on Software Product Line(SPLC)*. [S.l.]: ACM, 2015.

BACH, J.; SCHROEDER, P. J. Pairwise testing: A best practice that isn't. In: CITESEER. *22nd Pacific Northwest Software Quality Conference*. [S.l.], 2004.

BALLER, H.; LOCHAU, M. Towards incremental test suite optimization for software product lines. In: *6th International Workshop on Feature-Oriented Software Development(FOSD)*. [S.l.]: ACM, 2014.

BENAVIDES, D.; SEGURA, S.; RUIZ-CORTÉS, A. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, Elsevier, 2010.

BEOHAR, H.; MOUSAVI, M. R. Input–output conformance testing for software product lines. *Journal of Logical and Algebraic Methods in Programming*, Elsevier, 2016.

BEUCHE, D.; PAPAJEWSKI, H.; SCHRÖDER-PREIKSCHAT, W. Variability management with feature models. *Science of Computer Programming*, Elsevier, 2004.

BORAZJANY, M. N.; YU, L.; LEI, Y.; KACKER, R.; KUHN, R. Combinatorial testing of acts: A case study. In: IEEE. *5th International Conference on Software Testing, Verification and Validation (ICST)*. [S.l.], 2012.

BOURQUE, P.; FAIRLEY, R. E. et al. *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0*. [S.l.]: IEEE Computer Society Press, 2014.

BURNSTEIN, I. *Practical software testing: a process-oriented approach*. [S.l.]: Springer Science & Business Media, 2006.

CHEN, L.; BABAR, M. A.; ALI, N. Variability management in software product lines: A systematic review. In: *13th International Software Product Line Conference (SPLC)*. [S.l.]: Carnegie Mellon University, 2009.

CLEMENTS, P.; NORTHROP, L. *Software Product Lines: Practices and Patterns*. [S.l.]: Addison-Wesley Professional, 2001.

COHEN, D. M.; DALAL, S. R.; FREDMAN, M. L.; PATTON, G. C. The aetg system: An approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering*, IEEE, 1997.

COHEN, M. B.; DWYER, M. B.; SHI, J. Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach. *IEEE Transactions on Software Engineering*, IEEE, 2008.

COLES, H.; LAURENT, T.; HENARD, C.; PAPADAKIS, M.; VENTRESQUE, A. Pit: a practical mutation testing tool for java. In: ACM. *25th International Symposium on Software Testing and Analysis*. [S.l.], 2016.

CZERWONKA, J. Pairwise testing in real world. In: CITESEER. *24th Pacific Northwest Software Quality Conference*. [S.l.], 2006.

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. *Introdução ao teste de software*. [S.l.: s.n.], 2007.

DELAMARO, M. E.; MALDONADO, J. C.; MATHUR, A. P. Integration testing using interface mutation. In: IEEE. *7th International Symposium on Software Reliability Engineering*. [S.l.], 1996.

DEMILLO, R. A. *Mutation Analysis as a Tool for Software Quality Assurance*. [S.l.], 1980.

DEVINE, T.; GOSEVA-POPSTOJANOVA, K.; KRISHNAN, S.; LUTZ, R. R. Assessment and cross-product prediction of software product line quality: accounting for reuse across products, over multiple releases. *Automated Software Engineering*, Springer, 2016.

DEVROEY, X.; PERROUIN, G.; CORDY, M.; SCHOBBENS, P.-Y.; LEGAY, A.; HEYMANS, P. Towards statistical prioritization for software product lines testing. In: *8th International Workshop on Variability Modelling of Software-Intensive Systems (VaMOS)*. [S.l.]: ACM, 2013.

DEVROEY, X.; PERROUIN, G.; LEGAY, A.; SCHOBBENS, P.-Y.; HEYMANS, P. Search-based similarity-driven behavioural spl testing. In: *10th International Workshop on Variability Modelling of Software-intensive Systems(VaMOS)*. [S.l.]: ACM, 2016.

DEVROEY, X.; PERROUIN, G.; SCHOBBENS, P.-Y. Abstract test case generation for behavioural testing of software product lines. In: ACM. *18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools-Volume 2*. [S.l.], 2014.

EDWIN, O. O. *Testing in Software Product Lines*. Tese (Doutorado) — Blekinge Institute of Technology, 2007.

ENGSTROM, E.; RUNESON, P. Software product line testing–a systematic mapping study. *Information and Software Technology*, Elsevier, 2011.

FILHO, R. A. M.; VERGILIO, S. R. A mutation and multi-objective test data generation approach for feature testing of software product lines. In: IEEE. *29th Brazilian Symposium on Software Engineering (SBES)*. [S.l.], 2015.

FISCHER, S.; LINSBAUER, L.; LOPEZ-HERREJON, R. E.; EGYED, A. Enhancing clone-and-own with systematic reuse for developing software variants. In: *International Conference on Software Maintenance and Evolution*. [S.l.: s.n.], 2014.

FISCHER, S.; LOPEZ-HERREJON, R. E.; EGYED, A. Towards a fault-detection benchmark for evaluating software product line testing approaches. 2018.

FISCHER, S.; LOPEZ-HERREJON, R. E.; RAMLER, R.; EGYED, A. A preliminary empirical assessment of similarity for combinatorial interaction testing of software product lines. In: ACM. *9th International Workshop on Search-Based Software Testing*. [S.l.], 2016.

GALINDO, J. A.; TURNER, H.; BENAVIDES, D.; WHITE, J. Testing variability-intensive systems using automated analysis: An application to android. *Software Quality Journal*, Kluwer Academic Publishers, 2016.

GARVIN, B. J.; COHEN, M. B.; DWYER, M. B. An improved meta-heuristic search for constrained interaction testing. In: IEEE. *1st International Symposium on Search Based Software Engineering*. [S.l.], 2009.

GRAHAM, D. R. Testing, verification and validation. In: *Colloquium on Layman's Guide to Software Quality*. [S.l.]: IET, 1993.

GURP, J. V.; BOSCH, J.; SVAHNBERG, M. On the notion of variability in software product lines. In: IEEE. *Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA)*. [S.l.], 2001.

HAJJAJI, M.; KRIETER, S.; THUM, T.; LOCHAU, M.; SAAKE, G. Incling: efficient product-line testing using incremental pairwise sampling. In: ACM. *International Conference on Generative Programming: Concepts and Experiences*. [S.l.], 2016.

HAJJAJI, M.; THUM, T.; LOCHAU, M.; MEINICKE, J.; SAAKE, G. Effective product-line testing using similarity-based product prioritization. *Software & Systems Modeling*, Springer, 2016.

HENARD, C.; PAPADAKIS, M.; PERROUIN, G.; KLEIN, J.; HEYMANS, P.; TRAON, Y. L. Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines. *IEEE Transactions on Software Engineering*, IEEE, 2014.

HERVIEU, A.; BAUDRY, B.; GOTLIEB, A. PACOGEN: automatic generation of pairwise test configurations from feature models. In: *International Symposium on Software Reliability Engineering*. [S.l.]: IEEE, 2011. p. 120–129.

HERVIEU, A.; MARIJAN, D.; GOTLIEB, A.; BAUDRY, B. Practical minimization of pairwise-covering test configurations using constraint programming. *Information and Software Technology*, Elsevier, 2016.

HOWDEN, W. E. Weak mutation testing and completeness of test sets. *IEEE Transactions on Software Engineering*, IEEE, 1982.

IEEE. Ieee standard for software and system test documentation. *IEEE Std 829-2008*, 2008.

JALOTE, P. *An integrated approach to software engineering*. [S.l.]: Springer Science & Business Media, 2012.

JIA, Y.; HARMAN, M. Higher order mutation testing. *Information and Software Technology*, Elsevier, 2009.

JIA, Y.; HARMAN, M. An analysis and survey of the development of mutation testing. *IEEE transactions on software engineering*, IEEE, 2011.

JOHANSEN, M. F. Testing product lines of industrial size: Advancements in combinatorial interaction testing. 2013.

JOHANSEN, M. F.; HAUGEN, O.; FLEUREY, F. Properties of realistic feature models make combinatorial testing of product lines feasible. In: SPRINGER. *International Conference on Model Driven Engineering Languages and Systems*. [S.l.], 2011.

JONES, J. A.; HARROLD, M. J. Test-suite reduction and prioritization for modified condition/decision coverage. *IEEE Transactions on software Engineering*, IEEE, 2003.

KANG, K. C.; COHEN, S. G.; HESS, J. A.; NOVAK, W. E.; PETERSON, A. S. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. [S.l.], 1990.

KIM, H.-Y. Analysis of variance (anova) comparing means of more than two groups. *Restorative dentistry & endodontics*, 2014.

KITCHENHAM, B. Procedures for performing systematic reviews. *Keele, UK, Keele University*, 2004.

KOLB, R. A risk-driven approach for efficiently testing software product lines. In: CITESEER. *5th Researches Workshop GPCE Young, Erfurt, Germany*. [S.l.], 2003.

KUHN, R.; KACKER, R.; LEI, Y.; HUNTER, J. Combinatorial software testing. *Computer*, 2009.

LACHMANN, R.; LITY, S.; HAJJAJI, M.; FüRCHTEGOTT, F.; SCHAEFER, I. Fine-grained test case prioritization for integration testing of delta-oriented software product lines. In: *7th International Workshop on Feature-Oriented Software Development (FOSD)*. [S.l.]: ACM, 2016.

LACHMANN, R.; LITY, S.; LISCHKE, S.; BEDDIG, S.; SCHULZE, S.; SCHAEFER, I. Delta-oriented test case prioritization for integration testing of software product lines. In: *19th International Conference on Software Product Line (SPLC)*. [S.l.]: ACM, 2015.

LACKNER, H.; SCHMIDT, M. Towards the assessment of software product line tests: A mutation system for variable systems. In: *18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools (SPLC)*. [S.l.]: ACM, 2014.

LAMANCHA, B. P.; USAOLA, M. P.; VELTHUIS, M. P. Software product line testing - A systematic review. In: *4th International Conference on Software and Data Technologies (ICSoft 2009), Sofia, Bulgaria*. [S.l.]: INSTICC Press, 2009.

LEE, K.; KANG, K. C.; LEE, J. Concepts and guidelines of feature modeling for product line software engineering. In: SPRINGER. *International Conference on Software Reuse*. [S.l.], 2002.

LIMA-NETO, C. R.; ALMEIDA, E. S.; MEIRA, S. R. L. A software product lines system test case tool and its initial evaluation. In: *13th International Conference on Information Reuse Integration (IRI)*. [S.l.]: IEEE, 2012.

LIMA-NETO, C. R.; NETO, P. A. da M. S.; ALMEIDA, E. S. de; MEIRA, S. R. de L. A mapping study on software product lines testing tools. In: *24th International Conference on Software Engineering & Knowledge Engineering (SEKE)*. [S.l.]: Knowledge Systems Institute Graduate School, 2012.

LITY, S.; MORBACH, T.; THUM, T.; SCHAEFER, I. Applying incremental model slicing to product-line regression testing. In: *15th International Conference on Software Reuse: Bridging with SociAwareness (ICSR)*. [S.l.]: Springer, 2016.

LOCHAU, M.; BURDEK, J.; LITY, S.; HAGNER, M.; LEGAT, C.; GOLTZ, U.; SCHURR, A. Applying model-based software product line testing approaches to the automation engineering domain. *at-Automatisierungstechnik*, 2014.

LOCHAU, M.; LITY, S.; LACHMANN, R.; SCHAEFER, I.; GOLTZ, U. Delta-oriented model-based integration testing of large-scale systems. *Journal of Systems and Software*, Elsevier, 2014.

LOCHAU, M.; OSTER, S.; GOLTZ, U.; SCHURR, A. Model-based pairwise testing for feature interaction coverage in software product line engineering. *Software Quality Journal*, Kluwer Academic Publishers, 2012.

LOPEZ-HERREJON, R. E.; FERRER, J.; CHICANO, F.; EGYED, A.; ALBA, E. Comparative analysis of classical multi-objective evolutionary algorithms and seeding strategies for pairwise testing of software product lines. In: IEEE. *IEEE Congress on Evolutionary Computation (CEC)*. [S.l.], 2014.

LOPEZ-HERREJON, R. E.; FERRER, J. J.; CHICANO, F.; HASLINGER, E. N.; EGYED, A.; ALBA, E. A parallel evolutionary algorithm for prioritized pairwise testing of software product lines. In: ACM. *Conference on Genetic and Evolutionary Computation*. [S.l.], 2014.

LOPEZ-HERREJON, R. E.; FISCHER, S.; RAMLER, R.; EGYED, A. A first systematic mapping study on combinatorial interaction testing for software product lines. In: IEEE. *8th International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. [S.l.], 2015.

LUO, L. Software testing techniques. *Institute for software research international Carnegie mellon university Pittsburgh, PA*, 2001.

MACHADO, I. do C.; MCGREGOR, J. D.; CAVALCANTI, Y. C.; ALMEIDA, E. S. D. On strategies for testing software product lines: A systematic literature review. *Information and Software Technology*, Elsevier, 2014.

MACHADO, I. do C.; NETO, P. A. da M. S.; ALMEIDA, E. S. de; MEIRA, S. R. de L. Riple-te: A process for testing software product lines. In: *23rd International Conference on Software Engineering & Knowledge Engineering (SEKE)*. [S.l.]: Knowledge Systems Institute Graduate School, 2011.

MCGREGOR, J. Testing a software product line. 2001.

MURNANE, T.; REED, K. On the effectiveness of mutation analysis as a black box testing technique. In: IEEE. *Software Engineering Conference*. [S.l.], 2001.

NATRELLA, M. Nist/sematech e-handbook of statistical methods. NIST/SEMATECH, 2010.

NETO, P. A. d. M. S.; MACHADO, I. do C.; MCGREGOR, J. D.; ALMEIDA, E. S. D.; MEIRA, S. R. de L. A systematic mapping study of software product lines testing. *Information and Software Technology*, Elsevier, 2011.

OFFUTT, A. J.; UNTCH, R. H. Mutation 2000: Uniting the orthogonal. In: *Mutation testing for the new century*. [S.l.]: Springer, 2001.

OLIMPIEW, E. M. *Model-based testing for software product lines*. Tese (Doutorado), 2008.

OSTER, S.; MARKERT, F.; RITTER, P. Automated incremental pairwise testing of software product lines. In: SPRINGER. *International Conference on Software Product Lines*. [S.l.], 2010.

OSTER, S.; ZORCIC, I.; MARKERT, F.; LOCHAU, M. Moso-polite: tool support for pairwise and model-based software product line testing. In: ACM. *5th Workshop on Variability Modeling of Software-Intensive Systems*. [S.l.], 2011.

PAREJO, J. A.; SANCHEZ, A. B.; SEGURA, S.; RUIZ-CORTES, A.; LOPEZ-HERREJON, R. E.; EGYED, A. Multi-objective test case prioritization in highly configurable systems: A case study. *Journal of Systems and Software*, Elsevier, 2016.

PERROUIN, G.; OSTER, S.; SEN, S.; KLEIN, J.; BAUDRY, B.; TRAON, Y. L. Pairwise testing for software product lines: comparison of two approaches. *Software Quality Journal*, Springer, 2012.

PETKE, J.; COHEN, M. B.; HARMAN, M.; YOO, S. Practical combinatorial interaction testing: Empirical findings on efficiency and early fault detection. *IEEE Transactions on Software Engineering*, IEEE, 2015.

PETTICREW, M.; ROBERTS, H. *Systematic reviews in the social sciences: A practical guide*. [S.l.]: John Wiley & Sons, 2008.

POHL, K.; BOCKLE, G.; LINDEN, F. J. v. *Software Product Line Engineering: Foundations, Principles and Techniques*. [S.l.]: Springer-Verlag New York, Inc., 2005.

PRESSMAN, R. S. *Software engineering: a practitioner's approach*. [S.l.]: Palgrave Macmillan, 2005.

REULING, D.; BURDEK, J.; ROTARMEL, S.; LOCHAU, M.; KELTER, U. Fault-based product-line testing: Effective sample generation based on feature-diagram mutation. In: *19th International Conference on Software Product Line (SPLC)*. [S.l.]: ACM, 2015.

ROTHBERG, V.; DIETRICH, C.; ZIEGLER, A.; LOHMANN, D. Towards scalable configuration testing in variable software. In: *ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences (GPCE)*. [S.l.]: ACM, 2016.

RUBIN, J.; CHECHIK, M. Locating distinguishing features using diff sets. In: ACM. *27th IEEE/ACM International Conference on Automated Software Engineering*. [S.l.], 2012.

SANCHEZ, A. B.; SEGURA, S.; PAREJO, J. A.; CORTÉS, A. R. Variability testing in the wild: the drupal case study. *Software and System Modeling*, 2017.

SANCHEZ, A. B.; SEGURA, S.; RUIZ-CORTES, A. The drupal framework: A case study to evaluate variability testing techniques. In: *Proceedings of the 8th International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)*. [S.l.]: ACM, 2013.

SANCHEZ, A. B.; SEGURA, S.; RUIZ-CORTES, A. A comparison of test case prioritization criteria for software product lines. In: IEEE. *7th International Conference on Software Testing, Verification and Validation (ICST)*. [S.l.], 2014.

SHATNAWI, A.; SERIAI, A.; SAHRAOUI, H. Recovering Architectural Variability of a Family of Product Variants. In: *Software Reuse for Dynamic Systems in the Cloud and Beyond*. [S.l.]: Springer International Publishing, 2015, (Lecture Notes in Computer Science).

SHERWOOD, G. Effective testing of factor combinations. In: *3th International Conference on Software. Test., Anal., and Rev.(STAR94)*. [S.l.: s.n.], 1994.

SOMMERVILLE, I. Engenharia de software, 8ª edição, tradução: Selma shin shimizu mel-nikoff, reginaldo arakaki, edilson de andrade barbosa. *São Paulo: Pearson Addison-Wesley*, 2007.

STRICKLER, A.; LIMA, J. A. P.; VERGILIO, S. R.; POZO, A. T. Deriving products for variability test of feature models with a hyper-heuristic approach. *Applied Soft Computing*, Elsevier Science Publishers B. V., 2016.

TAI, K.-C.; LEI, Y. A test generation strategy for pairwise testing. *IEEE Transactions on Software Engineering*, IEEE, 2002.

TEVANLINNA, A.; TAINA, J.; KAUPPINEN, R. Product family testing: A survey. *SIGSOFT Software Engineering Notes*, ACM, 2004.

THUM, T.; KASTNER, C.; BENDUHN, F.; MEINICKE, J.; SAAKE, G.; LEICH homas. Featureide: An extensible framework for feature-oriented software development. *Sci. Comput. Program.*, 2014.

TRAVASSOS, G. H.; GUROV, D.; AMARAL, E. *Introdução a engenharia de software experimental*. [S.l.]: UFRJ, 2002.

TRIOLA, M. F. *Elementary statistics*. [S.l.]: Pearson/Addison-Wesley Reading, MA, 2006.

VIDACS, L.; HORVATH, F.; MIHALICZA, J.; VANCSICS, B.; BESZEDES, A. Supporting software product line testing by optimizing code configuration coverage. In: *8th IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. [S.l.: s.n.], 2015.

VISUAL Pair-wise Test Array Generator. VPTag, 2010. Disponível em: ⟨https://sourceforge.net/projects/vptag/⟩.

WANG, S.; ALI, S.; GOTLIEB, A. Cost-effective test suite minimization in product lines using search techniques. *Journal of Systems and Software*, Elsevier, 2015.

WANG, S.; BUCHMANN, D.; ALI, S.; GOTLIEB, A.; PRADHAN, D.; LIAAEN, M. Multi-objective test prioritization in software product line testing: an industrial case study. In: *18th International Software Product Line Conference*. [S.l.]: ACM, 2014.

WOHLIN, C. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE)*. [S.l.]: ACM, 2014.

WOHLIN, C.; RUNESON, P.; HOST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLEN, A. *Experimentation in software engineering*. [S.l.]: Springer Science & Business Media, 2012.

YANG, Q.; LI, J. J.; WEISS, D. M. A survey of coverage-based testing tools. *The Computer Journal*, OUP, 2009.

YILMAZ, C.; FOUCHE, S.; COHEN, M. B.; PORTER, A.; DEMIROZ, G.; KOC, U. Moving forward with combinatorial interaction testing. *Computer*, IEEE, 2014.

YU, L.; LEI, Y.; KACKER, R. N.; KUHN, D. R. Acts: A combinatorial test generation tool. In: IEEE. *6th International Conference on Software Testing, Verification and Validation (ICST)*. [S.l.], 2013.

APPENDIX A

SYSTEMATIC LITERATURE REVIEW - PRIMARY STUDIES

This Appendix presents detailed information on the primary studies included the Systematic Literature Review discussed in Chapter 3.

A.1 PRIMARY STUDIES

Table A.1 Primary studies

ID	Title	Author(s)	Venue
P1	Abstract Test Case Generation for Behavioural Testing of Software Product Lines	Devroey, Perrouin and Schobbens (2014)	SPLC'14
P2	A Comparison of Test Case Prioritization Criteria for Software Product Lines	Sanchez, Segura and Ruiz-Cortes (2014)	ICST'14
P3	A Mutation and Multi-objective Test Data Generation Approach for Feature Testing of Software Product Lines	Filho and Vergilio (2015)	SBES'15
P4	A parallel evolutionary algorithm for prioritized pairwise testing of software product lines	Lopez-Herrejon et al. (2014b)	GECCO'14
P5	A Preliminary Empirical Assessment of Similarity for Combinatorial Interaction Testing of Software Product Lines	Fischer et al. (2016)	SBST'16
P6	Applying Incremental Model Slicing to Product-Line Regression Testing	Lity et al. (2016)	ICSR'16
P7	Applying Model-based Software Product Line Testing Approaches to the Automation Engineering Domain	Lochau et al. (2014a)	AT'14
P8	Assessment and cross-product prediction of software product line quality: accounting for reuse across products, over multiple releases	Devine et al. (2016)	ASE'16
P9	Automation of test scripts in software product line using Model driven architecture	Ahmed, SidAhmed and Eltoum (2015)	ICCNEEE'15
P10	Bypassing the Combinatorial Explosion: Using Similarity to Generate and Prioritize T-Wise Test Configurations for Software Product Lines	Henard et al. (2014)	TSE'14
P11	Comparative analysis of classical multi-objective evolutionary algorithms and seeding strategies for pairwise testing of Software Product Lines	Lopez-Herrejon et al. (2014a)	CEC'14
P12	Cost-effective test suite minimization in product lines using search techniques	Wang, Ali and Gotlieb (2015)	JSS'14

Continued on next page.

Table A.2 Continued.

ID	Title	Author(s)	Venue
P13	Delta-oriented model-based integration testing of large-scale systems	Lochau et al. (2014b)	JSS'15
P14	Delta-oriented Test Case Prioritization for Integration Testing of Software Product Lines	Lachmann et al. (2015)	SPLC '15
P15	Deriving products for variability test of Feature Models with a hyper-heuristic approach	Strickler et al. (2016)	ASC'16
P16	Effective product-line testing using similarity-based product prioritization	Hajjaji et al. (2016b)	SSM'15
P17	Fault-based Product-Line Testing Effective Sample Generation based on Feature-Diagram Mutation	Reuling et al. (2015)	SPLC'15
P18	Fine-grained Test Case Prioritization for Integration Testing of Delta-oriented Software Product Lines	Lachmann et al. (2016)	FOSD'16
P19	IncLing: Efficient Product-line Testing Using Incremental Pairwise Sampling	Hajjaji et al. (2016a)	GPCE'16
P20	Input-output conformance testing for software product lines	Beohar and Mousavi (2016)	JLAMP'16
P21	Multi-objective test case prioritization in highly configurable systems: A case study	Parejo et al. (2016)	JSS'16
P22	Multi-objective Test Prioritization in Software Product Line Testing: An Industrial Case Study	Wang et al. (2014)	SPLC'14
P23	Practical minimization of pairwise-covering test configurations using constraint programming	Hervieu et al. (2016)	IST'16
P24	Search-based similarity-driven behavioural SPL testing	Devroey et al. (2016)	VaMoS'16
P25	Similarity-based Prioritization in Software Product-line Testing	Al-Hajjaji et al. (2014)	SPLC'14
P26	Supporting software product line testing by optimizing code configuration coverage	Vidacs et al. (2015)	STTT'15
P27	Test Control Algorithms for the Validation of Cyber-physical Systems Product Lines	Arrieta, Sagardui and Etxeberria (2015)	SPLC'15
P28	The Drupal Framework A Case Study to Evaluate Variability Testing Techniques	Sanchez, Segura and Ruiz-Cortes (2013)	VaMoS'13
P29	Towards Incremental Test Suite Otimization for Software Product Lines	Baller and Lochau (2014)	FOSD'14
P30	Towards scalable configuration testing in variable software	Rothberg et al. (2016)	GPCE'16
P31	Towards Statistical Prioritization for Software Product Lines Testing	Devroey et al. (2013)	VaMoS'14
P32	Towards the Assessment of Software Product Line Tests: A Mutation System for Variable Systems	Lackner and Schmidt (2014)	SPLC'14
P33	Variability testing in the wild: the Drupal case study	Sanchez et al. (2017)	SoSyM'15

APPENDIX B

EXPERIMENTAL EVALUATION

This appendix presents raw data for all the observations from the experiment carried out in this dissertation.

B.1 SAMPLING

Table B.1 shows the number of test case sets for each product that has been reduced regarding each of the analyzed CIT tools.

Table B.1 Number of Sets of Test Cases

Tools	Products	BankAccount	Poker	DiGraph	ExamDB
ACTS	P1	26	2	2	6
	P2	27	3	2	6
	P3	27	3	2	6
	P4	26	4	2	5
	P5	28	3	-	-
	P6	28	3	-	-
CATS	P1	23	3	2	8
	P2	26	4	2	7
	P3	27	4	2	5
	P4	24	3	2	4
	P5	25	3	-	-
	P6	21	3	-	-
	P7	25	3	-	-
PICT Maste	P1	31	3	2	6
	P2	27	3	2	6
	P3	29	3	2	5
	P4	27	3	2	6
	P5	23	3	-	-
	P6	28	2	-	-
VPTag	P1	23	4	2	8
	P2	26	4	2	7
	P3	29	3	2	5
	P4	27	2	2	4
	P5	30	4	-	-
	P6	28	3	-	-

Table B.2 shows the number of test case for each product that has been reduced regarding each of the analyzed CIT tools.

Table B.2 Number of Test Cases

Tools	Products	BankAccount	Poker	DiGraph	ExamDB
ACTS	P1	12215	134	203	2325
	P2	12285	727	60	2331
	P3	12560	587	216	2041
	P4	12416	1084	301	1598
	P5	13439	693	-	-
	P6	13404	655	-	-
	P7	-	950	-	-
	P8	-	1036	-	-
CATS	P1	12215	1016	280	3442
	P2	12416	947	387	2764
	P3	12560	645	136	1695
	P4	10714	623	57	1466
	P5	11600	727	-	-
	P6	9909	984	-	-
	P7	11982	-	-	-
PICT Master	P1	14750	693	60	2331
	P2	12285	727	203	2325
	P3	13946	940	301	1598
	P4	12560	645	216	2041
	P5	12215	134	-	-
	P6	13404	1084	-	-
	P7	-	1016	-	-
	P8	-	587	-	-
VPTag	P1	12215	134	280	3442
	P2	12416	1016	387	2764
	P3	13729	640	136	1695
	P4	12977	-	57	1466
	P5	13686	-	-	-
	P6	13439	-	-	-

Table B.3 shows the number of classes that each product contains.

Table B.3 Number of classes by project

Tools	Products	BankAccount	Poker	DiGraph	ExamDB
ACTS	P1	2	6	6	3
	P2	2	8	5	3
	P3	2	6	5	3
	P4	2	8	4	3
	P5	2	6	-	-
	P6	2	8	-	-
	P7	-	7	-	-
	P8	-	8	-	-
CATS	2	8	6	3	
	P1	2	8	4	3
	P2	2	8	5	3
	P3	2	8	5	3
	P4	2	8	-	-
	P5	2	8	-	-
	P6	2	-	-	-
PICT Master	2	6	5	3	
	P1	2	8	6	3
	P2	2	7	4	3
	P3	2	8	5	3
	P4	2	6	-	-
	P5	2	8	-	-
	P6	-	8	-	-
	P7	-	6	-	-
VPTag	2	6	6	3	
	P1	2	8	4	3
	P2	2	8	5	3
	P3	2		5	3
	P4	2	-	-	-
	P5	2	-	-	-

Table B.4 shows the total number of lines and the number of lines covered by the PIT tool.

Table B.4 Line Coverage

Tools	Products	BankAccount	Poker	Digraph	ExamDB
ACTS	P1	8/21	76/76	137/161	83/102
	P2	16/26	118/127	84/96	106/133
	P3	10/23	76/76	126/154	102/132
	P4	17/25	122/122	25/81	72/91
	P5	16/35	76/76	-	-
	P6	8/12	120/120	-	-
	P7	-	102/102	-	-
	P8	-	122/122	-	-
CATS	P1	17/36	124/124	133/165	123/138
	P2	17/25	122/122	29/85	97/127
	P3	10/23	118/118	121/150	77/96
	P4	7/12	120/120	72/92	78/97
	P5	18/35	118/127	-	-
	P6	17/35	124/124	-	-
	P7	9/13	-	-	-
PICT Master	P1	8/22	76/76	84/96	106/133
	P2	16/26	118/127	137/161	83/102
	P3	18/34	102/102	25/81	72/91
	P4	10/23	118/118	126/154	102/132
	P5	17/36	76/76	-	-
	P6	8/12	122/122	-	-
	P7	-	124/124	-	-
	P8	-	76/76	-	-
VPTag	17/36	76/76	133/165	123/138	
	P1	17/25	124/124	29/85	97/127
	P2	9/22	118/118	121/150	77/96
	P3	8/13		72/92	78/97
	P4	19/35	-	-	-
	P5	16/35	-	-	-

Table B.5 shows Min, 1st Qu., Median, Mean, 3rd Qu. Max. and stdev., related to the code coverage.

Table B.5 Mean related to the line coverage

	Min	1st Qu.	Median	Mean	3rd Qu.	Max.	stdev.
ACTS	12.00	45.25	93.50	86.95	122.00	161.00	46.05222
CATS	12.00	35.00	97.00	88.76	124.00	165.00	49.52868
PICT Master	12.00	46.00	93.50	87.45	123.50	161.00	45.39743
VPTag	12.00	35.00	97.00	88.76	124.00	165.00	49.52868

Table B.6 shows the amount of total mutants inserted, the dead mutants, the mutants not detected by the PIT tool in relation to the BankAccount project.

Table B.6 Number of Mutants in the BankAccount Project

Tools	**Products**	**Total**	**Dead Mutants**	**Alive Mutants**	**Undetected**
ACTS	P1	31	4	9	18
	P2	49	22	16	11
	P3	42	19	3	20
	P4	40	21	12	7
	P5	61	11	21	29
	P6	14	11	1	1
CATS	P1	69	22	18	29
	P2	40	21	12	7
	P3	42	19	3	20
	P4	15	4	7	4
	P5	60	21	14	25
	P6	65	22	18	25
	P7	22	19	1	2
PICT Master	P1	35	4	9	22
	P2	49	22	16	11
	P3	56	21	14	21
	P4	42	19	3	20
	P5	69	22	18	29
	P6	14	11	1	1
VPTag	P1	69	22	18	29
	P2	40	21	12	7
	P3	34	11	3	20
	P4	23	15	4	4
	P5	64	29	14	21
	P6	61	11	21	29

Table B.7 shows the amount of total mutants inserted, the dead mutants, the mutants not detected by the PIT tool in relation to the PokerSPL project.

Table B.7 Number of Mutants in the PokerSPL Project

Tools	Products	Total	Dead Mutants	Alive Mutants	Undetected
ACTS	P1	66	34	32	-
	P2	113	62	48	3
	P3	66	37	29	-
	P4	112	64	48	-
	P5	66	37	29	-
	P6	110	61	49	-
	P7	90	53	39	-
	P8	112	64	49	-
CATS	P1	113	64	49	-
	P2	112	62	50	-
	P3	109	61	48	-
	P4	110	61	49	-
	P5	113	62	48	3
	P5	113	64	49	-
PICT Master	P1	66	37	29	-
	P2	113	62	48	3
	P3	90	53	37	-
	P4	109	61	48	-
	P5	66	34	32	-
	P6	112	64	48	-
	P7	113	64	49	-
	P8	66	37	29	-
VPTag	P1	66	34	32	-
	P2	113	64	49	-
	P3	109	61	48	-

Table B.8 shows the amount of total mutants inserted, the dead mutants, the mutants not detected by the PIT tool in relation to the Digraph project.

Table B.8 Number of Mutants in the Digraph Project

Tools	Products	Total	Dead Mutants	Alive Mutants	Undetected
ACTS	P1	354	197	82	75
	P2	206	121	43	42
	P3	337	193	56	88
	P4	183	32	12	139
CATS	P1	357	179	83	95
	P2	186	35	12	139
	P3	334	189	61	84
	P4	203	104	30	69
PICT Master	P1	206	121	43	42
	P2	354	197	82	75
	P3	183	32	12	139
	P4	337	193	56	88
VPTag	P1	357	179	83	95
	P2	186	35	12	139
	P3	334	189	61	84
	P4	203	104	30	69

Table B.9 shows the amount of total mutants inserted, the dead mutants, the mutants not detected by the PIT tool in relation to the ExamDB project.

Table B.9 Number of Mutants in the Project ExamDB

Tools	Products	Total	Dead Mutants	Alive Mutants	Undetected
ACTS	P1	275	175	47	53
	P2	364	225	69	70
	P3	367	219	72	76
	P4	240	146	41	53
CATS	P1	383	279	64	40
	P2	348	204	68	76
	P3	259	162	44	53
	P4	256	152	51	53
PICT Master	P1	364	225	69	70
	P2	275	175	47	53
	P3	240	146	41	53
	P4	367	219	72	76
VPTag	P1	383	279	64	40
	P2	348	204	68	76
	P3	259	162	44	53
	P4	256	152	51	53

Tables B.10, B.11, B.12 show Min, 1st Qu., Median, Mean, 3rd Qu. Max. and stdev. regarding data for the Number of dead mutants, Number of live mutants and Defect detection capability respectively.

Table B.10 Mean related to the dead mutants

	Min	1st Qu.	Median	Mean	3rd Qu.	Max.	stdev.
ACTS	4.00	24.50	57.00	82.18	139.75	225.00	7485518
CATS	4.00	22.00	62.00	87.91	146.00	279.00	7587139
PICT Master	4.00	24.50	57.00	82.68	139.75	225.00	7439239
VPTag	11.00	22.00	61.00	92.47	162.00	279.00	8457624

Table B.11 Mean related to the live mutants

	Min	1st Qu.	Median	Mean	3rd Qu.	Max.	stdev.
ACTS	1.0	17.25	40.00	36.68	48.75	82.0	2252488
CATS	1.0	14.0	48.0	37.1	50.0	83.0	23.84514
PICT Master	1.0	16.5	39.0	36.5	48.0	82.0	2250238
VPTag	3.00	14.00	32.00	36.12	51.00	83.0	2448694

Table B.12 Mean related to the defect detection

	Min	1st Qu.	Median	Mean	3rd Qu.	Max.	stdev.
ACTS	14.00	62.25	111.00	149.91	231.50	367.00	120.0373
CATS	15.0	66.0	113.0	164.6	258.2	383.0	121.1782
PICT Master	14.0	66.0	110.5	151.2	231.5	367.0	118.9244
VPTag	23.0	64.0	113.0	170.9	259.0	383.0	128.3734

Table B.13 presents the time of complete execution for the detection of mutants in relation to the products reduction BankAccount and Poker.

Table B.13 Execution Time for the BankAccount and Poker projects

Tools	Products	BankAccount	Poker
ACTS	P1	7 min. and 12 sec.	16 sec.
	P2	13 min. and 16 sec.	58 sec.
	P3	4 min. and 14 sec.	48 sec.
	P4	8 min. and 52 sec.	1 min. and 1 sec.
	P5	17 min. and 12 sec.	49 sec.
	P6	5 min. and 50 sec.	52 sec.
	P7	-	58 sec.
	P8	-	1 min. and 1 sec.
CATS	P1	10 min. and 56 sec.	57 sec.
	P2	8 min. and 52 sec.	55 sec.
	P3	4 min. and 14 sec.	47 sec.
	P4	5 min. and 43 sec.	45 sec.
	P5	9 min. and 09 sec.	58 sec.
	P6	8 min. and 35 sec.	58 sec.
	P7	2 min. and 46 sec.	-
PICT Master	P1	9 min. and 35 sec.	49 sec.
	P2	14 min. and 16 sec.	58 sec.
	P3	11 min. and 20 sec.	57 sec.
	P4	4 min. and 16 sec.	47 sec.
	P5	10 min. and 56 sec.	16 sec.
	P6	5 min. and 50sec.	1 min. and 1 sec.
	P7	-	57 sec.
	P8	-	48 sec.
VPTag	P1	10 min. and 56 sec.	14 sec.
	P2	8 min. and 52 sec.	57 sec.
	P3	4 min. and 35 sec.	53 sec.
	P4	4 min. and 58 sec.	-
	P5	10 min. and 42 sec.	-
	P6	17 min. and 12 sec.	-

Table B.14 presents the time of complete execution for the detection of mutants regarding data for the projects Digraph and ExamDB.

Table B.14 Execution Time for the Digraph and ExamDB projects

Tools	**Products**	**DiGraph**	**ExamDB**
ACTS	P1	30 sec.	3 min. and 12 sec.
	P2	14 sec.	7 min. and 09 sec.
	P3	33 sec.	5 min. and 56 sec.
	P4	12 sec.	2 min. and 01 second
	P5	-	18 min. and 18 sec.
CATS	P1	40 sec.	4 min. and 35 sec.
	P2	20 sec.	3 min. and 57 sec.
	P3	31 second	1 minute and 37 sec.
	P4	8 sec.	1 minute and 37 sec.
PICT Master	P1	14 sec.	7 min. and 09 sec.
	P2	20 sec.	3 min. and 12 sec.
	P3	12 sec.	2 min. and 01 second
	P4	33 sec.	5 min. and 56 sec.
VPTag	P1	40 sec.	4 min. and 35 sec.
	P2	20 sec.	3 min. and 57 sec.
	P3	31 second	1 minute and 37 sec.
	P4	8 sec.	1 minute and 37 sec.

Table B.15 show Min, 1st Qu., Median, Mean, 3rd Qu. Max. and stdev. regarding data for the Test execution length in all the projects.

Table B.15 Mean related to the Test execution length

	Min	1st Qu.	Median	Mean	3rd Qu.	Max.	stdev.
ACTS	12000	49250	60200	184754	284250	894000	236400
CATS	8000	46000	59000	155090	240100	551000	166584
PICT Master	12000	48250	59100	188064	315000	691000	224578
VPTag	8000	40000	97000	212571	275000	886000	244985