,,

PAULO ANSELMO DA MOTA SILVEIRA NETO

ASSESSING SECURITY IN SOFTWARE PRODUCT LINES: A MAINTENANCE ANALYSIS

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/posgraduacao

Recife
2017

PAULO ANSELMO DA MOTA SILVEIRA NETO

ASSESSING SECURITY IN SOFTWARE PRODUCT LINES: A MAINTENANCE ANALYSIS

Tese apresentada ao Centro de Informática da Universidade Federal de Pernambuco, como parte dos requisitos necessários à obtenção do título de Doutor em Ciência da Computação.

Orientador: Vinicius Cardoso Garcia
Coorientador: Eduardo Santana de Almeida

Recife
2017

Catalogação na fonte
Bibliotecário Jefferson Luiz Alves Nazareno CRB 4-1758

S587a Silveira Neto, Paulo Anselmo da Mota.
Assessing security in software product lines; a maintenance analysis / Paulo Anselmo da Mota Silveira Neto – 2017.
176.: fig., tab.

Orientador: Vinícius Cardoso Garcia.
Tese (Doutorado) – Universidade Federal de Pernambuco. CIn. Ciência da Computação, Recife, 2017.
Inclui referências, apêndice e anexo.

1. Engenharia de software. 2. Segurança de software. I. Garcia, Vinícius Cardoso. (Orientador). II. Titulo.

005.3 CDD (22. ed.) UFPE-MEI 2017-205

Paulo Anselmo da Mota Silveira Neto

Assessing Security in Software Product Lines: A Maintenance Analysis

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação

Aprovado em: 02/06/2017.

_____

Orientador: Prof. Dr. Vinicius Cardoso Garcia

BANCA EXAMINADORA

_____

Prof. Dr. Paulo Henrique Monteiro Borba
Centro de Informática / UFPE

_____

Prof. Dr. Nelson Souto Rosa
Centro de Informática / UFPE

_____

Prof. Dr. Marco Tulio de Oliveira Valente
Departamento de Ciência da Computação / UFMG

_____

Prof. Dr. Uirá Kulesza
Departamento de Informática e Matemática Aplicada / UFRN

_____

Prof. Dr. Vander Ramos Alves
Departamento de Ciência da Computação / UnB

I dedicate this work to all my family, friends and professors who gave me the necessary support to get here. . . .

Abstract

Different terms such as "the real-time enterprise", "software infrastructures", "service oriented architectures" and "composite software applications" have gained importance in industry. It brings us the need of information systems that support cross-application integration, cross-company transactions and end-user access through a range of channels, including the Internet. In this context, Software Product Line (SPL) Engineering has gained importance by product oriented companies, as a strategy to cope with the increasing demand of large-scale product customization, providing an effective and efficient ways of improving productivity, software quality, and time-to-market. These benefits combined with the need of most applications interact with other applications, and the internet access makes critical assets vulnerable to many threats. For most of the product oriented companies, security requirements are likely to be as varied as for any other quality. Thus, it is important to supply variants of the same product to satisfy different needs. Owing to its variability management capabilities, software product line architectures can satisfy these requirements if carefully designed the resulting system has a better chance of meeting its expectations. All these requirements should be achieved at early design phases. Otherwise the cost to design a secure architecture will increase, which could worsen in SPL context, due to its complexity. In this context, this thesis evaluates different techniques to implement security tactics for the purpose of assessing conditional compilation and aspect-oriented programming as variability mechanisms concerning maintainability by accessing code size, separation of concerns, coupling and cohesion from software architects in the context of Software Product Lines projects. Hence, to better support SPL architects during design decisions, a family of experiments using three different testbeds was performed to analyze different security techniques regarding to maintainability. We have found that for most of the techniques conditional compilation had a smaller amount of lines of code when compared with Aspect Oriented Programming. The separation of concerns attribute had the low impact on maintainability when implemented with aspect-oriented programming. The analysis also showed that detect attack techniques are less costly than resist attack techniques. The results are useful for both researchers and practitioners. On the one hand, researchers can identify useful research directions and get guidance on how the security techniques impact on maintainability. On the other hand, practitioners can benefit from this thesis by identifying the less costly variability implementation mechanism, as well as, learning concrete techniques to implement security tactics at the code level.

Keywords: Software Engineering. Software Security. Tactics. Design Patterns. Non functional properties.

Resumo

Diferentes termos como “empresa em tempo real”, “infraestrutura de software”, “arquiteturas orientadas a serviço” e “aplicações de software” tem ganhado importância na indústria. Isso requer sistemas de informação que suportem a integração com outras aplicações, transações entre empresas e acesso ao usuário final por uma variedade de canais, incluindo internet. Nesse contexto, Linha de Produto de Software (LPS) tem ganhado importância por empresas orientadas a produtos de software, como uma estratégia para lidar com a crescente demanda de personalização de produtos em grande escala, proporcionando uma forma eficaz e eficiente de melhorar a produtividade, a qualidade do software e o tempo de lançamento para o mercado. Esses benefícios combinados com a necessidade da maioria dos aplicativos precisarem interagir com outras aplicações e o acesso à Internet tornam essas aplicações vulneráveis a muitas ameaças. Para a maioria das empresas orientadas à produto, os requisitos de segurança podem variar assim como outro atributo de qualidade do software. Assim, é importante fornecer variantes do mesmo produto para satisfazer diferentes necessidades. Devido às suas capacidades de gerenciamento de variabilidade, arquiteturas de linha de produtos têm a capacidade de satisfazer esses requisitos, se cuidadosamente projetada o sistema resultante terá uma melhor chance de satisfazer as expectativas. Todos esses requisitos devem ser alcançados nas primeiras fases do projeto, caso contrário, o custo para projetar uma arquitetura segura aumentará, o que poderia piorar no contexto SPL, devido à sua natureza complexa. Assim, para melhor apoiar os arquitetos durante as decisões de projeto. Uma família de experimentos utilizando três SPLs distintas foram utilizadas para analisar diferentes técnicas de segurança, implementadas usando compilação condicional (CC) e programação orientada a aspectos (AOP). Essa avaliação teve como objetivo analisar as técnicas e mecanismos em relação a: tamanho, “separation of concerns”, coesão e acoplamento. O resultado nos mostra que para a maioria das técnicas quando implementadas com compilação condicional apresentavam uma menor quantidade de código quando comparadas com AOP. O atributo de “separation of concerns” teve menor impacto na manutenção quando implementado com programação orientada a aspectos. A análise também mostrou que técnicas de detecção de ataque são menos onerosas do que técnicas para resistir a ataque. Os resultados são úteis para pesquisadores e profissionais. Por um lado, os pesquisadores podem identificar direções de pesquisa e obter orientação sobre como as técnicas de segurança impactam na manutenção. Por outro lado, os profissionais podem se beneficiar deste estudo, identificando o mecanismo de implementação da variabilidade menos dispendioso, bem como aprendendo técnicas concretas para implementar táticas de segurança a nível de código.

Palavra-Chave: Linha de produto de software. Segurança de software. Vari-

abilidade. Táticas de segurança. Técnicas de segurança. Compilação condicional. Programação orientada a aspectos.

Agradecimentos

When you begin to write the acknowledgments of your thesis is when realizing the size of the journey faced and the number of people who have met in your life. Doubts and certainties are constant in this period, as well as, the sure that you have to go to the end. Certainty and Strength guaranteed by the people who have supported you throughout this time.

Everything gets easier when you have:

I would like to especially thank my co-adviser Eduardo Almeida, for the support my decisions during my academic life and for always challenging myself with new barriers and uncertainties to be overcome. I also would like to thank you, my advisor, Vinicius Cardoso who provided me with this opportunity and all support with academic and personal decisions. I really admire these guys and thank you for changing my life and for the teachings.

Thank you, my brothers, Yguarata Cavalcanti and Padraig O'Leary, by the teachings, making difficult moments easier and enjoyable, and giving me the feeling that I have gained two brothers for life. I would also to thank you Leandro Marques and Vanilson Buregio for all the advice and strength I have received from them. Last but not least, my youngest bearded brother, Bruno Melo.

Thank you my research partners from Reuse in Software Engineering group (RiSE) for the contributions and inquiries at all meetings and discussions. Also, I would like to thank you for the valuable moments researching, organizing scientific events and interacting with the best researchers in the area. Be sure that you are essential in my academic life.

Thank you to those who made possible all my achievements: my mother, my father, my sister, my grandmother, uncles, aunts, and cousins. I also would like to thank you, Giselle Pinho, for the support and patience that were necessary along this journey. Finally, I would like to thank you for my inspiration which is no longer present in our lives, but which was and will be essential for all my achievements. I will never forget your love and lessons, thank you so much, my grandfather.

"Você não sabe o quanto eu caminhei
Pra chegar até aqui"(Cidade Negra)

Lista de tabelas

Sumário

1 INTRODUCTION

Nowadays, the modern world is getting harder and harder to live without software, since it is a key element in many devices and systems. The manufacture and industrial are managed by software, as well as the financial systems. They are also used in the entertainment area, including the music, cinema and games industry (SOMMERVILLE, 2010). This way, software control from small gadgets to the largest civil aircraft build so far (BURGER; HUMMEL; HEINISCH, 2013). Thus, they are becoming even more complex, which makes researchers investigate for more cost-effective methods, to cope with tight deadlines and market pressure.

The Software Product Lines (SPL) approach has proved to be an efficient and effective strategy to achieve economies of scope and results such as substantial cost savings, reduction of time to market, and large productivity gains (CLEMENTS; NORTHROP, 2001). The SPL engineering explores the commonalities and manages variabilities among derived products, allowing the establishment of a common platform (reference architecture) on top of which assets can be reused and assembled into different products, meeting particular customers and market needs (POHL; BOCKLE; LINDEN, 2005).

In order to achieve all of these benefits, a reference architecture is defined to represent how the requirements, including variability, are implemented. It is a complex task since it should have the ability to select and configure reusable software artifacts to enable different combinations, allowing the derivation of different products (POHL; BOCKLE; LINDEN, 2005). This architecture is designed with various quality attributes (such as security, performance, maintainability deployment and availability) and requirements in mind. Security is a quality attribute of software systems that must be addressed by most of the architectures. It is a cross-cutting concern that is affected by a wide range of architectural decisions (FAEGRI; HALLSTEINSEN, 2006). If the requirements and design do not encompass security needs, problems may arise, especially if this system is manipulated by different users in different places. If these considerations are addressed during the architecture design, a secure structure at a reasonable price is achieved. On the other hand, a more expensive structure will be designed, since latter decisions in design entail costly changes (TAYLOR; MEDVIDOVIC; DASHOFY, 2009).

After all, architecture design is all about making sound trade-offs. Make these trade-offs effectively becomes even more important in a context where a company wants to deliver multiple product variants to the market (FAEGRI; HALLSTEINSEN, 2006). As the architectural design is a knowledge intensive task that depends heavily upon experience to encode and reuse this knowledge, the software architecture community has created the concepts of architectural tactics and architectural patterns (FAEGRI;

HALLSTEINSEN, 2006). The most suitable known software architecture approach for building secure systems is based on tactics, which is basically a design decision (FERNANDEZ; ASTUDILLO; PEDREZA-GARCIA, 2015; RYOO; KAZMAN; ANAND, 2015; CERVANTES et al., 2016). Although the tactics are considered an important approach to improve system quality factors, there is a lack of studies regarding how to implement these design decisions, which can lead to misunderstandings (FERNANDEZ; ASTUDILLO; PEDREZA-GARCIA, 2015). In SPL context, the architectures need to be stable and flexible enough to support the most varied types of product derivation, as well as, facilitating its management.

This work investigates the impact on maintainability of different security techniques implemented by Aspect-oriented programming and Conditional compilation. These variability implementation mechanisms were chosen since it allows the developer to map variability concerns into core assets represented as features, during compile time, allowing products to be derived only with the necessary code. Thus, different security tactics were implemented at the code level, in order to understand how each one influences the code size, separation of concerns, coupling, and cohesion. We hypothesize that by generating a body of knowledge, it is possible to better support software architects to make more conscious design decisions, regarding to which is *the most suitable implementation variability mechanism to implement software security in SPL project*. The study can provide software architects with a means to achieve a more balanced and secure reference architecture, by avoiding misunderstandings, trade-offs and improving their ability to make design decisions.

1.1 Motivation

SPL products are distinguished in terms of features, i.e., end-user visible characteristics of products (KANG et al., 1990). Based on the selection of features, stakeholders can derive tailor-made products satisfying a range of functional (FPs) and nonfunctional properties (NFPs). Functional properties implement the tasks/functionalities of a system. On the other hand, nonfunctional properties are those that impose special conditions and qualities on the system (LOHMANN et al., 2005), usually observable by end-users. For instance, if a software system runs slower than expected, users may not be interested in using it, regardless the provided functionalities. Thus, functional and nonfunctional aspects should be synchronized for a product to be viable in the market.

Nonfunctional properties play important role in SPL engineering. In the SPL literature, these properties can also be referred to as *quality attributes* (ZHANG; JARZABEK; YANG, 2003; BASS; CLEMENTS; KAZMAN, 2012), *nonfunctional requirements* (AOYAMA; YOSHINO, 2008), *extra-functional properties* (BENAVIDES; TRINIDAD; RUIZ-CORTÉS, 2005) and *softgoals* (NGUYEN, 2009). In this work, we will interchan-

geably mention quality attributes (QAs) and non functional properties.

The variability management challenges and proposed solutions are being investigated for 20 years, focusing on the variability of functional properties, methods, techniques, and tools (CHEN; BABAR, 2011). However, nonfunctional properties have not received much attention in the context of variability (GALSTER et al., 2014). Most of the approaches have spent some effort trying to understand how products in an SPL differentiate from each other from the functional point of view, and less attention is given to understand these differences regarding to nonfunctional properties.

Depending on the customers and market segments, different needs related to quality attributes may arise. For example, a user-adaptive system which takes individual characteristics of their users into account and adapts their behavior accordingly requires more stringent user data reliability and security than desktop applications. To address this customer or market need, two approaches can be used. The first one builds product variants with a common quality attribute level (FAEGRI; HALLSTEINSEN, 2006), and the second one builds products with purposefully different levels. All these characteristics should be achieved at early design phases, otherwise, the cost to design a secure architecture will increase, which could worsen in SPL context, due to its complex nature (TAYLOR; MEDVIDOVIC; DASHOFY, 2009).

In addition, there are some aspects which make nonfunctional properties more challenging than functional properties. Firstly, some nonfunctional properties are continuous and nonlinear (REGNELL; SVENSSON; OLSSON, 2008), which means that instead of viewing the quality level as a binary property of "in"or "out"as functional requirements behave, the quality levels have different shades and sliding scales. This characteristic allows the derivation of a product which satisfies more than one customer, with an intermediate nonfunctional property level. For example, if customer A requires a low level of security and customer B requires a medium level of security, a product with medium level will satisfy both. However, it is important to mention that, there has to be a good motivation to purposefully vary nonfunctional properties levels in SPL, due to its high investments.

Secondly, it may be difficult to the customer to understand the differences between the products and select a variant that matches their needs, when this choice involves quality attributes. In many domains, the quality attributes are described in imprecise and vague terms. This way, it is worthwhile to know how the quality attribute differences in a product variant can be distinguished to the customers. Using the aforementioned example, it cannot be clear to the customer which is the real difference between low and medium security. According to (GALSTER, 2015), it is important to conduct exploratory and descriptive studies to better understand variability in quality attributes and create approaches for describing, analyzing and implementing variability

in quality attributes.

Thirdly, some functional properties (i.e., manipulate complex graphical images) might be inherently complex, making a nonfunctional property (i.e. performance) very difficult to achieve. But, what is possible is that for any of these functions the architect's choices will determine the relative level of quality. Some architectural choices will lead to higher performance; some will lead in the other direction (BASS; CLEMENTS; KAZMAN, 2012). In addition, a simple quality attribute may affect many assets in an SPL architecture, which makes the cost of the nonfunctional property prohibitive (HALLSTEINSEN et al., 2006).

Fourthly, in order to implement SPL variability a variety of mechanisms can be used (GACEK; ANASTASOPOULES, 2001; BOSCH; CAPILLA, 2013). In this work, two mechanisms were considered due to its large application in industry and previous studies (FIGUEIREDO et al., 2008; GAIA et al., 2014; FERREIRA et al., 2014): conditional compilation (SCHULZE et al., 2013) and aspect oriented programming, due to its natural purpose (KICZALES et al., 1997). Both implementations can provide a good understanding regarding to which is the most suitable mechanism to implement security at SPL code level.

Fifth, according to (GALSTER et al., 2014) with regard to quality attributes, performance is addressed most, whereas quality attributes such as security or safety are rarely a concern of variability handling approaches.

Finally, a well known approach to design different quality attributes at architectural level is the use of tactics (BASS; CLEMENTS; KAZMAN, 2012; MIRAKHORLI; MäDER; CLELAND-HUANG, 2012). Although well known, few studies were performed in the SPL context regarding how to implement these tactics at code level (MIRAKHORLI; MäDER; CLELAND-HUANG, 2012; FERNANDEZ; ASTUDILLO; PEDREZA-GARCIA, 2015), which could cause misunderstandings during its implementation.

In summary, based on the requirement or architectural decisions made to use a quality attribute tactic, the developer should find a way to concretize the tactic in code level (MIRAKHORLI; MäDER; CLELAND-HUANG, 2012). Unfortunately, the variability points found in individual tactics can make this a challenging task. In addition, secure coding practices are expensive, requiring not only careful attention to the code, but also extensive testing, inspections, and scanning (CERVANTES et al., 2016). In this context, our work aims to avoid misunderstandings regarding to how to code software security and do not degenerate code modularity and minimize costs on the code maintenance. In addition, it also focus on how the security tactics should be implemented at code level, by evaluating different mechanisms to implement it.

1.2 Problem Statement

The software security is a problem with different perspectives which states from logical (coding/design), operation and policies areas. However, it is not considered throughout software development life-cycle (RYOO; KAZMAN; ANAND, 2015). Frequently, software architects and developers focus on functional properties and software security is developed to "close a hole"after the software development. The code is sent to the security responsible without any previous analysis or impact about how to implement security. This generates in most cases inefficient solutions that make the software vulnerable to different types of attack. In addition, the secure coding practices are expensive since it requires extensive code inspection and testing. For this reason, we assess different ways to implement security and bring important insights to be considered in early stages of development.

Due to the diversity of quality attributes and their intrinsic characteristics, it is challenging to propose constructs that are applicable to all quality attribute similarly. Therefore, this work has made an explicit decision to focus on the security quality attribute in more detail.

In this context, it is important to consider the security quality attribute since its initial phases of SPL life-cycle, due to the high complexity inherent to SPL architecture nature. The sooner the definition of how security will be carried out, the less costly will be its implementation, testing, and maintenance. In addition, security cannot be directly tested, instead verification of system security can be performed by testing confidentiality, auditability, and vulnerability (RIBEIRO; TRAVASSOS, 2016). Recent work in this area focuses on requirements and design phases (FAEGRI; HALLSTEINSEN, 2006; D.; M.; C., 2006; MELLADO; FERNÁNDEZ-MEDINA; PIATTINI, 2009; BASS; CLEMENTS; KAZMAN, 2012; MYLLäRNIEMI; RAATIKAINEN; MäNNISTö, 2015) without explaining how security will be implemented at code level, neither how this behave regarding to maintainability (ARVANITOU et al., 2017). It is also important to reduce the gap between security requirements/design and how it can be implemented, reducing the chances of wrong interpretations on their implementation (FERNANDEZ; ASTUDILLO; PEDREZA-GARCIA, 2015). Although it was being studied for the single system it is interesting to evaluate the software product line context to understand how the variability impacts/behaves with security implementation.

Encouraged by the motivations presented in the previous section and problem statement, the goal of this work can be stated as follows:

- *It provides evidence regarding different ways to implement security techniques by comparing maintainability throughout several internal attributes, such as: code size, separation of concerns, cohesion, and coupling.*

- *To understand the impact on maintainability by using different variability implementation mechanism to implement security techniques. It is achieved by comparing aspect-oriented programming and conditional compilation implementation and analyzing their impact on code size, separation of concerns, coupling, and cohesion.*
- *In addition, to be considered an initial attempt to understand the behavior of security techniques in code, it also reduces the gap between security requirements/design and how it can be implemented, reducing the chances of wrong interpretations and supporting design decisions.*
- *It also provides three SPL Test Beds considering different domains, without the interference of any SPL framework in its implementation, which makes easy to maintain and be explored by nonexperts.*

In summary, all aforementioned goals will provide the base to support architecture design decisions which consider the different maintainability internal attributes, such as: code size, separation of concerns, coupling and cohesion applied to security techniques implementations using conditional compilation and aspect oriented programming as variability mechanisms.

1.3 Research Design

This section describes the research design used as the basis for this work. Considering the research objectives and problem statement earlier described in this chapter, we intend to conduct quantitative studies to gain further understanding of the research problem and enrich our conclusions. Triangulation is important to increase the precision of empirical research, taking different angles towards the studied object and thus providing a broader picture (RUNESON; HöST, 2009). The studies replication and their data triangulation enhances the conclusions and completeness of the overall study, bringing more credibility to the research findings (RUNESON; HöST, 2009).

This research is organized in four successive stages: **1. Background**, **2. Infrastructure Creation**, **3. Evaluation** and **Proposal**. Figure 1 shows a diagram with these stages and an overview of their activities.

1st Stage. Background

The background of this research was built based on relevant topics from the Software Engineering area. The studied topics included Software Product Lines (SPL), its essential activities, the importance, and characteristics of a reference architecture and examples of how to deal with quality attributes and its peculiarities. In addition, it

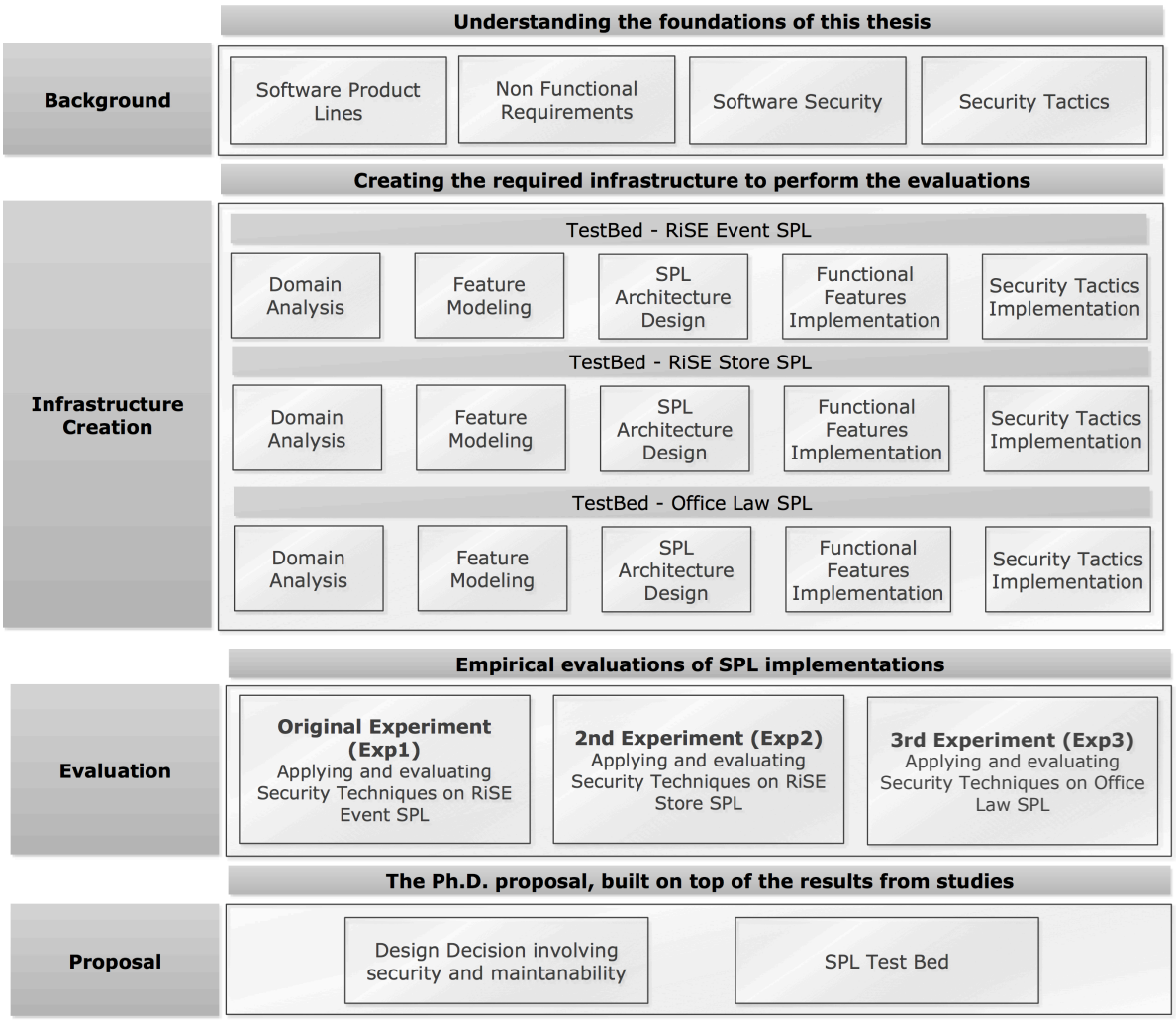

Figura 1 – Research Design

also describes characteristics which make software safe and how it could be achieved at architectural level.

2nd Stage. Infrastructure Creation

In this stage, all the environment to perform our study was developed. Thus, three SPLs were built considering different domains and stakeholders by applying the following activities: domain analysis, feature modeling, SPL Architecture design, functional features implementation, security tactics implementation and testing. We chose to develop this environment due to the following motivations: *(i)* To implement each security technique in each SPL; *(ii)* Most of the available SPLs were built with support of different SPL frameworks which hinders the extraction of the metrics and the maintenance by other people who do not know the frameworks; *(iii)* Most of the security techniques have their code spread over user interface classes, this way all SPLs need to use the same user interface in order to allow comparison; *(iv)* The variability mechanism can only be implemented in `.java` files which make it impossible to use another type

of graphical interface, such as: `html` ; *(v)* It is important to use real size SPLs in order to increase the study reliability and generalization.

3rd Stage. Evaluation

Based on the literature review, conditional compilation and aspect-oriented programming were used to implement different security techniques on all three SPL test beds. Moreover, a family containing three experiments were performed in order to evaluate how these security techniques impacts on SPL maintainability considering conditional compilation and aspect-oriented programming.

4th Stage. Proposal

Based on the findings identified during all previously performed stages, several design decisions were collected. It will guide software architects who want to implement security in a product line. Based on this design decisions, architects can select the most suitable way to implement security considering its impact on code maintainability.

1.4 Contributions

As result of the work presented, the following contributions can be highlighted:

- *Empirical Evidence:* it presents empirical evidence in the context of the implementation of security in Software Product Lines architectures, such as:
  - It provides empirical evidence regarding to security implementation. As discussed by (MYLLäRNIEMI; RAATIKAINEN; MäNNISTö, 2012), empirical evidence is needed regarding to quality attributes, such as: why and which quality attributes vary, what their variants are, and how they are realized in the architecture.
  - It presents empirical evidence comparing maintainability attributes such as code size, separation of concerns, cohesion and coupling among the implementation of different security techniques, as well as classifying them according to its impact, since depending on the application, one is more convenient or acceptable than other.
  - It also considered during security techniques implementations two different variability implementation mechanisms: Conditional compilation and Aspect oriented programming. The importance of our study is also reinforced by (FIGUEIREDO et al., 2008) and (CARVALHO et al., 2016), which analyzed the impact of changes in functional properties using different implementation variability mechanisms.

  - It considered the gap between the proposed security approaches and its implementation at code level (FERNANDEZ; ASTUDILLO; PEDREZA-GARCIA, 2015) in order to reduce the chances of wrong interpretations on their implementation, as well as, support software architects during design decisions. It is important to provide architects with more detailed guidance based on several concrete examples of complete implementation using patterns and techniques about how to coding each of the security tactics.
  - It provides important information regarding to indicators to metrics and further define their baselines definition for size, separation of concerns, cohesion, and coupling in the context of security in software product lines implementations.

- *Test Bed:* It contributes by developing a platform for conducting rigorous, transparent and replicable studies on Software Product Lines Engineering.

1.5 Out of Scope

The following topics are considered out of the scope of this thesis:

- *Analysis of other quality attributes:* This work considered security aspects in SPL architectures. Other important quality attributes such as performance, availability, and testability are not part of the scope of this work;
- *Analysis of Trade-offs:* Although we have plans to evaluate the impact of different quality attributes at architecture level considering modularity, complexity, and scattering, it is out of scope of this work to analyze the existing trade-offs among performance, availability and testability;
- *Tool Support:* Although we have used some tools (CIDE [1], Eclemma [2]) and identified some extension points and improvements, it is out of scope the extension of such tools and the development of a new tool for supporting the tasks performed;
- *Database Derivation:* During product derivation, the code related to each selected feature is instantiated in order to assemble the product. This feature selection could reflect changes on the database relational model. It is out of scope to understand how the Database should be modeled to fit with specificities of the products derived from the SPL.
- *Recover from Attack Tactics:* Although we are investigating the impact on the implementation of different security tactics, an analysis considering SPLs domains was performed which makes some tactics not applicable to our context.

[1] <http://wwwiti.cs.uni-magdeburg.de/iti_db/research/cide/>
[2] <http://www.eclemma.org/>

Besides, the recover from attack tactics are part of another quality attribute called availability (BASS; CLEMENTS; KAZMAN, 2012).

1.6 Organization of the Thesis

The remainder of this work is organized as follows:

- Chapter 2. This chapter provides the necessary background to understand the key concepts related to this work. It discusses the software product line basic concepts and essential activities, besides how the variability is managed throughout the Software Product Line life-cycle. In addition, it also provides information regarding to reference architecture, quality attributes and different SPL adoption strategies.
- Chapter 3. It presents software security concepts, a known taxonomy, as well as, security tactics which describe several techniques to achieve software security at architectural level. At the end of this chapter, some studies regarding how to represent security quality attribute as SPL variability are described.
- Chapter 4. It describes three test beds created using different stakeholders and domains: store management, event management, and office law management. They were developed using JAVA language without any support of a specific SPL development framework. It can be used as basis to the security implementations and evaluations.
- Chapter 5. It describes the definition, planning, operation, analysis and interpretation and packaging of a quantitative analysis from a family of experiments in the context of the implementation of different security tactics. It analysis different techniques to implement security considering maintainability internal attributes: size, separation of concerns, cohesion, and coupling.
- Chapter 6. It concludes the work by summarizing the findings and proposing future enhancements to the solution, discussing next steps of this work.

2 SOFTWARE PRODUCT LINES: AN OVERVIEW

2.1 Introduction

The concept of software reuse started to be used since 1949, in which the first subroutine library was proposed (TRACZ, 1988). It gained importance in 1968, during the NATO Software Engineering Conference, considered the birthplace of the field. It focused on the software crisis - the problem of building large, reliable software systems in a controlled, cost-effective way. Firstly, software reuse was pointed as being the solution to the software crisis. McIlroy's paper entitled "Mass Produced Software Components" (MCILROY, 1968), ended up being the seminal paper in the software reuse area. In his words: "the software industry is weakly founded and one aspect of this weakness is the absence of a software component sub-industry", it was the basis to consider and investigate mass-customization in software (ALMEIDA, 2007).

The mass-customization idea was born in 1908, in the automobiles domain, when Henry Ford the father of assembly-line automation, built the Model T based on interchangeable parts. It enables the production for mass market more cheaply than individual product creation. However, the production line reduced the products diversification. Although some customers were satisfied with standardized mass products, not all people wanted the same kind of car for any purpose. Hence, the industry was faced with a growth interest for individualized products. However, mass customization is a "coin" with two distinct faces. In the customer's face, mass customization means the ability to have an individualized product, based on specific needs. For the company, however, it means technological investments, which leads to higher product's prices and/or lower profit margins for the company (POHL; BOCKLE; LINDEN, 2005).

Considering the software context, two types of software can be observed: *(i)* traditional software engineering usually focused on building individual software systems, one system at a time (SOMMERVILLE, 2010) and *(ii)* modern software development practices which have support for *variability* and *mass customization*. While the first is more expensive to develop since each product is treated as an individual unit and developed from scratch, the designed assets are not variable enough to be reusable for different products. The second suffers a lack of diversification.

To avoid higher prices for individualized products and lower profit margins, some companies introduced the common platform concept for their different types of products, planning beforehand which parts will be further instantiated in different product types. A systematic combination between mass-customization and platform-based development allows us to reuse a common base of technology and, at the same time, to develop

products in close accordance with customer needs. In this context, arises the concept of Software Product Lines (SPL) as being“ *a set of software-intensive systems sharing a common, managed set of features that satisfy 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, 2001). Examples of SPL are manifold and can be found in different successful stories about its application of SPL engineering in their development process, as in the Product Line Hall of Fame [1].

In order to achieve the potential benefits of SPL engineering, it is crucial to understand and control common and distinguishing characteristics between the systems that are part of the product line. This way, SPL engineering was founded in three main activities: *(i)* Core Asset Development (Domain Engineering), *(ii)* Product Development (Application Engineering) and *(iii)* Management. These activities are subject to discussion in the following sections: Section 2.2 introduces the software product lines essential activities. Some software product line adoption strategies are described in Section 2.3 and Section 2.4 describes the variability management ideas. The SPL product line architecture and its characteristics are described in Section 2.5. Next, Section 2.6 presents a set of software quality attributes and Section 2.7 summarizes the chapter.

2.2 SPL Essential Activities

Software Product Lines combine three essential and highly iterative activities that blend business practices and technology. Firstly, the *Core Asset Development (CAD)* activity that does not directly aim at developing a product, but rather aims to develop assets to be further reused in other activities. Secondly, *Product Development (PD)* activity which takes advantage of existing reusable assets to build different products. Finally, *Management* activity, which includes technical and organizational management (LINDEN; SCHMID; ROMMES, 2007). Figure 2 shows this triad of essential activities.

2.2.1 Core Asset Development

Core Asset Development comprehends the necessary activities (POHL; BOCKLE; LINDEN, 2005) to: *(i)* define the variability and commonality of the SPL; *(ii)* determine the set of product line planned members; *(iii)* specify and develop reusable artifacts to accomplish the desired variability and to be further instantiated in product line members.

It is an iterative activity that influences the way in which the core assets are produced (see Figure 3). To do so, some inputs are required (NORTHROP, 2002), such as: *Product Constraints:* responsible for identifying commonalities and variations

[1] http://splc.net/fame.html

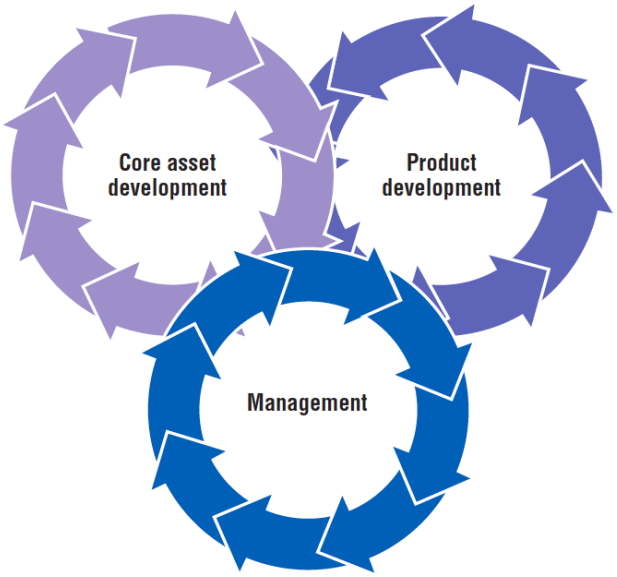

Figura 2 – Essential product line activities (NORTHROP, 2002).

among members that will constitute the product line, including the behavioral features, the standards they must follow, performance limits and quality requirements imposed on them. *Styles, patterns, and frameworks:* it comprehends the existing architectural build blocks that can be applied to meet the product an production constraints. *Product Constraints:* it describes the commercial, military or company-specific standards that must be applied to the products in the SPL, specifying the infrastructure on which the products must be built, when the product will be brought to market, what are the components that could/should be reused. *Production Strategy:* is the overall approach to create the core assets, it should decide if the product line will be built by starting with a set of core assets and spinning products off or starting with a set of product and generalizing their components to produce product line assets. *Inventory of preexisting assets* software and organizational assets (architecture pieces, components, libraries, frameworks and so on) available at the outset of the product line effort that can be included in the asset base.

Those aforementioned inputs provide necessary information for each of the five disciplines that composes the CAD, they are: *(i)* domain requirements, *(ii)* domain design, *(iii)* domain realization (implementation), *(iv)* domain testing and *(v)* evolution management, all of them administered by the management activity (POHL; BOCKLE; LINDEN, 2005). These disciplines are responsible for creating the core assets, as well as, the following outputs (Figure 3) (CLEMENTS; NORTHROP, 2001): *Product line scope* the description of the products derived from the product line or that the product line is capable of including. The scope should be small enough to accommodate future growth and big enough to accommodate the variability. *Core assets* comprehend the basis for production of products in the product line, besides the reference architecture,

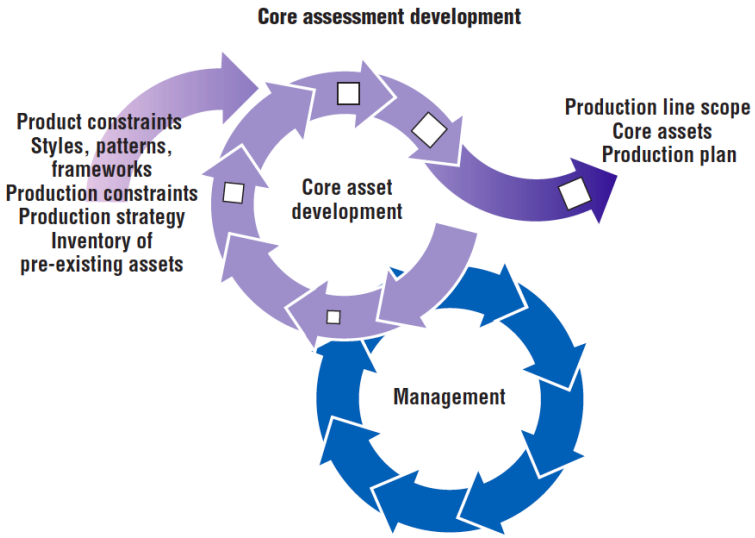

Figura 3 – Core Asset Development (NORTHROP, 2002).

that will satisfy the needs of the product line by admitting a set of variation points required to support the spectrum of products, these assets can also be components and their documentation. The *Production plan* describes how the products are produced from core assets, it also describes how specific tools are to be applied in order to use, tailor and evolve the core assets.

2.2.2 Product Development

The product development main goal is to create individual (customized) products by reusing the core assets previously developed. The CAD outputs (product line scope, core assets, and production plan), in conjunction with the requirements for individual products, are the main inputs for PD activity (Figure 4).

In possession of the production plan, which details how the core assets will be used in order to build a product, the software engineer can assemble the product line members. The product requirements are also important to create a product. Product engineers have also the responsibility to provide feedback on any problem or deficiency encountered in the core assets. It is crucial to avoid the product line decay and keep the core asset base healthy.

2.2.3 Management

The management of both technical and organizational levels are extremely important to the software product line effort. The former supervises the CAD and PD activities by certifying that both groups that build core assets and products are engaged in the activities and to follow the process, the latter must make sure that

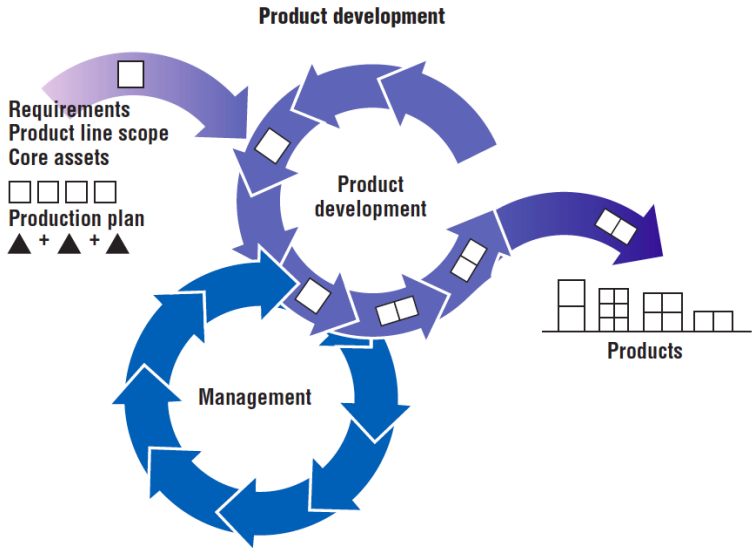

Figura 4 – Product Development (NORTHROP, 2002).

the organizational units receive the right and enough resources. It is, many times, responsible for the production strategy and the success or failure of the product line.

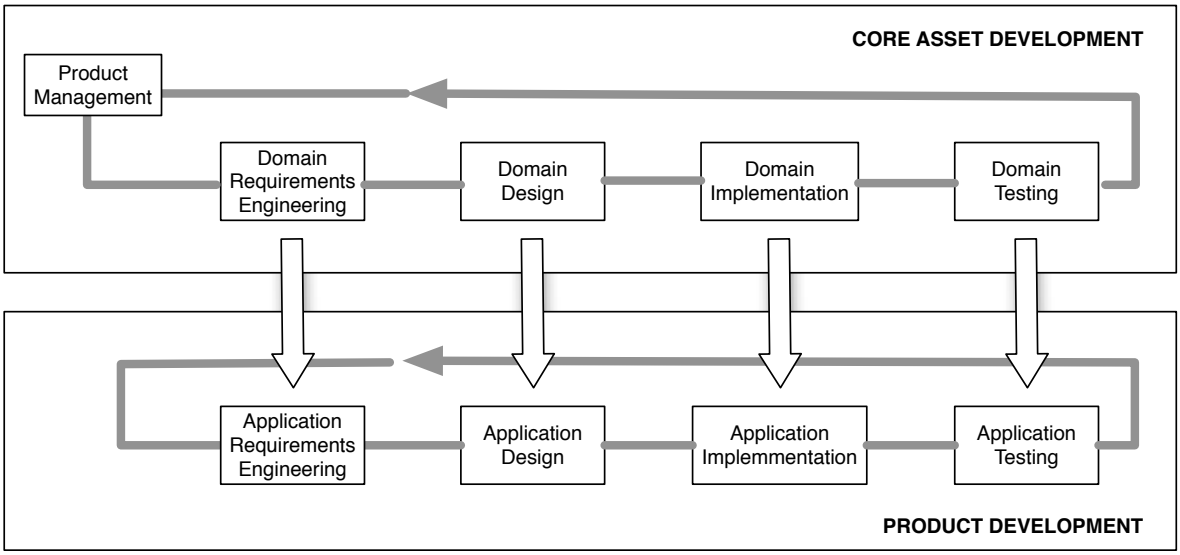

Figura 5 – SPL Engineering Framework, adapted from (POHL; BOCKLE; LINDEN, 2005).

Figure 5 shows the SPL engineering framework, originally introduced by (POHL; BOCKLE; LINDEN, 2005), which has an insider view regarding to each SPL essential activities by encompassing each activity and their associated life-cycle disciplines.

2.3 SPL Adoption Strategies

With the growth of competitiveness, the companies are increasingly seeking for a way to improve development time, quality and decrease the time-to-market. This

scenario makes SPL methodology invited to theses companies. In order to introduce SPL methodology, a company should consider its business goals, the adoption strategy to be considered and their pros and cons to the organization (POHL; BOCKLE; LINDEN, 2005). Besides, the authors in (BASTOS et al., 2011), also point organizational structure, technology barriers and organization maturity level as aspects that should be considered during SPL adoption.

In (POHL; BOCKLE; LINDEN, 2005), four transition strategies are described, as follows: **(i) Incremental Introduction**. It starts small and expands incrementally, it may occur in two ways, *expanding organizational scope* which starts with a single group doing SPL engineering and other groups are added incrementally after the first group succeeds and *expanding investment* which starts with a small investment that is incrementally increased, depending on the achieved success. **(ii) Tactical Approach** starts introducing partially SPL concepts in sub-process and methods, starting form the most problematic sub-process. It is often used when architects and engineers drive this introduction. **(iii) Pilot Project Strategy**, this strategy may be started using one of the several alternative ways, such as, starting as a potential first product, starting as a toy product, starting as a product prototyping. **(iv) Big Bang Strategy**, the SPL adoption is done by the organization at once. The core assets and the platform are built, after that, the product development starts and the products are derived from the platform.

Another point of view is presented by (KRUEGER, 2002a), which advocates three adoption models: using the *proactive approach*, the organization analyzes, designs and implements the overall SPL to support the full scope of products needed on the foreseeable horizon. In the *reactive approach*, the organization incrementally grows their SPL as the demand arises for new products or new requirements on existing products. Finally, using *extractive approach*, the organization capitalizes on existing custom software systems by extracting the common and varying source code into a single production line.

Nevertheless, the strategies are not necessarily mutually exclusive. It is possible to startup using the extractive approach, by applying refactoring in existing software systems (ALVES et al., 2006a), and then move on to a reactive approach to incrementally evolve the SPL over time (KRUEGER, 2002a).

Independent from the approach used, there are common barriers to be overtaken, and they may affect different levels into a company (BASTOS et al., 2011), such as: *Project and Customers View:* Initial associated cost and time to devote to product line activities. *Organization and Development groups view:* Lack of product line vision, lack of documentation, an absence of an explicitly defined development process, inadequate organizational structure, and so on. *SPL Engineering Community and Resources view:* Lack of tool support, lack of management maturity, lack of SPL experts and a high cost

of training, etc.

2.4 SPL Variability Management

During Core Asset Development, variability is introduced in all domain engineering artifacts (requirements, architecture, components, test cases, etc.), to be exploited during Product Development to derive applications tailored to the specific needs of different customers. According to (SVAHNBERG; GURP; BOSCH, 2005), variability is defined as *"the ability of a software system or artifact to be efficiently extended, changed, customized or configured for use in a particular context"*. It is described through variation points and variants. While, the variation point is the representation of a variability subject (variable item of the real world or a variable property of such an item) within the core assets, enriched by contextual information; the variant is the representation of the variability object (a particular instance of a variability subject) within the core assets (POHL; BOCKLE; LINDEN, 2005).

In the SPL development life-cycle, the assets go through different disciplines. Each of them has its own way to represent variability, and the development consists of transformations of these representations (GURP; BOSCH; SVAHNBERG, 2001). For example, firstly a domain analysis is performed to identify the existing commonality and variability. Based on that, SPL requirements are specified and can be transformed into a feature model. Next, the feature model and requirements provide an important basis to design the SPL architecture. Once the architecture is documented (e.g. styles, patterns), it will drive the source code development. This code is then compiled, linked and finally run. This way, the variability contained in the source code must reflect the variability early defined in each previous artifacts.

The variability management involves issues, such as: variability identification and representation, variability binding and control (POHL; BOCKLE; LINDEN, 2005). They are following described.

2.4.1 Variability Identification and Representation

Documenting and managing variability is one key property to characterize software product line engineering. The explicit definition and management of variability distinguish software product line engineering from both, single-system development and software reuse (POHL; BOCKLE; LINDEN, 2005).

Three questions are helpful to variability identification, *what vary?* the variability subject, *why does it vary?* the drivers of the variability need, such as stakeholder needs, technical reasons, market pressures, etc. The later, *how does it vary?* the possibilities of variation, also known as variability objects.

Once the variabilities are identified, they need to be modeled. In order to achieve it, several approaches have been proposed (KANG et al., 1990; CZARNECKI; EISENECKER, 2000; GOMAA; SHIN, 2002; DASHOFY; HOEK; TAYLOR, 2002; SCHMID; JOHN, 2004; GOMAA, 2004; POHL; BOCKLE; LINDEN, 2005; DHUNGANA; GRüNBACHER; RABISER, 2011; CHEN; BABAR, 2011; BERGER et al., 2015). Among them, the *feature modeling* (KANG et al., 1990) is perhaps the most popular technique (BERGER et al., 2015). It is described by using the feature model, which is a hierarchical way to organize commonalities and variabilities into levels of increasing detail, as well as their relationship. According to (LISBOA et al., 2011), there are four possible types of features (see Figure 6), such as: *(i) Mandatory:* the feature will always be in the product; *(ii) Optional:* the feature may or not be present in the product; *(iii) Alternative:* From a set of feature, one and only one of them will be present in the product and *(iv) Or:* From a set of features, at least one feature of them be present in the product. Additionally, the feature model also shows the dependencies between features. These dependencies are: *(v) Implication:* obligates the inclusion of destination feature in the product, whenever the first is selected and *(vi) Exclusion:* Means that it is not possible to have both features in the relationship in the same product.

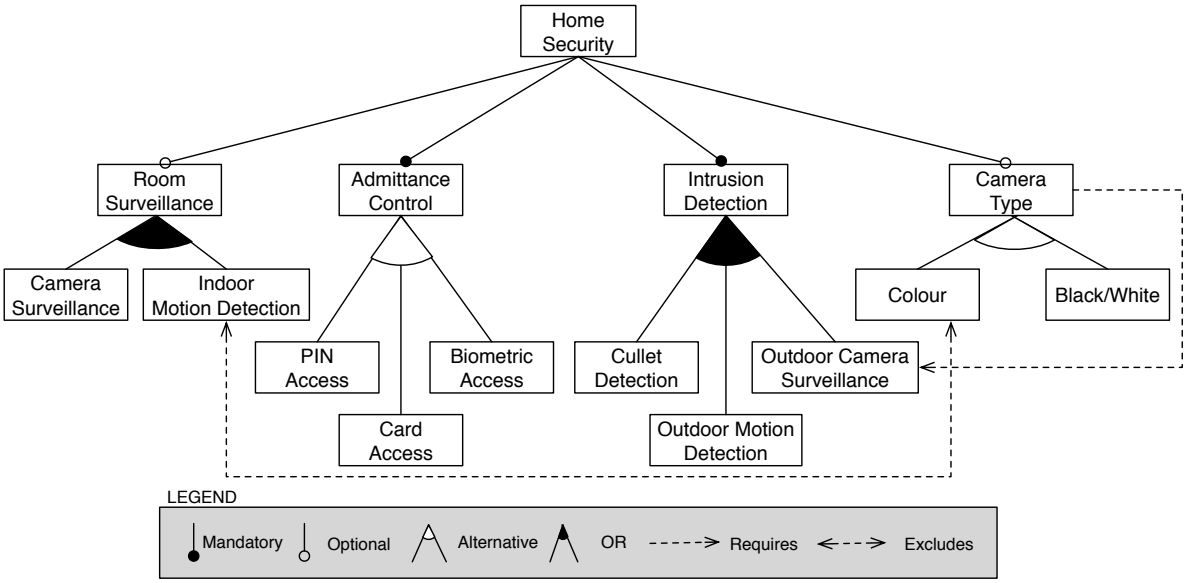

Figura 6 – Sample Feature Model.

Figure 6 shows a model in which each of those types of features and dependencies are illustrated. It uses a widely used notation (CZARNECKI; EISENECKER, 2000; KANG et al., 1990) to present the feature tree for a home security system. Besides, the type of features and relationships described previously it also shows the parent»child relations.

The root feature describes the application domain under analysis, which has two

mandatory features (*Admittance Control and Intrusion Detection*), two optional features (*Room Surveillance and Camera Type*), also known as variation point, two OR-group features (variation point) from which one or more (variant) features may be selected to compose an SPL member, and two alternative group of features, from which only one feature must be selected. Representing the *requires* constraint dependency, whenever the feature *Camera Type* is selected, the feature *Outdoor Motion Surveillance* must be selected as well. On the other hand, whenever the feature *Indoor Motion Detection* is selected, the feature *Camera Type»Colour* must not be selected, and vice-versa.

From this feature model, a set of valid configurations can be generated. A valid configuration is one in which all rules are satisfied, or, a valid subtree that satisfies all cross-tree dependencies is instantiated.

2.4.2 Variability Binding and Control

After all modeling phases, it is important to know how to deal with SPL product line assets genericity at code level. A feature implementation is usually spread across many source files and modules, for this reason, the relation between features and variabilities (variation points) is 1:N in many cases, except for Aspect-oriented Programming(AOP) which isolate a crosscutting concerns in one aspect (GACEK; ANASTASOPOULES, 2001).

Additional assistance in handling variability at the code level can be provided if the exact binding time of variability is known. The variability binding indicates the life-cycle milestone that the variants related with a variation point will be realized. According to (GACEK; ANASTASOPOULES, 2001; SVAHNBERG; GURP; BOSCH, 2005; BOSCH; CAPILLA, 2013), the binding time can be classified as follows:

- *Compile-time:* The variability is resolved before the actual program compilation (for example, using preprocessor directives) or at compile time.
- *Link-time:* The variability is resolved during module or library linking (for example, selecting different libraries with different versions of the exported operations).
- *RunTime:* The variability is resolved during program execution (for example, depending on user rights functionalities get disabled or enabled with conditions in the code)
- *Update-time or Post-runtime:* The variant binding happens during program updates or after program execution (for example, an update utility adds functionalities to existing modules).

The different binding times (e.g., compile-time, link-time, runtime, and post-runtime) involve different mechanisms (e.g., inheritance, parameterization, conditional compilation) and are appropriate for different variability implementation schemes (MCGREGOR; SODHANI; MADHAVAPEDDI, 2004). These mechanisms encapsulate the variable parts and provide support for instantiating the variations at code level. In (GACEK; ANASTASOPOULES, 2001), the authors list several variability mechanisms, such as:

- *Aggregation/Delegation:* It is an object-oriented technique, used when an object (Aggregated) cannot satisfy a request, thus it forwards a request to the delegated object which provides such requested services.
- *Inheritance:* It assigns base functionalities to super classes and extensions to subclasses.
- *Parameterization:* The idea behind it is to represent reusable software as a library of parameterized components, which has their behavior determined by the values parameters that were set to.
- *Overloading:* It uses an existing name, to operate on different types. This name or symbol can be assigned to functions, procedures or operators.
- *Delphi Properties:* Considered an attribute of an object, it associates a specific action with reading or modifying its data. These properties provide control over access to an object's attributes, and they allow attributes to be computed.
- *Dynamic Class Loading:* It is a standard in Java in which classes are loaded into memory as soon as they are needed. It is interesting in SPL infrastructure because in that way a product can query its own context and that of its user, and decide at runtime which class versions to load.
- *Static Libraries:* It contains a set of external functions that can be linked to an application after it has been compiled. The functions signature are known to the compiled code and must remain unchanged. This way, the different libraries can be selected providing some kind of variability support.
- *Dynamic Link Libraries:* It can be useful for the selection of variant functionalities and are loaded when needed into applications at runtime.
- *Conditional Compilation:* It enables the programmer to control over the code fragments/segments to be included or excluded from a product version. It uses directive marks to delimiter the feature code.

- *Frames:* They are source files equipped with preprocessor-like directives which allow parents to copy and adapt children. The idea is to provide ways to maximize code reusability through the definition and use of frames.
- *Reflection:* Relates to metaprogramming in which objects in higher levels of abstraction are established to represent entities like operating systems, programming languages, processors, object models, etc. It enables access to such metaobjects and therefore allows architecting flexible systems.
- *Aspect Oriented Programming:* It enables the programmer in cleanly separating components and aspects from each other by providing mechanisms that make it possible to abstract and compose them to produce the overall system.
- *Design Patterns:* It could be applied on SPL context since many of them identify system aspects that can vary and provide solutions to manage this variation.

The authors in (GACEK; ANASTASOPOULES, 2001) state that different approaches are needed to support different problems. There is no silver bullet, thus, the techniques need to be mapped to known problems, and sometimes a combination of them is the most appropriated approach.

2.5 Software Product Lines Architecture

In the context in which the software complexity is growing at an alarming rate and the costs of software development and maintenance must be restrained, the Product Line Architectures (PLAs) is a key aspect since it enables companies to amortize the effort of software design and development over multiple products, thereby substantially reducing costs. It is a guideline for design of architectures within a given domain, i.e., it provides insights regarding how to build a particular kind of system (NAKAGAWA; ANTONINO; BECKER, 2011).

The PLA should ideally be flexible enough to support changes in features and in how they are composed to develop products (NAKAGAWA; ANTONINO; BECKER, 2011), since PLAs have a longer life span and support the development of closely related products. Thus, it is important to carefully evaluate the design decisions prior to perform implementation and refactor (ANDRADE; ALMEIDA; CRNKOVIC, 2014).

Through the architecture, the business rules are expressed, and the requirements (i.e., functional and nonfunctional properties) are projected to be satisfied. It does not consider details of implementation or algorithms, but rather addresses the interactions and the behavior of components. All this knowledge intensive art depends on software architect experience, registered by architectural tactics and architectural patterns, which are reusable architectural solutions that promise to address specific concerns in software

architecture (FAEGRI; HALLSTEINSEN, 2006). In summary, they are representation of knowledge regarding to how to solve recurrent problems (SCHMIDT; BUSCHMANN, 2003).

It is important to highlight that no set of tactics or patterns will create an architecture that is optimal for all stated requirements since their use can generate various effects. Making these trade-offs effectively becomes even more important in the SPL context which aims to deliver multiple product variants to the market (FAEGRI; HALLSTEINSEN, 2006).

2.6 Non Functional Properties

According to (SIEGMUND et al., 2008), there is no general agreement regarding to the nomenclature related to non-functional requirements, instead there are closely related concepts, such as: *quality attribute* (IEEE. . . , 1990; SOFTWARE. . . , 2001; BASS; CLEMENTS; KAZMAN, 2012), *nonfunctional requirements* (MYLOPOULOS; CHUNG; NIXON, 1992), *quality characteristics* (ISO/IEC, 2010), *quality factors* (IEE, 1998), and *quality properties* (ROZANSKI; WOODS, 2005; SOARES et al., 2014). In this work, we will interchangeably mention quality attributes and nonfunctional properties.

The nonfunctional properties are defined as being those properties that do not describe or influence the principal task / functionality of the software but can be observed by end users in its runtime behavior (LOHMANN et al., 2005). For instance, if it is important that a software system runs functionalities related to its business properly, it may also be interesting that it runs with a certain degree of security, performance, and availability. This way, functional and nonfunctional properties should work fine for a product to be viable in the market. In addition, these nonfunctional properties play a crucial role during system development, serving as selection criteria for choosing among myriads of decisions (MYLOPOULOS; CHUNG; NIXON, 1992).

There are several nonfunctional properties, which manifest themselves in the product, whereas some attributes manifest themselves in the interaction when the product is used (ISO/IEC, 2010). According to (ISO/IEC, 2010; BASS; CLEMENTS; KAZMAN, 2012), the product nonfunctional property can be divided into those that are observable or measurable at runtime (i.e., security, availability, performance) and to those that are not (i.e. modifiability and testability). Many nonfunctional properties are architectural, which means that the software architecture is critical to their realization (BASS; CLEMENTS; KAZMAN, 2012). A considerable portion of nonfunctional properties is determined by the choices done during the architecture design. These decisions and design tactics encapsulate reusable design strategies and solutions (BASS; CLEMENTS; KAZMAN, 2012; FERNANDEZ; ASTUDILLO; PEDREZA-GARCIA, 2015).

According to (MYLLäRNIEMI; RAATIKAINEN; MäNNISTö, 2012), there are two explanations for varying nonfunctional properties. There may be *differences in the user or customer needs* and *differences in the hardware or resources that affect or constrain*. Configuring it in SPL is still a challenge (SINCERO; SCHRODER-PREIKSCHAT; SPINCZYK, 2010; MYLLäRNIEMI; RAATIKAINEN; MäNNISTö, 2012), and many design decisions improve one single nonfunctional property at the expense of another, resulting in trade-offs which are usually resolved by finding a consensus (BARBACCI et al., 1995).

2.6.1 Types of Non Functional Properties

As mentioned before about the standard nomenclature, there is no common way to classify nonfunctional properties. One example of such classification is the product quality model and model of quality in use proposed by ((ISO), 20011) which was strongly influenced by (SOFTWARE. . . , 2001). It defines quality with categories of characteristics to the software product and computer systems, besides providing a terminology for specifying, measuring and evaluating system qualities. While the product quality model defines six categories: Functional Suitability, Maintainability Usability, Performance Efficiency, Security, Reliability Compatibility, and Portability; the model of quality in use defines five: Effectiveness, Efficiency, Satisfaction, Safety, Usability. Each one of these characteristics is still refined in same others sub-characteristics.

According to (WAGNER, 2013), due to the ambiguity among the decomposition principles used for quality characteristics and the lack of detailed measures, less than 28% of the companies use this standard model and 71% of them have developed their own variants, which could indicate the need of an update.

(MARI; EILA, 2003) divided the quality attributes into two categories, execution and evolution quality attributes. The execution qualities are observable at runtime as the behavior of the system and evolution attributes are observable during the system life-cycle that characterize different phases in the development and maintenance process (see Table 1).

In the context, the maintainability is an important quality attribute to be early addressed during software development life cycle, especially on SPL development due to its natural complexity. Next Section different categories of maintenance are described.

2.6.2 Maintainability

Characterized by their huge cost and expensive implementation, maintenance initiates after the product release and aims to correct, keep the software updated, as well as fit with the environment new needs. According to (PRESSMAN, 2014), around 20% of all maintenance work is spent fixing mistakes, the remaining 80% is spent adapting the

Tabela 1 – Execution and Evolution quality attributes (MARI; EILA, 2003)

Execution Quality Attributes	Description
Performance	Responsiveness of the system, which means the time required to respond to stimuli (events) or the number of events processed in some interval of the time.
Security	The systems ability to resist unauthorized attempts at usage and denial of service while still providing its service to legitimate users.
Availability	Availability measures the proportion of time the system is up and running.
Usability	The system's learnability, efficiency, memorability, error avoidance, error handling and satisfaction concerning the users' actions.
Scalability	The ease with which a system or component can be modified to fit the problem area.
Reliability	The ability of the system or component to keep operating over the time or to perform its required functions under stated conditions for a specified period of time.
Interoperability	The ability of a group of parts to exchange information and use the one exchanged.
Adaptability	The ability of software to adapt its functionality according to the current environment or user.
Evolution Quality Attributes	**Description**
Maintainability	The ease with which a software system or component can be modified or adapt to a changed environment.
Flexibility	The ease with which a system or component can be modified for use in applications or an environment other than those for which it was specifically designed.
Modifiability	The ability to make changes quickly and cost-effectively.
Extensibility	The systems ability to acquire new components.
Portability	The ability of the system to run under different computing systems: hardware, software or combination of the two.
Reusability	The system's structure or some of its components can be reused again in future applications.
Integrability	The ability to make the separately developed components of the system work correctly together.
Testability	The ease with which software can be made to demonstrate its faults.

system according to the external environment needs, making enhancements requested by users and reengineering an application for future use.

In (LIENTZ; SWANSON, 1980) and ISO/IEC 14764 (ISO, 2006), four categories of maintenance are defined, as follows:

- *Adaptive Maintenance:* Aims to adapt the system in response to data requirements or environment changes;
- *Perfective Maintenance:* Addresses the modifications after product delivery to handle any enhancements in respect of system performance or maintainability improvements;
- *Corrective Maintenance:* It is a reactive modification of a system usually called "fixes" and performed after delivery. It is responsible for fix discovered problems (software, implementation and performance failures); and
- *Preventive Maintenance:* It is concerned to correct and detect faults before it becomes a fault, preventing problems in the future.

During the adaptive or perfective maintenance, the software specification is modified to join the improvements or adaptations (WAHL, 1999). In corrective maintenance, the specification may not be modified or no new modules may not be added. Most of the changes imply in addition, modification and deletion of instructions (LEUNG; WHITE, 1989). Preventive maintenance is usually performed on critical systems (ABRAN et al., 2004).

2.7 Chapter Summary

Software Product Lines is an approach to software reuse that during the last years has proven its applicability in a broad range of situations, producing impressive results [2]. In order to achieve all software product lines benefits, three essential activities must be followed: Core Asset Development, Product Development, and Management. The assets are created during the core asset development phase to be further instantiated during product development to derive products, all of it controlled by management activity.

In this chapter, an SPL overview was presented, discussing its essential activities, the concepts regarding variability and how to manage it, some SPL adoption strategies, as well as, how to implement variability considering different binding times and mechanisms. Finally, it provides important discussion about types of nonfunctional

[2] http://splc.net/fame.html

properties and the ways in which the software maintenance can happen during the software life-cycle.

Next Chapter presents an overview of the software security area discussing its fundamental concepts, security taxonomy, tactics and techniques in order to define the base for the studies performed in this work.

3 AN OVERVIEW ON SOFTWARE SECURITY

Security has become an important requirement with the advent of multi-user computers in the late 1960s (DENNING; DENNING, 1979). With the increased number of people connected, the web and banks transactions, apps purchasing, it is even more important to the companies to protect its and customer data against unauthorized access. This protection can be also due to company secrets, public laws and/or regulations and to guarantee data and process safety and integrity, for example, planes flown on autopilot could crash if their process were corrupted. This security can be achieved in two different ways, the "logical security"addressed by software and "physical security"addressed by physical countermeasures or barriers. This degree of security needed for every company will depend on the environment in which the company operates.

To achieve this security, we can consider three logical areas: *Procedural Security:* which includes operational, administrative and accountability procedures, *Environmental or Physical Security:* it comprehends personnel and physical security, e.g., physical controls such as door locks, and *Technical Security:* which consists in security related to all communications, data, and automated information system security (e.g. protecting the authenticity and integrity of message traffic, hardware, software and firmware protection, etc.). In this work, we are considering the technical security logical area since it is related to software development.

Extensive and philosophical discussions about how to organize these diverse security elements are available in well-known references accepted in academia and industry (HUI et al., 2010; VERDON, 2006; HAMED; AL-SHAER, 2006), however, they are developed for specific purposes which make difficult to define a single unified taxonomy (SCHUMACHER et al., 2005). Along with this Chapter, we describe the foundations of software security by emphasizing the elements that are important for this research.

This chapter is organized as follows: Section 3.1 presents a security taxonomy. Section 3.2 details how to achieve security at the architecture level, by describing the following tactics: detect attacks, resisting attacks, react from attacks and recover from attacks. Section 3.3 discusses how security has been treated by SPL research community.

3.1 Security Taxonomy

As mentioned before, it is clear that security is spread or should be held by different company decision levels. (SCHUMACHER et al., 2005) benchmarked existing

security taxonomies and come up with one containing three major divisions: *Security Strategy and Policy*, *Services*, and *Mechanisms and Implementations*. Table 2 summarizes each of them with their respective elements.

Tabela 2 – Security taxonomy (SCHUMACHER et al., 2005).

Security Strategy and Policy	
Violations	- Deception, disruption, unauthorized disclosure, usurpation.
Risk Management	- Asset valuation, vulnerability, assessment, threat assessment, risk assessment, risk mitigation
Approaches	- Prevention, detection, response, planning, diligence, mitigation
Services	
Security support services	- Authorization, system security policy, security planning, registration, operational maintenance, concept of operations, continuity of operations, ...
Security Services	- Identification and authentication, deterrence, accounting, access control, boundary protection, non-repudiation, system recovery, and so on
Mechanisms and implementations	
Management support mechanisms	- Information system security policies, training, configuration management, disaster recovery, connection service agreements, ...
Automated mechanisms	- Encryption, scanners, firewalls, proxies, filters, packet sniffers, hashing, integrity monitoring, log parsers, making/labeling, logon/off (user ID and passwords), biometrics, tokens intrusion detection system, access control lists, RBAC, digital signatures audit, ...
Physical mechanisms	- Human guards, doors, vaults, locks, sensors, walls, ...
Procedural mechanisms	- Sign-ins, backup, restore, removal, incident response handling, training, security administration, personnel, configuration procedures, and so on

It is important to mention that security is concerned with the protection of assets, ensuring that actions are appropriate, and holding actors responsible for their actions. However, in order to achieve the simplest security approach in all previously stated levels, some properties should be achieved, such as (BASS; CLEMENTS; KAZMAN, 2012): *Confidentiality:* The data or service are protected from unauthorized access, *Integrity:* the data or service are not subject to unauthorized manipulation and, finally, *Availability:* the system will be available for legitimate use.

In order to achieve security at the code level, the software architect should understand how the company conducts its activities in its business, professional, economic, social and legal environment (DASHOFY; HOEK; TAYLOR, 2002). This information provides important insights regarding how to apply tactics, which in turn may be realized using for example security patterns (FERNANDEZ; ASTUDILLO; PEDREZA-GARCIA, 2015). Figure 7 shows the way in which the software architect can decrease the semantic gap between company policies and software code. It is important to mention that the associations between them (policies, tactics and security patterns) are many-to-many, since a policy may be applied using several tactics, which in turn can be realized using several security patterns. On the other hand, a security pattern may realize more than one tactic, and a tactic may come from several policies. Next Section presents more details about security tactics and how to achieve them.

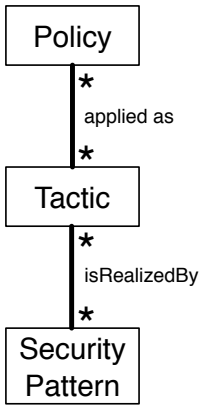

Figura 7 – From policies to security patterns (FERNANDEZ; ASTUDILLO; PEDREZA-GARCIA, 2015).

3.2 Software Security Tactics

Firstly introduced by (BASS; CLEMENTS; KAZMAN, 2003), architectural tactics are described as "measures"or "decisions"taken to improve some quality factor or *"architectural building blocks from which architectural patterns are created"* (BASS; CLEMENTS; KAZMAN, 2012). The authors in (ROZANSKI; WOODS, 2005; WOODS; ROZANSKI, 2005) define architectural tactics *"as architectural design guidance, strategies, or advices on how to drive a general design issue related to improving required quality attributes without imposing a particular software systems."*. In summary, it corresponds to design decisions with respect to a quality attribute.

Since the tactics initial formulations, there have been formalized (BAGHERI; SULLIVAN, 2011), compared with patterns (RAY et al., 2004), associated to the Common Criteria (PRESCHERN, 2012) and associated to styles (HARRISON; AVGERIOU, 2010). Nonetheless, the initial set (BASS; CLEMENTS; KAZMAN, 2003; BASS; CLEMENTS;

KAZMAN, 2012) has only been refined once (RYOO; LAPLANTE; KAZMAN, 2012). In this context, a study recently published (FERNANDEZ; ASTUDILLO; PEDREZA-GARCIA, 2015) presented examination, pruned and reclassified the architectural tactics for security, considering the previous defined in (RYOO; LAPLANTE; KAZMAN, 2012; BASS; CLEMENTS; KAZMAN, 2012), since the authors believe that for an effective and correct use in building security systems, the original set of tactics needed that revision. Figure 8 shows the result of this refinement.

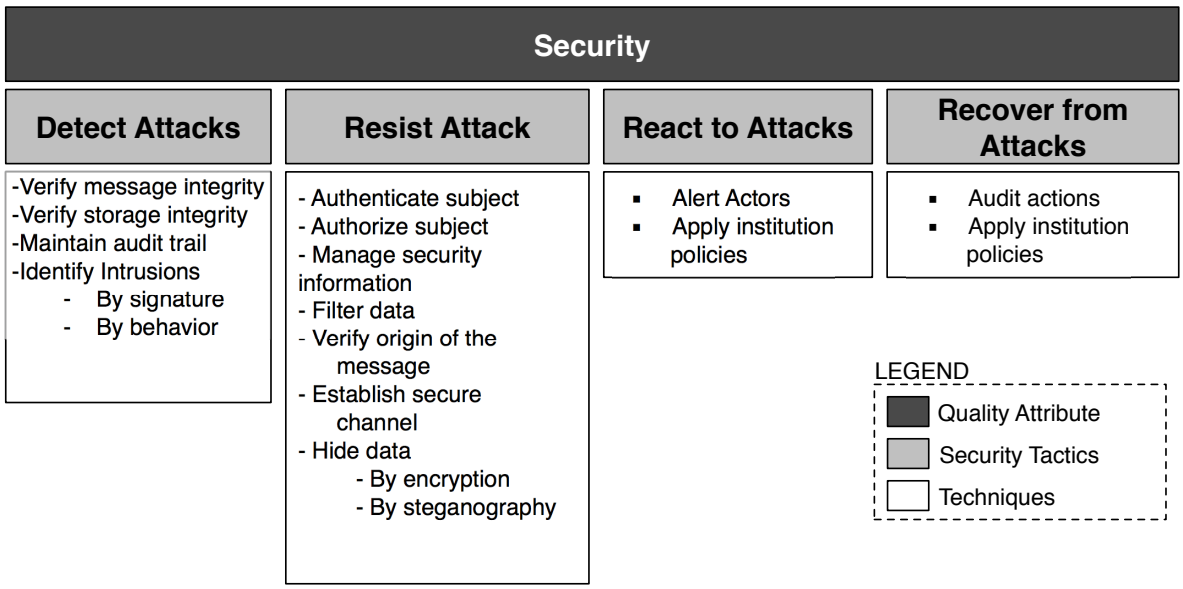

Figura 8 – Classification of security tactics according to (FERNANDEZ; ASTUDILLO; PEDREZA-GARCIA, 2015).

It is important to mention that a tactic should not be too general or too specific. If too general may be confused because of the number of alternatives; if too specific we have mechanisms or techniques instead of tactics, and it can reduce the number of possibilities of the software architecture (FERNANDEZ; ASTUDILLO; PEDREZA-GARCIA, 2015).

3.2.1 Detect Attacks

In this Section, all techniques related to how to detect an attack are described.

- *Verify message integrity:* It implies techniques such as checksums or hash values to verify the integrity of messages, resource files, deployment files, and configuration files. While the former is a validation mechanism in which the system maintains redundant information for configuration files and messages, and uses this information to verify the configuration files or message when it is used. The second generates a unique string by a hash function whose input

could be configuration files or messages. Even a slight modification in the original files or messages results in a big change in the hash value.

- *Verify storage integrity:* It should define measures and/or functions to make sure that the database or files have not been modified.
- *Maintain audit trail:* A system function that can be used to detect attacks and recover from attacks. An audit trail consists of a copy of each transaction applied to the data in the system together with identifying information. This audit information can be useful to trace the actions of an attacker, support nonrepudiation, and support system recovery.
- *Identify Intrusions:* There are two ways to achieve this technique. The first approach, demands that all users must have a signature in the system, it allows the security personnel to keep track of all operations of a given user. The second approach considers the user behavior, in which the system analysis the user behavior to indicates if that sequence of operations is a suspicious action.

3.2.2 Resisting Attacks

Although it is not always possible to resist an attack, due to the ever-increasing variety of attacks, there are some ways to achieve it. The authentication and authorization are techniques to ensure confidentiality and integrity, especially, when the system requires different users with different rights (SANDHU; SAMARATI, 1994; SCHUMACHER et al., 2005). Next, we describe these techniques:

- *Authenticate Users:* Verifies *Who is the user?*. Check if a user is a person he/she should be. It addresses the need to recognize an actor (human, a process or other entity) that is interacting with the system, ensuring that access is only granted to authorized people. It can be achieved by different ways, such as: *(i)* something user knows, login and password; *(ii)* something the user possesses, credit-card, smart card; *(iii)* something user is, biometric signature, fingerprint, voice-print or the eye iris. The Identification and authentication service obtains an identity from the actor, translate the identity to an ID, and authenticates the ID using an authenticator. Most of the time, it is used in support to other security service. For example, the access control depends on authentication to guarantee that only legitimate users access the system. The individual and group are two primary categories of identification and authorization. The former determines the individual actor interacting with a process, for example, a person logging on to a computer. The second determines whether an actor interacting with a process is a member of a particular group. For example, checking if a person is an employee of the financial department, to ensure their proper rights.

- *Authorize Users:* It verifies who is authorized to access certain resources in a system when the access to these resources need to be controlled (SCHUMACHER et al., 2005). It ensures that an authenticated actor has the rights to access and modify either data or services (BASS; CLEMENTS; KAZMAN, 2012). This access control can be performed using two models: *discretionary access control*, in this model, there is no clear separation of user and administration and users can be owners of data they create and behave as their administrator. On the other hand, in the mandatory access control, only selected users are allowed to grant rights, and users cannot transfer them. Users and data are classified by administrators. Next, we will present a set of patterns that are essential to understand our study.

  In order to describe each of the authorization patterns, we used a short description of the problem it solves, where the pattern is applicable and a brief description of how to solve that problem. The main patterns to address this issue are following described (SCHUMACHER et al., 2005; D.; M.; C., 2006; FERNANDEZ-BUGLIONI, 2013):

  Authorization: It describes who is authorized to access the system functionalities and resources in any environment where a resource control is required. It is achieved by explicitly indicating the permissions granted to subjects that have access to protected functionalities. On the contrary, any subject could access any functionality. Figure 9 shows the class diagram of the entities involved. The `Subject` class represents an entity that attempts to access a resource (`Protection Object`) in some way. The relationship between subject and object defines an authorization (`Right`), which describes the access type (for example, read, write) the subject has to the object. This class can check the rights for a given subject or identify who is allowed to access a given object.

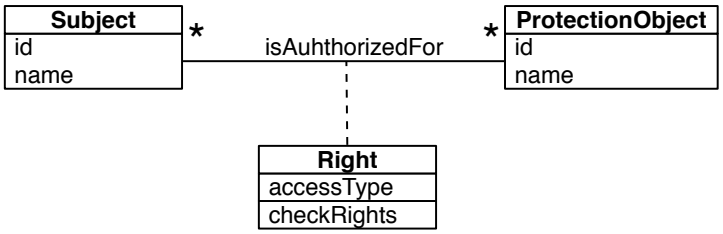

Figura 9 – Authorization Pattern.

  Role-Based Access Control: It controls the access resources only based on the subject role. It is used in any environment where access should be controlled and users can be classified according to their jobs and tasks. It provides ways to convenient manage authorization rights, otherwise, the number of individual

rights is just too large, requiring the storing of many authorization rules and making difficult to the administrators to keep track of all these rules. In addition, it is hard to associate semantic meanings to the rules. This pattern extends the idea of the authorization pattern by translating roles as subjects. Figure 10 shows the class diagram in which the User and Role classes describe registered users and their predefined roles respectively. Users are assigned to roles, roles are given rights according to their functions. The `right` association class defines types of access that a user within a role is authorized to apply to the protection object.

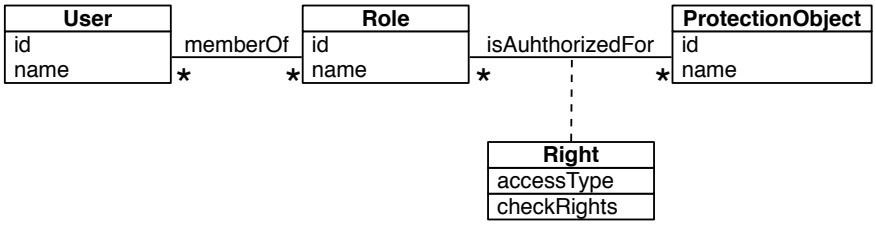

Figura 10 – Role-Based Access Control Pattern.

Multilevel Security Pattern: It is used when the systems need to provide several security levels, as well as, how to decide on accessing it an environment with security classifications. This pattern provides a structure that allows us to have different security levels for both subjects and objects. To implement the structure of Multilevel security (see Figure 11), there must be an instance of the class `Subject` Classification for each subject and an instance of the class `Object` Classification for each object. These instances are used to set security levels and object security categories to a subject. This pattern can be expensive since subjects and objects should be classified in security levels.

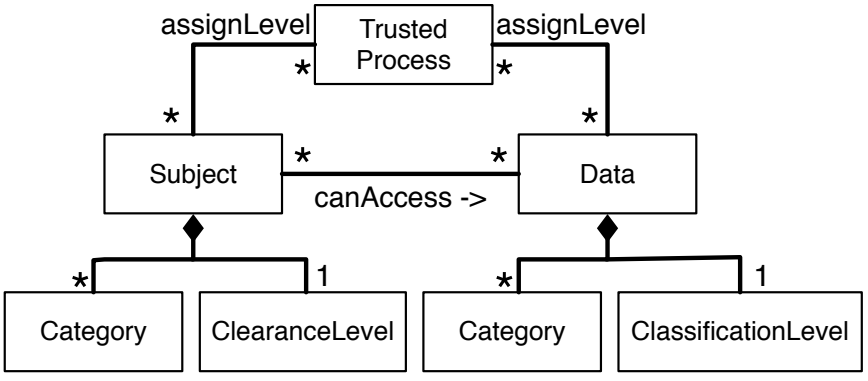

Figura 11 – Multilevel Security Pattern.

Session Pattern: The idea is to provide an environment where a user's rights can be restricted and controlled. A subject can be in several sessions at the same time and it has a limited lifetime. When a session is started a user only activates a set of authorization contexts assigned to him/her, then, only the necessary rights are available within this session. To codify the Session security pattern (Figure 12) there must be a `Subject` class which describes an active entity that accesses the system and asks for resources. The `Execution Context` class describes a set of contexts of execution or active rights that the user has in a given interaction.

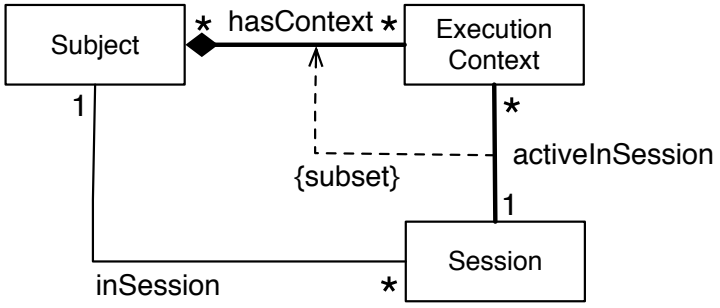

Figura 12 – Session Security Pattern.

Reference Monitor Pattern: It is used in a computational environment in which users or processes make requests for data or resources. The problem happens if we do not enforce the authorizations it is the same as not having them, allowing users and processes to perform all type of illegal actions. The simple definition of authorization rules is not enough they must be enforced whenever a user makes a request for a resource. In this context, the Reference monitor pattern (see Figure 13) defines an abstract process that intercepts all requests for resources and checks them for compliance with authorizations. The authorization rules indicate a collection of authorization rules organized as access control lists (ACLs).

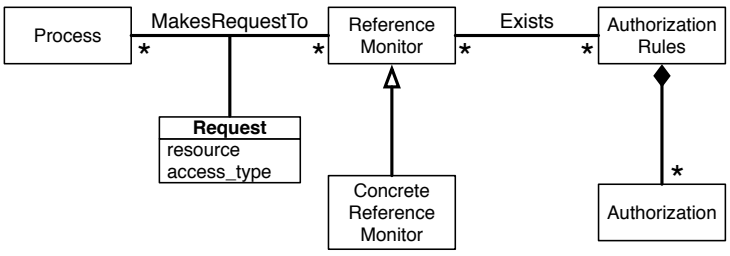

Figura 13 – Reference Monitor Pattern.

Single Access Point (SAP) and CheckPoint: The use of SAP prevents external entities from communicating directly with components in the system. All inbound traffic is routed through one channel, in which monitoring can be easily performed. It is the appropriate place to capture an information log on the parties currently accessed in the system. In order to perform a check to distinguish between user mistakes and malicious attacks, the Checking Point pattern is used to analyze all petitions and messages. It is important to mention that SAP is predestined to be combined with Checking Point (see Figure 14) for all messages to be supervised.

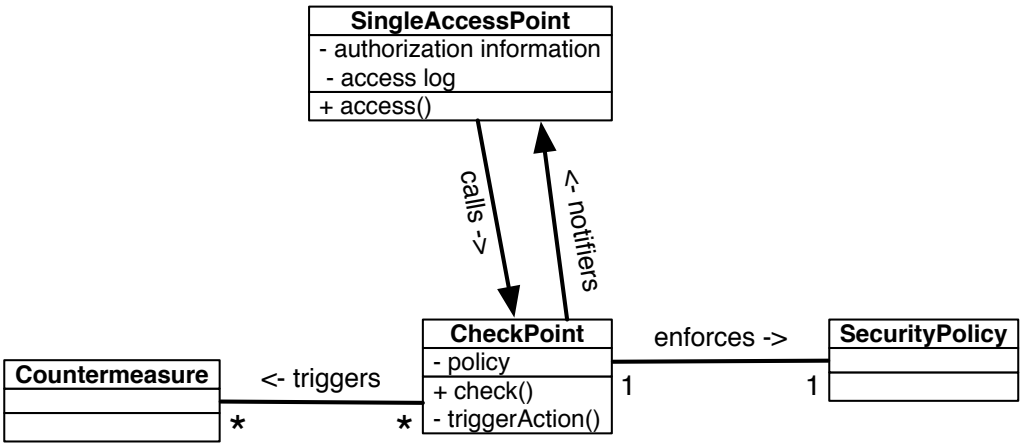

Figura 14 – Single Access Point and CheckPoint.

- *Manage Security Information:* It includes the management of keys for cryptography, the secure storage of authorization rules, and other ways to handle security information. It is a fundamental technique to have a secure system and should be performed by the system.
- *Filter Data:* This technique is responsible for avoiding attacks based on abnormal inputs or coming from untrusted sources.
- *Verify Origin of the Message:* It is responsible for verifying the data authenticity, ensuring the legitimacy of the input data.
- *Establish Secure Channel:* As applied in the distributed system context, it is responsible to establish a safe channel to data traffic. This channel can be implemented by a virtual private network (VPN) or by a Secure Sockets Layer (SSL). Once this secure channel is established the data could be hidden using the next technique.
- *Hide data:* It could be achieved by two possible implementations: by *Encryption,* an algorithm encodes a message or information in such a way that only authorized parties can read it. By *Steganography* which is concerned with concealing

the fact that a secret message is being sent, as well as, concealing the content of the message.

3.2.3 React from Attacks

These techniques are responsible for responding to a potential attack. Either by generating an alert for the responsible or by applying internal security policies of the institution.

- *Alert Authors:* From the attack occurrence, some system actors (operators, other personnel or collaborator systems) must be notified in order to provide some action. These actions may involve from the hiring of new services, functions or even security equipment.
- *Apply Institution Policies:* The reality imposes that the company must be prepared for things to go wrong within the company. This way, contingency plans should comprehend actions to react from attacks. This involves more administrative actions and updates in their security policies.

3.2.4 Recover from Attacks

Once a system has detected and attempted to resist an attack, it needs to recover from it.

- *Audit Actions:* The system keeps a record of user and system actions and their effects to help trace the actions of, and to identify, an attacker. This way, the audit trails are then analyzed to attempt to prosecute attackers or to create better defenses in the future. It is important to mention that availability tactics make use of audit trails to restore of services (BASS; CLEMENTS; KAZMAN, 2012).
- *Apply Institution Policies:* The reality imposes that the company must be prepared for things to go wrong within the company. This way, contingency plans should comprehend actions to react from attacks.

The last two set of tactics were generically described since the definition of a specific function depends on institution policies, type of the application, and should be performed by the system. For this reason, a general function was not defined (FERNANDEZ; ASTUDILLO; PEDREZA-GARCIA, 2015).

3.3 Variability in Software Security

Most of the studies in the literature consider quality attributes in general, by proposing a method or a construct to be applicable to all quality attributes (MYLLäRNI-

EMI; RAATIKAINEN; MäNNISTö, 2012; SOARES et al., 2014). Few of them focused on security, a cross-cutting concern that should be carefully held during the design phases and support the decision-making (FAEGRI; HALLSTEINSEN, 2006), since a given weakness in security can cause problems throughout the products of a product line.

In this context, the authors in (WANG et al., 2006), proposed a dynamic, privacy-enabling personalization infrastructure and conceived it as a product line architecture. The idea behind it is that user-adaptative system applications take individual characteristics of their current users into account and adapt their behavior accordingly. For doing so, they collect amounts of personal data about users that need to be subject to privacy laws and regulations. The author's primary focus was on data processing step of web personalization where methods can be applied to derive additional information about users. Based on this personalization point of view, they ask how can personalized web-based systems maximize their personalization benefits, while being compliant with privacy laws, industry and company regulations, and privacy preferences of the current user. Despite implementing the proposed solution at the code level, it is not directly related to software security issue, instead of the way that identifies the customer personal data in order to apply different privacy law and regulations. It is critical in websites with customers from different countries obeying different laws.

The study in (FAEGRI; HALLSTEINSEN, 2006), discusses the viability to represent security knowledge in a reference architecture and if this architecture is useful for security architecture design in SPL. Based on that, they presented a reference architecture which is based on techniques and practices from SPL engineering and information security, which composes a framework for security architecture design in SPL. Despite having taken into account and documented the data from three companies, no evaluation regarding the implementation of these proposed design decisions was considered.

The authors in (DANIEL; EDUARDO; MARIO, 2008) proposed a security standard-based process for software product line development, which is a set of activities in the domain engineering. It deals with security requirements from early stages of SPL life-cycle in a systematic and intuitive way. Additionally, it deals with security artifacts variability and traceability, providing a security core asset repository. The idea is that the products generated from this process will conform with the most relevant security standards with regard to the management of security requirements such as ISO/IEC 15408, ISO/IEC 17799:2005 and ISO/IEC 27001 standards. Despite proposing activities for each step of the SPL framework, nothing was done in relation to how to implement such security at the code level, nor to evaluate this implementation.

In (MYLLäRNIEMI; RAATIKAINEN; MäNNISTö, 2015), the authors use design

models to build artifacts and generalize design theory for representing and configuring security and functional variability from requirements to architecture in a configurable SPL. They developed and evaluated a configurator tool to find consistent products as stable models. This study did not evaluate the code related to security issues or any other related aspect, it focuses on product derivation.

3.4 Chapter Summary

In an SPL, security may need to be varied to meet the different customer's wishes. Consequently, security variability must be managed both from customer and product line architecture point of view (MYLLäRNIEMI; RAATIKAINEN; MäNNISTö, 2015). In order to achieve this security at the architecture level, different security tactics and techniques can be used.

In this chapter, a taxonomy described different levels in which the security should be held by different company decision levels. It also presented a benchmark regarding to security taxonomies with three major divisions: *Security Strategy and Policy*, *Services*, and *Mechanisms and Implementations*. Considering the mechanism and implementations division, four security tactics were presented each one containing different techniques to achieve the tactics mains purpose. Finally, an overview about how software security is being addressed by studies in the SPL area was also discussed.

Next Chapter describes the testbed developed to support all evaluations performed in our study. It also presents its feature model, architecture and how the security was implemented in each testbed.

4 SOFTWARE PRODUCT LINES TEST BED

Software Product Lines (CLEMENTS; NORTHROP, 2001; POHL; BOCKLE; LINDEN, 2005) is a contemporary approach to software development that promotes reuse while reducing overall development time and cost, as well as improving product quality. All these benefits do not come for free, it demands mature software engineering, planning and reuse, adequate practices of management and development, and also the ability to deal with organizational issues and architectural complexity. Thus, it is important the development of new techniques, tools, and methods to deal with SPL complexity required by the variability management.

Nevertheless, in order to evaluate new ideas before being used in practice, it is important to build platforms for conducting rigorous, transparent, and replicable testing of scientific theories, computational tools, and new technologies.

In this chapter, the results of the development from three SPLs are presented, serving as the base for a family of experiments (BASILI; SHULL; LANUBILE, 1999) which enables us to evaluate the impact of security techniques on SPL maintainability. It is important to mention that all SPLs were built using the *proactive* approach, in which the organization analyzes, designs and implements the overall SPL to support the full scope of products needed on the foreseeable horizon (KRUEGER, 2002b). The remainder of this chapter is organized as follows: Section 4.1 presents a related work and the main reason to create new testbeds. Section 4.2, Section 4.3 and Section 4.4 detail how each SPL was built regarding to activities, tools, stakeholders, code repositories and artifacts. In addition, some refactoring and code evolution implemented before security technique implementation were also described. Finally, Section 4.5 describes each of the security techniques and how they were integrated to SPL architectures.

4.1 Related Work

This section describes some studies found in the literature that also use SPLs to perform different studies and experiments.

SPL2go[1] is a catalog of SPLs for which domain implementation (i.e., source code) and variability model (e.g. feature model) are publicly available. It brings information such as tool used to automation, languages, number of features, lines of code, classes, and products. A brief description of the domain is also available.

In (FIGUEIREDO et al., 2008), the MobileMedia SPL was designed with an academic purpose, but including change scenarios with mandatory and optional features

[1] http://spl2go.cs.ovgu.de/

to be further exercised during evolution. It serves also as a reference for aspect-orientated programming studies.

In (APEL; BEYER, 2011), the authors provided a supplementary material of an empirical study on forty SPLs of different sizes and domains. They also made available data collected from these SPLs and some results from the statistical analysis.

A pedagogical product line (MCGREGOR, 2014) developed by Software engineering institute (SEI) provides different artifacts of a conceptual SPL. In addition to SPL artifacts such as business case, scope, a concept of operations, requirements, and architecture, they also provide class-tested pedagogical elements and suggested exercises.

In (GAIA et al., 2014), the authors developed an SPL to represent major features of an interactive web store system. It was designed for the academic purpose, but focusing on real features available in typical web store systems. It manages products and their categories, show products catalog, control access, and payments.

In (VALE; FIGUEIREDO, 2015), authors gathered releases of open-source Java software systems, often multiple versions. It has 112 systems, 15 systems with 10 or more versions, and 754 version total. There are two main distributions: the "r"(recent) release, containing the most recent versions of every system (112 systems) and the "e"(evolution) release, containing all versions of the 15 systems with 10 or more versions, a total of 579 versions.

Our Test Beds do not only present the source code but also provides the artifacts used during their development. They are: repository with commits and comments, feature model, relational model, feature code and product map. All of them were developed using JAVA language without any support of SPL development frameworks. It is important since most of the existing testbeds require that researchers know how to develop using a specific SPL tool or framework, which is considered a major barrier to the adoption of SPL and metrics extraction. As we are interested in evaluating security techniques, the SPLs needed to be developed in a way that allows the addition of those techniques to be further compared. Moreover, the SPLs considered in this study have a considerable size when compared with other such as: MobileMedia (FIGUEIREDO et al., 2008) and WebStore (GAIA et al., 2014). In summary, the idea is to concentrate complete updated information to enable other researchers to replicate or perform new studies.

4.2 RiSE Event SPL

The RiSE Event (SILVEIRA et al., 2016) is an SPL which aims to develop a product line that comprises the papers submission in conferences, journals, and related

events, and its management, including the control over the review life-cycle, as well as, the management of activities (workshop, tutorial, panels), users (speakers, organizers, reviewers), registrations, payments and certificates. It was built based on the main features found on largely used conference management systems, such as: EasyChair[2], JEMS[3] and CyberChair[4]. It constitutes a core asset base integrating many features to make it suitable for several conferences. Thus, based on this common base, the products can be derived.

The SPL was developed using the JAVA language following the MVC architectural pattern (see Figure 15 [5]) and a remote instance of MySQL database. The architecture is composed of four layers, they are: *(i) View,* composed of 79 graphical user interface classes responsible for generating outputs to the user based on changes in the model. *(ii) Controller,* can send commands to the model in order to update the models' state or send commands to its associated view to change the view's presentation of the model. This layer is composed of 39 classes. *(iii) Interface,* 19 interfaces establish a contract between the controller and its associated model, allowing to use different ways to store the data. *(iv) Model,* 19 classes are responsible to store data that is retrieved according to commands from the controller and displayed in the view. In summary, the SPL contains 34 functional features totalizing 26.395 lines of code, 1673 methods and 496 classes. The SPL code, as well as, the security techniques implemented are available at: <https://github.com/pamsn/RiSEEventSPL>

This TestBed was built considering five phases, as described: (i) the RiSE Event SPL was coded based on the results from the domain analysis and feature model which was built using Feature IDE tool[6]. (ii) Next, conditional compilation tags were inserted in the code in order to isolate each functional property (feature) code. It was supported by: Eclemma[7] to identify the functional property code during its execution, and CIDE [8] used to color the functional property code, as well as, identify their code interactions. The results were used in a build [9] `xml` file used to derivate SPL products.

4.2.1 Functional Properties

Figure 16 presents a simplified view of the RiSE Event SPL feature model (SILVEIRA et al., 2016). Examples of core properties are Submission, Review and

[2] www.easychair.org/
[3] jems.sbc.org.br
[4] www.borbala.com/cyberchair/
[5] Due to size limitation and to better illustrate the security impact the class diagram was not shown. It is also important to mention that the boxes located at the view layer are the system screens built using Swing framework.
[6] <http://wwwiti.cs.uni-magdeburg.de/iti_db/research/featureide/>
[7] http://www.eclemma.org/
[8] <http://wwwiti.cs.uni-magdeburg.de/iti_db/research/cide/>
[9] http://www.eclipse.org/eclipse/ant/

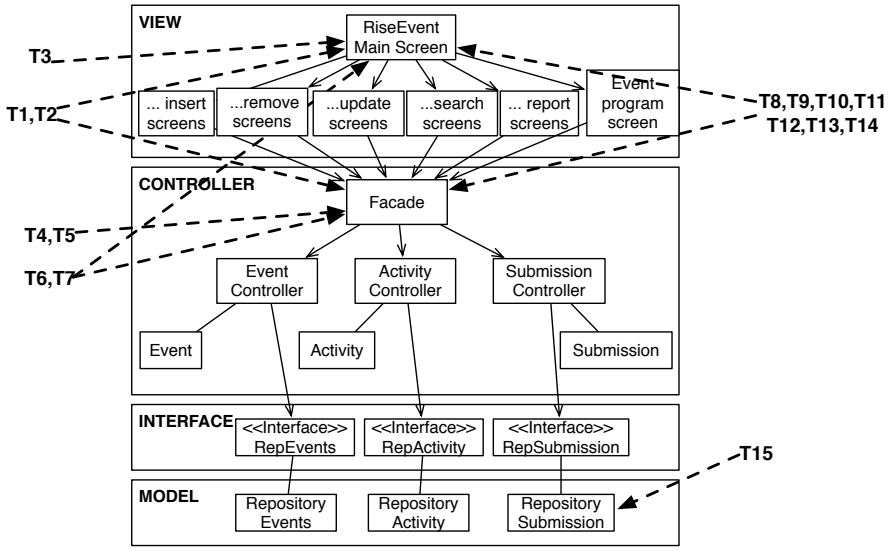

Figura 15 – RiSE Events SPL Architecture.

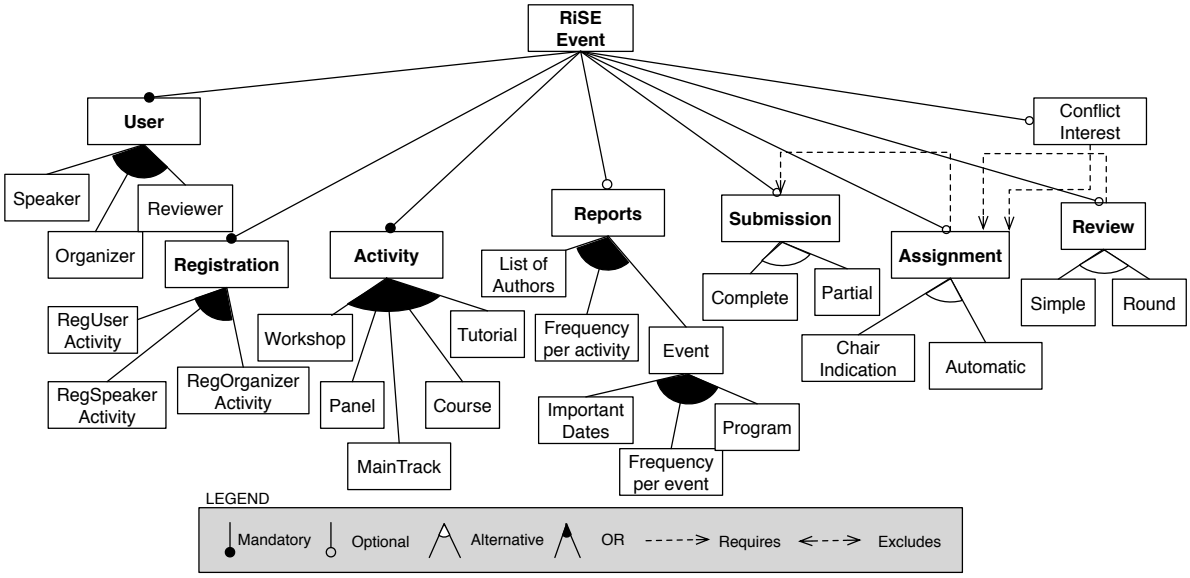

Figura 16 – RiSE Events SPL Feature Model.

Assignment and some optional properties are Reports and Conflict of Interest. More detailed information regarding all functional properties, their type and description can be seen in Appendix 7, Table 31).

4.3 RiSE Store SPL

The RiSE Store SPL was developed to support the main features of an interactive store. It was developed in an academic scenario, as part of M.SC/PH.D course, inspired

by an application called WebStore (FERREIRA et al., 2014) and Java Pet Store [10]. The Rise Store SPL is a desktop SPL which manages products, their categories and catalog, and control payments, FAQs and Bug reports. It comprehends the main features required to manage a shop store.

The SPL was developed by eleven post-graduate students as a project in a reuse course. They used JAVA language, adapting the MVC architectural pattern (see Figure 17 [11]) and a remote instance of MySQL database with an object-relational mapping framework for mapping an object-oriented domain model to a relational database. The SPL architecture was composed of three layers: *i View.* 23 graphical user interface classes to interact with potential users. *(ii) Controller.* composed of a facade class with 20 basic entities in order to send information from views classes to model layer classes. Different from the previous SPL, the controller classes were not built, leaving the business rules to views classes. *(iv) Model.* It was the layer with the most different structure when compared with the other SPL build. As it was using hibernate framework, to object-relational mapping, the developers reuse code as much as possible by concentrating common methods (e.g.: insert, update, remove, search) in one class, sending the variable parts (e.g.: find all, search by id) to specific classes. It resulted in 40 functional features totalizing 5426 lines of code, 443 methods and 74 classes. The SPL code, as well as, the security techniques implemented are available at: <https://github.com/pamsn/ufbadcc>

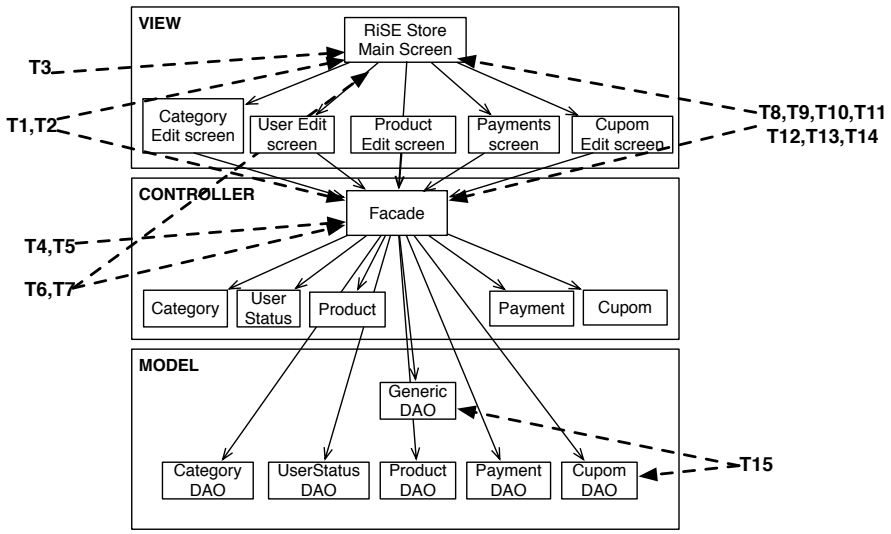

Figura 17 – RiSE Store SPL Architecture.

The RiSE Store SPL was developed based on benchmarking and domain analysis, as well as, the SPL concepts learned during the reuse course (started on Feb-2016

[10] http://www.oracle.com/technetwork/java/petstore1-3-1-02-139690.html

[11] Due to size limitation and to better illustrate the security impact the class diagram was not shown. It is also important to mention that the boxes located at the view layer are the system screens built using Swing framework.

to Jun-2016). Firstly, the data model was built considering all entities identified during brainstorm meetings. The professor acted as a client and the students as a software factory. Next, the feature model draft was built which was further refined using FeatureIDE tool [12], resulting in 40 functional features. The user interface mockups were build using Balsamiq[13] tool and the conditional compilation directives were placed throughout the code without any tool support. Each developer responsible to implement that feature was also responsible to isolate its code. Finally, a `xml` document was build with all features and applied to `ant` [14] in order to derivate different products. The timesheet, screen mockups, database script, code commits and `xmls` can be seen in the project repository website [15].

4.3.1 Functional Properties

A simplified view of the RiSE Store SPL is shown in Figure 18. Some of the SPL features are user management, notification and visualization through product management and categorization, client communication (e.g., FAQ, contact us and bug track management) cart management with checkout, promotions, gift cards and so on. See Appendix 7, Table 34 to detailed information regarding to this SPL.

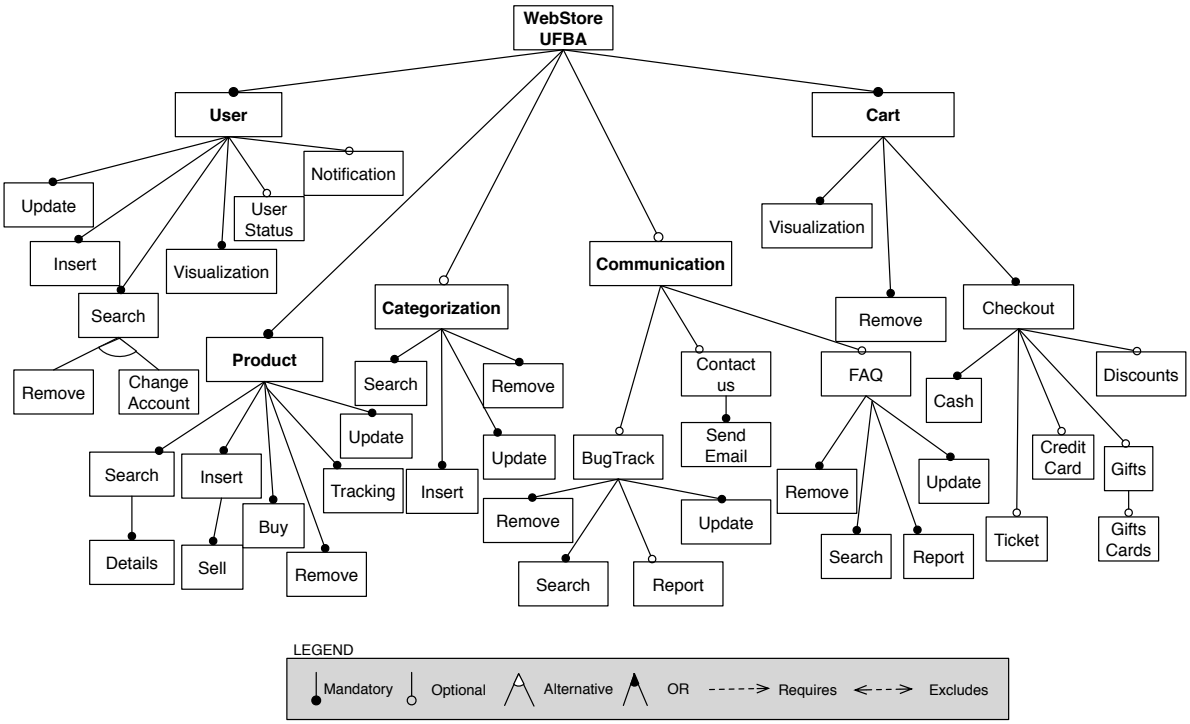

Figura 18 – RiSE Store SPL Feature Model.

[12] <http://wwwiti.cs.uni-magdeburg.de/iti_db/research/featureide/>
[13] https://balsamiq.com/
[14] http://www.eclipse.org/eclipse/ant/
[15] https://github.com/samiacapistrano/ufbadcc

4.3.2 RiSE Store Refactoring and Evolution

In order to apply the security concepts and mitigate some experiment threats during security techniques implementation, the SPL code has undergone some modifications and corrections, such as:

- As previously mentioned, the RiSE Store SPL makes use of hibernate framework to deal with object-relational mapping. Since the security techniques do not make use of hibernate framework and they have their code compared among them, we keep the same implementation as used in the RiSE Event SPL. Thus, the security techniques had their entities stored using JAVA with pure `sql` code, while the other entities had their data stored using the hibernate framework.
- During the SPL development, the students implemented an authentication in order to control the user access. As the authentication is one of the security techniques evaluated by our study, it needed to be removed. Thus, the user access code was removed and a refactoring was performed both in the SPL code and database scripts. In addition, singleton design pattern was added and the use of `Jpanel` was changed to `Jframe`, to enable the interaction among the security techniques screen with SPL screens.
- The JAVA generics was applied in order to get the buttons names and classes as used in the RiSE Event SPL.
- One of the security techniques needs to execute code when a specific screen is closed. As the original version of RiSE store SPL did not consider this scenario, we needed to insert the method responsible to close the screen using the button located on the top left corner (see Figure 19).

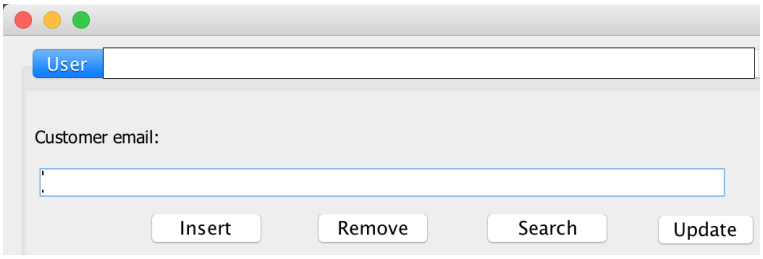

Figura 19 – Close button located at top left corner.

4.4 Law Office SPL

The Law Office SPL aims to manage essential activities of different law offices. It was developed in an academic context, as part of two undergraduate courses, databases

systems, and object oriented programming. It results in a desktop application which manages lawyers, trainees, processes, customers, departments, financial and reports. In addition, to manage a single law office, the SPL can be also used to manage a set of offices.

The SPL was developed by three students during the two courses. It uses JAVA language and a remote instance of MySQL database without any support of frameworks. The SPL architecture was composed of three layers (see Figure 20 [16]): *(i) View.* twenty-five graphical user interface classes were developed with the purpose to user interaction. The communication with the top layer is responsibility of *(ii) Controller* which has only the facade class with basic entities as found in the SPL Store, leaving the business rules to view classes. Finally, the *(iii) Model* layer with only one class which concentrates the code responsible for data storage of all software entities. It resulted in 34 functional features totalizing 16665 lines of code, 116 classes and 734 methods.

The Law Office SPL was developed based on domain analysis performed by the students, and Database and Object-Oriented Programming concepts viewed in both courses. First, the developers performed a set of interviews with two law offices in order to identify the main requirements. Based on that, the entities were modeled always with the supervision of the professor. The same happened in the Database course in which the relational model was created and transformed to its respective data tables. After some normalization and improvements, the database was created and finally integrated with the JAVA code. The SPL code, as well as, the security techniques implemented are available at: <http://www.paulosilveira.com/Implementa%c3%a7%c3%b5es_TestBed_SPLOffice.zip>

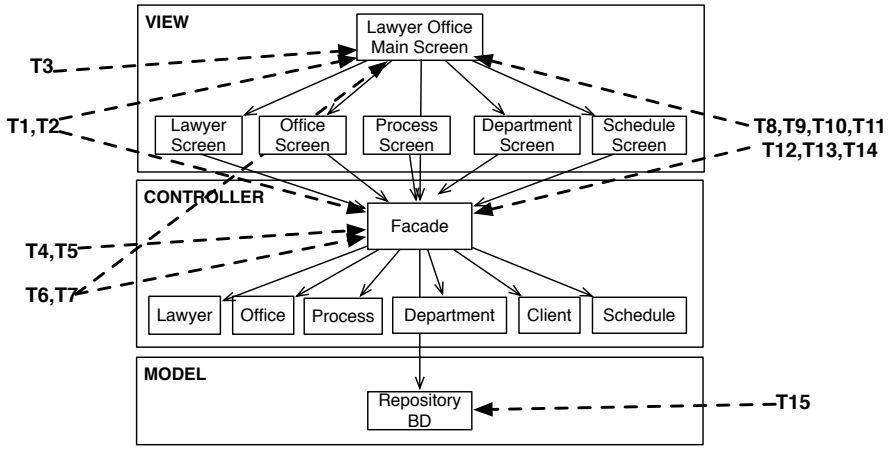

Figura 20 – Law Office SPL Architecture.

[16] Due to size limitation and to better illustrate the security impact the class diagram was not shown. It is also important to mention that the boxes located at the view layer are the system screens built using Swing framework.

4.4.1 Functional Properties

A short version of the feature model is shown in Figure 21. The system can have two types of customers: companies or a person. In addition, it manages the lawyers, all documentation and can generate different reports. Each lawyer may have a trainee in order to support they daily tasks such as opening, editing, and maintenance of legal processes. The system also manages part of financial functions such as: bills to pay and receive. See Appendix 7, Table 37 to detailed information regarding to the main features.

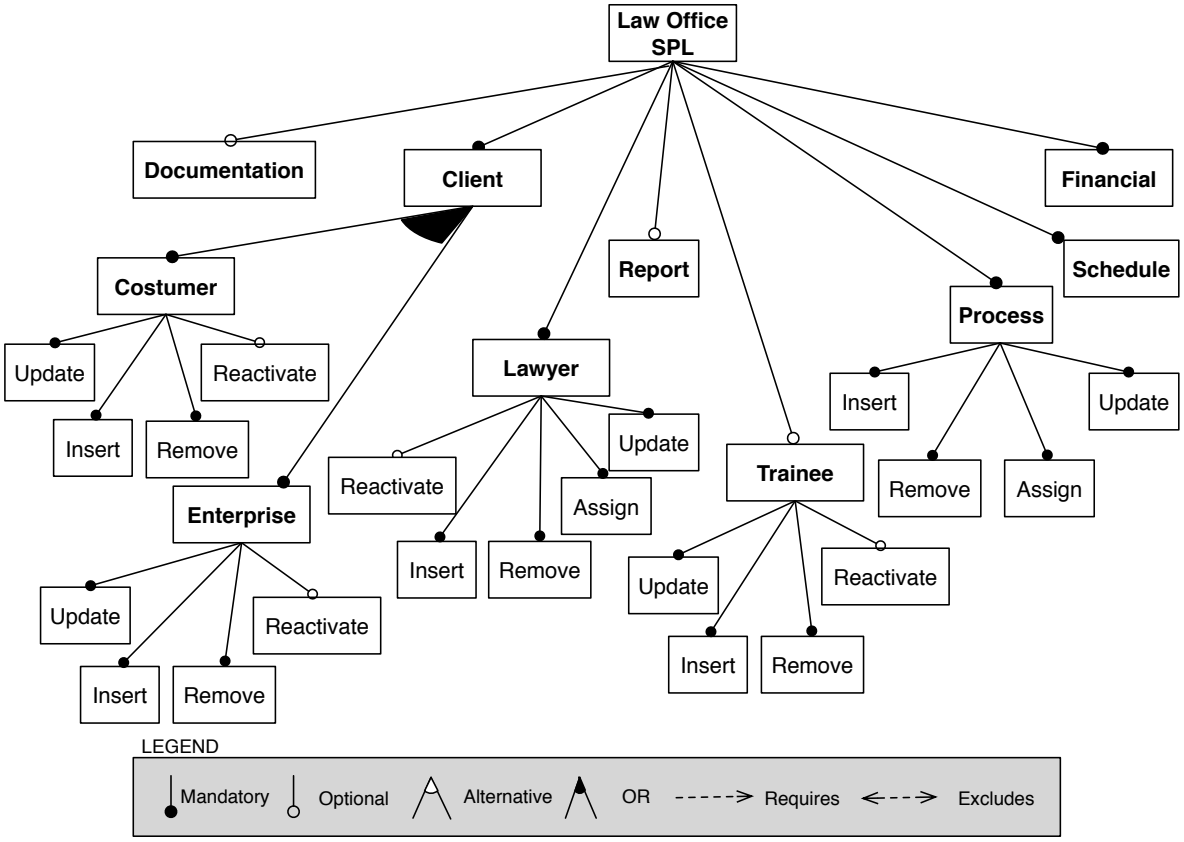

Figura 21 – Law Office SPL Feature Model.

4.4.2 Law Office SPL Refactoring and Evolution

In order to apply the security concepts and mitigate some threats during the security techniques implementation, the SPL code has also undergone some modifications and corrections. They are:

- A set of sixteen new features were added to the original Law Office SPL code in order to improve it. The main features added were Input and Output Finantial, Insert Department. In addition, a test suite was executed in order to locate and

fix bugs. This way, the SPL original code suffered some modifications regarding to feature addition and bug fixes.

4.5 Non Functional Properties

In our work, two different *security tactics*, in which can be achieved by coding, were implemented throughout fifteen different security techniques. The overall security tactics are related to how to *detect an attack*?, how to *resist an attack?*, how to *react to attack*? and how to *recovery from attack*?.

It is important to mention that all 15 techniques were implemented using conditional compilation and aspect orientation as variability mechanisms. Table 3 shows a brief description regarding each security technique implemented and Figures 22, 23 and 24 show how they are distributed over the SPLs architectures. Tables 4, 5 and 6 shows the percentage of code dedicated for software security when considering each of the security techniques. For more details regarding each technique and how they were implemented are described in the next chapter. Its code, relational model, feature model, information regarding each feature are available at: <https://github.com/pamsn/RiSEEventSPL>, <https://github.com/samiacapistrano/ufbadcc> and <www.paulosilveira.com/Office>.

Tabela 3 – Security Techniques.

ID	Security Technique	Security Tactic
T1	CheckSum	Detect Attack
T2	Hash Values	Detect Attack
T3	Verify Storage Integrity	Detect Attack
T4	MainTain Audit Trail	Detect Attack
T5	Identify Intrusion by Behavior	Detect Attack
T6	Authenticate Subject - Login/PassWord	Resist Attack
T7	Authenticate Subject - Login/Password/MachineToken	Resist Attack
T8	Authorize Subject - Authorization Pattern	Resist Attack
T9	Authorize Subject - Role Based Access Control (RBAC) Pattern	Resist Attack
T10	Authorize Subject - Multilevel security Design Pattern	Resist Attack
T11	Authorize Subject -Session Pattern	Resist Attack
T12	Authorize Subject -Reference Monitor Pattern	Resist Attack
T13	Authorize Subject - Single Access Point and CheckPoint Pattern	Resist Attack
T14	Manage Security Information	Resist Attack
T15	Hide Data by encryption	Resist Attack

4.5.1 Security techniques implementation

Once the SPLs were implemented, tested and evolved, the security techniques were implemented and integrated with them. During this integration, some decisions needed to be considered, such as: *(i)* the integration generated modifications in the SPL base relational model. This way, 45 different SQL scripts were created to hold

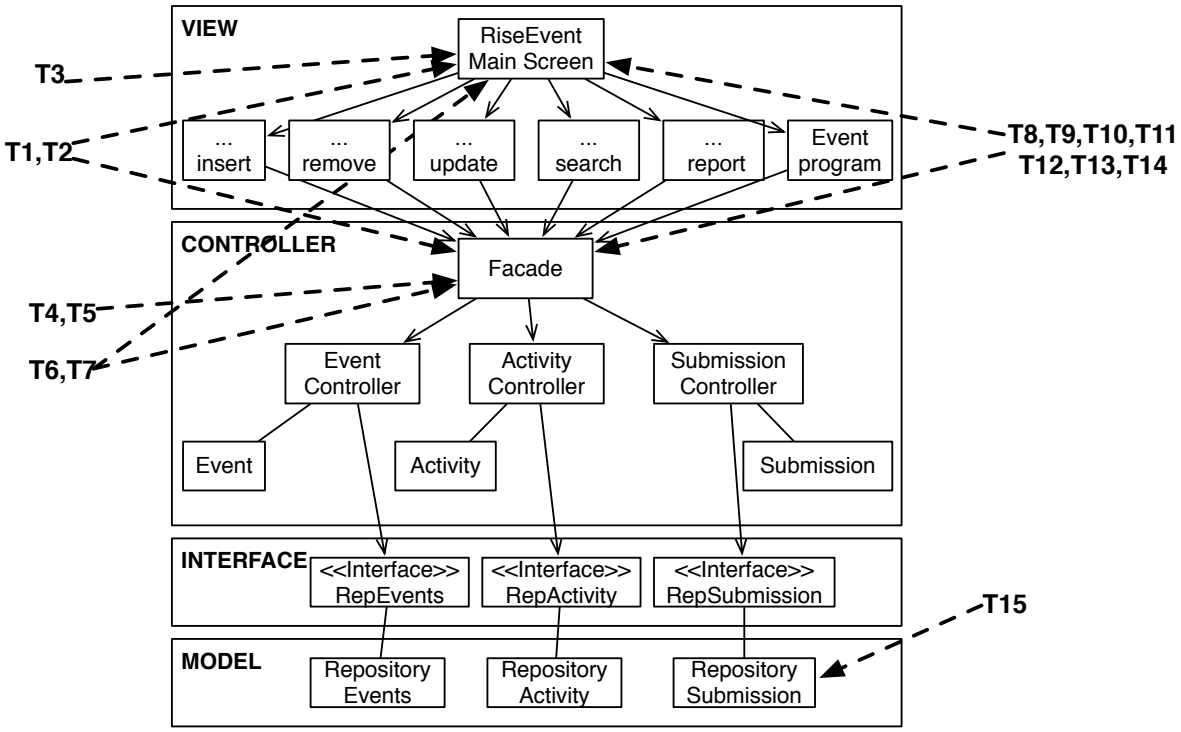

Figura 22 – Security Techniques used in the RiSE Event SPL Architecture.

Tabela 4 – Security Code Percentage in the RiSE Event SPL.

		Lines of Code	Security Code (%)			Lines of Code	Security Code (%)
	RiSEEvent	**26395**					
	T01	26886	1.86%		T01	27140	2.82%
	T02	26843	1.69%		T02	27097	2.65%
	T03	26473	0.29%		T03	26488	0.35%
	T04	27855	5.53%		T04	27680	4.86%
	T05	27853	5.52%		T05	27814	5.37%
	T06	27389	3.76%		T06	27468	4.06%
	T07	27438	3.95%		T07	27519	4.25%
CC	T08	28400	7.59%	AOP	T08	28371	7.48%
	T09	29174	10.52%		T09	29146	10.42%
	T10	30569	15.81%		T10	30534	15.68%
	T11	28442	7.75%		T11	28489	7.93%
	T12	28462	7.83%		T12	28446	7.77%
	T13	28900	9.49%		T13	28656	8.56%
	T14	29714	12.57%		T14	29684	12.46%
	T15	27448	3.98%		T15	27518	4.25%

each techniques data, the same script was used to conditional compilation and AspectJ implementations; *(ii)* in order to facilitate the data extraction we generated 31 versions of each SPL, one related to the SPL code base (without security techniques), 15 versions with security techniques implemented with conditional compilation and 15 versions with the techniques with AspectJ implementation resulting in 93 versions; *(iii)* once the techniques were plugged, some tests were executed in order to evaluate the integration between the technique and SPL code. It is important to mention that if a bug was found

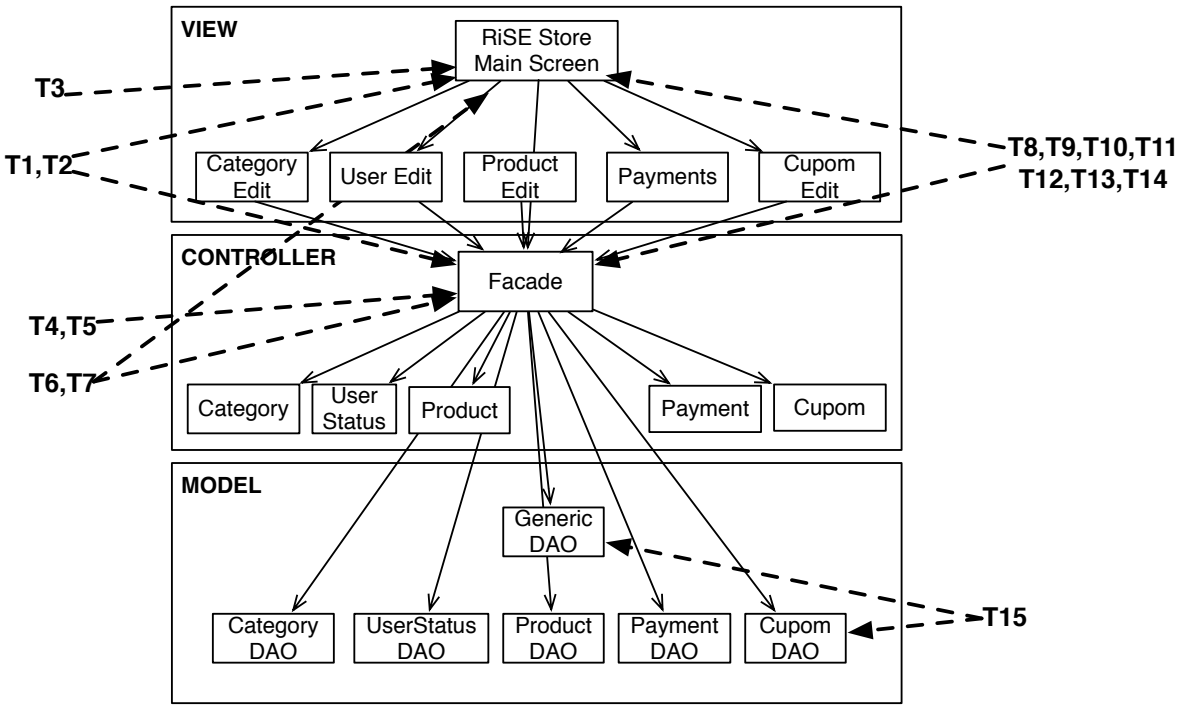

Figura 23 – Security Techniques used in the RiSE Store SPL Architecture.

Tabela 5 – Security Code Percentage in the RiSE Store SPL.

		Lines of Code	Security Code (%)			Lines of Code	Security Code (%)
	RiSE Store	**5426**					
	T01	5884	8.44%		T01	6174	13.78%
	T02	5856	7.92%		T02	6106	12.53%
	T03	5510	1.54%		T03	5525	1.82%
	T04	6951	28.10%		T04	6608	21.78%
	T05	6947	28.03%		T05	6672	22.96%
	T06	6371	17.41%		T06	6392	17.80%
	T07	6409	18.11%		T07	6428	18.46%
CC	T08	7251	33.63%	AOP	T08	7234	33.32%
	T09	8028	47.95%		T09	8012	47.65%
	T10	9359	72.48%		T10	9345	72.22%
	T11	7291	34.37%		T11	7330	35.09%
	T12	7311	34.74%		T12	7290	34.35%
	T13	7635	40.71%		T13	7528	38.73%
	T14	8567	57.88%		T14	8551	57.59%
	T15	6213	14.50%		T15	6144	13.23%

in the SPL base code, that fix was propagated over all related releases. The same holds if the bug was found in the technique code, in which the correction was propagated to the correspondent techniques code.

A summary regarding to data extracted from all 93 versions implemented can be viewed in Appendix 7. Tables 32 to 39 show the number of classes, methods, attributes, and parameters for each security technique using conditional compilation and AspectJ implemented in each SPL.

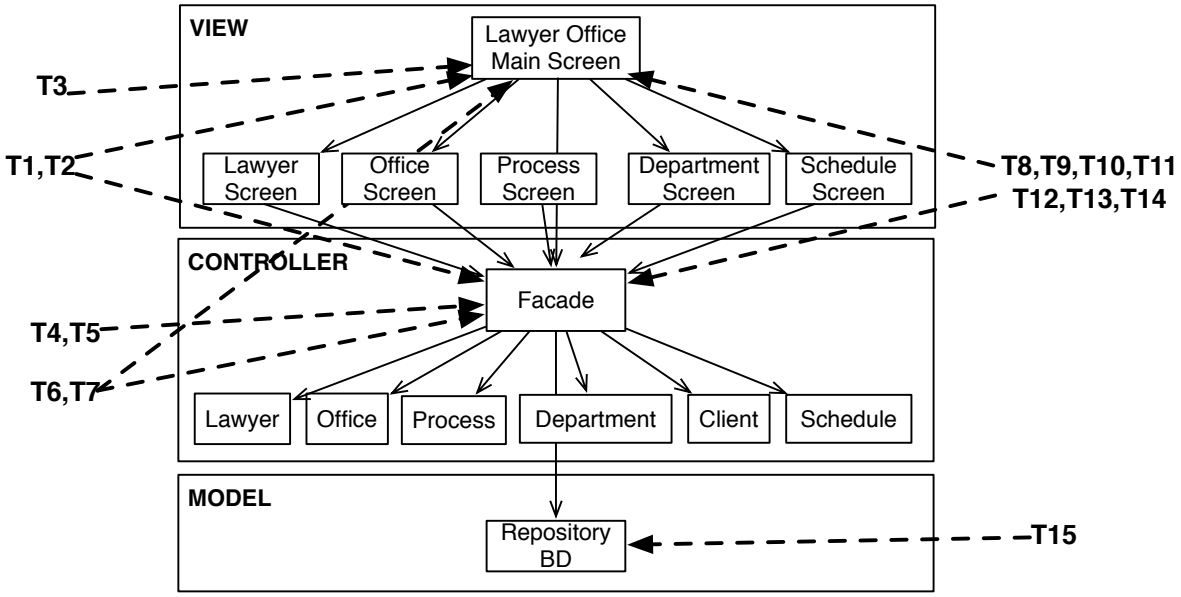

Figura 24 – Security Techniques used in the Law Office SPL Architecture.

Tabela 6 – Security Code Percentage in the Office Law SPL.

		Lines of Code	Security Code (%)			Lines of Code	Security Code (%)
	Office Law	**16665**					
	T01	17201	3.21%		T01	17641	5.85%
	T02	17149	2.90%		T02	17560	5.37%
	T03	16746	0.48%		T03	16762	0.58%
	T04	18245	9.48%		T04	17769	6.62%
	T05	18239	9.44%		T05	17835	7.02%
	T06	17570	5.43%		T06	17595	5.58%
	T07	17612	5.68%		T07	17637	5.83%
CC	T08	18436	10.62%	AOP	T08	18470	10.83%
	T09	19214	15.29%		T09	19248	15.49%
	T10	20603	23.63%		T10	20645	23.88%
	T11	18482	10.90%		T11	18569	11.42%
	T12	18496	10.98%		T12	18524	11.15%
	T13	18827	12.97%		T13	18722	12.34%
	T14	19762	18.58%		T14	19796	18.78%
	T15	17882	7.30%		T15	17619	5.72%

4.6 Chapter Summary

An increasing number of software development companies are adopting approaches or strategies which emphasize proactive reuse, interchangeable components, and multiproduct planning cycles to develop high-quality products faster and cheaper (CLEMENTS, 2001). In order to achieve all benefits, it is important to develop new techniques, approaches, tools and so on. It is increasingly important that these techniques, tools, and approaches be tested before being used in industry.

In this context, this chapter provides three SPL developed in three different

domains in which domain implementation (i.e., source code) and variability model (e.g, feature model) are publicly available. These Test Bed helps researchers and practitioners to find case studies for particular needs by providing data for all product lines.

Next chapter presents an assessment of security tactics and techniques considering maintainability internal attributes. It also analysis how the techniques are distributed regarding to its maintenance cost.

5 ASSESSING SECURITY IN SOFTWARE PRODUCT LINES: A MAINTENANCE ANALYSIS

The security research community has considered many ways to secure system parts to build secure systems or to avoid some attacks. However, few studies exist regarding to how to make secure a whole system (FERNANDEZ; ASTUDILLO; PEDREZA-GARCIA, 2015). In addition, several attacks on software systems result in a process execution deviating from its normal behavior (GAO; REITER; SONG, 2006). This way is increasingly clear the need to build safe software.

Secure systems are essentially difficult to build and as other quality attribute, they need to be considered since early phases of the software life-cycle. In (GALSTER, 2015) the authors reinforced the importance of conduct exploratory and descriptive studies to better understand variability in quality attributes, as well as, devise approaches for describing, analyzing and implementing variability in quality attributes. It becomes more critical in the Software Product Lines context, due to its complex and extensive nature. The implementation of such security properties must enable the derivation of different product line members through a Product Line Architecture (PLA), which also supports SPL maintenance by changes in its design decisions. The quality attributes requirements specify the answers of the system that, should achieve the goals previously defined. According to (BASS; CLEMENTS; KAZMAN, 2012) a useful software architecture approach for build secure systems is based on tactics, defined as being "a design decision that influences the achievement of quality attribute response – tactics directly affect the system's response to some stimulus. "(BASS; CLEMENTS; KAZMAN, 2012). Although these tactics are largely discussed by different studies (WOODS; ROZANSKI, 2005; ROZANSKI; WOODS, 2005; TAYLOR; MEDVIDOVIC; DASHOFY, 2009; BAGHERI; SULLIVAN, 2011; FERNANDEZ; ASTUDILLO; PEDREZA-GARCIA, 2015), few of them discuss how to achieve them. The lack of studies reporting how to implement a tactic can be a problem since the software architecture should make a decision regarding to this implementation, which could lead to misinterpretations (FERNANDEZ; ASTUDILLO; PEDREZA-GARCIA, 2015).

In this context, our study contributes to the research and practice in the following way: *(i)* providing information regarding to how to implement different security tactics and techniques; *(ii)* comparing two variability implementation mechanism, namely aspect-oriented programming and conditional compilation. *(iii)* describing how those security techniques and variability implementation mechanism may affect software maintainability throughout the following internal attributes: Code size, Separation of concerns, cohesion and coupling; and *(iv)* providing empirical evidence on why security

nonfunctional properties vary; what are the possible variants and how it is mapped in the architecture.

The remainder of this chapter is organized as follows: Section 5.1 provides an overview about conditional compilation as variability implementation mechanism. Section 5.2 details some ways to achieve security at software architecture level. Section 5.3 describes the family of experiments used. Section 5.5 details how the study was planned regarding to its context, hypotheses, by defining the goal, research questions and the metrics. Section 5.6 and Section 5.7 describe the study operation and the statistical tests and methods used. Section 5.8 presents the results and hypotheses testing analysis. The main finding are summarized in Section 5.9, Sections 5.10 and 5.11 discuss the threats to validity and related work, respectively. Finally, Section 5.12 draws conclusions about the study.

5.1 Variability Mechanism

In this section, we presented some concepts related to the techniques evaluated in this study: Conditional Compilation (CC) and Aspect-Oriented Programing (AOP). The main goal is to evaluate the different composition mechanisms (GACEK; ANASTASOPOULES, 2001) available to understand their strengths and weaknesses regarding to maintainability (SANT'ANNA et al., 2003; GARCIA et al., 2005; KULESZA et al., 2006a; GREENWOOD et al., 2007; EADDY et al., 2008; FIGUEIREDO et al., 2008; FIGUEIREDO et al., 2009; DANTAS; GARCIA, 2010; CARVALHO, 2016; ARVANITOU et al., 2017) based on the separation of concerns principle and other software attributes such as: separation of concerns, coupling, cohesion and size.

As the context of this thesis is in the implementation of security in software product lines, thus we decided to use variability mechanisms that provide support to map variability concerns into core assets represented as features. The idea of these mechanisms is to cover security patterns and best practices based on compile time configuration required in software product lines environment. It allows products to be derived considering only the code needed for such implementation.

5.1.1 Conditional Compilation

There are several implementation approaches for coding variabilities in product lines, as characterized and compared in (GACEK; ANASTASOPOULES, 2001; BOSCH; CAPILLA, 2013; SVAHNBERG; GURP; BOSCH, 2005). The conditional compilation is one of them and was used in this study since it is a well-known mechanism to isolate and identify crosscutting concerns code. It is also commonly used in both scenarios, the state-of-the-practice in industry (ALVES et al., 2006b; CHEN; BABAR, 2011; FERREIRA et al., 2014) and also literature (GACEK; ANASTASOPOULES, 2001; LEE; HWANG,

2014). It is powerful since enables control over code segments to be included or excluded from a program compilation. The major advantage of this mechanism is the encapsulation of multiple implementations in a single module. The desired functionality is selected by defining the appropriate conditional compilation directive marks in different locations in the code. In addition, the code can be marked at different granularities and architecture layers, from a single line of code to a whole file. Listing 5.1 shows the use of conditional compilation mechanism through the addition of *#if-defs* pre-processing marks to add code regarding to cryptography. If the cryptography feature is selected to be part of a product, the code responsible to decrypt password and email information should be included in that product.

A configuration file composed of #if-defs tags always associated with a boolean value is defined grouping all features of a given product. These values indicate the presence of the feature in the product, including the bounded piece of code in the compiled product.

Código 5.1 – Conditional Compilation pre-processors directives example.

```
1  ...
2              Statement statement = (Statement) pm.getCommunicationChannel();
3              ResultSet resultset  = statement.executeQuery("Select * from
4                  login  WHERE idUser =" + idUser);
5              if  (resultset.next())  {
6                  String  password = resultset.getString("password");
7                  String  email =  resultset.getString("email");
8                  //#if  ${Cryptography} == "T"
9              try  {
10                 password = MyCrypto.decrypt(password);
11                 email = MyCrypto.decrypt(email);
12             } catch  (Exception e) {
13                 e.printStackTrace();
14             }
15                 //#endif
16                 login.setIdUser(resultset.getInt("idUser"));
17                 login.setPassword(password);
18                 login.setEmail(email);
19 ...
```

5.1.2 Aspect-Oriented Programming

The basic abstractions of object-oriented software development are classes, objects, interface, methods, and attributes. However, it may not be enough to separate crosscutting concerns found in complex systems and naturally cut across the modularity of other concerns. In this context, the aspect is the main mechanism of modularity,

which encapsulates a concern code that would be tangled with and scattered across the code of other concerns. AspectJ (KICZALES et al., 2001) is a practical aspect-oriented extension to Java programming language which uses: *join points:* well-defined points in the program flow, such as method calls, field sets, and so on. *Pointcuts* which describes join points and values at those points and finally the *advices* which are a method-like abstraction that define code to be executed when a join point is reached. The AOP has emerged over the previous decade as the domain of systematic exploration of crosscutting concerns and corresponding support throughout the software development process (FILMAN et al., 2004). In addition to the aforementioned qualities, the aspect-oriented programming is largely used in academy as can be seen in (SANT'ANNA et al., 2003; GARCIA et al., 2005; GREENWOOD et al., 2007; FIGUEIREDO et al., 2008; KULESZA et al., 2006b; KASTNER; APEL; BATORY, 2007; D'AMORIM; BORBA, 2012; FERREIRA et al., 2014) and used in security frameworks such as Spring Security[1].

Listing 5.2 shows how the aspect can modularize the logging feature. The aspect usually needs to provide interception point in the base code in order to get the code adequately waved. Lines 2-4 show an example of intercepting the execution before any method located at RiSEventFacade class. Lines 6-14 show how and what will be executed after that interception point (pointcut).

Código 5.2 – Example of variability mechanism with AOP (aspect).

```
1
2  public aspect Logging{
3          pointcut logging():
4          execution(void RiSEventFacade.*(*))) && args();
5
6          void around():logging(){
7              String name = thisEnclosingJoinPointStaticPart
8              .getSignature().toShortString();
9              DateFormat dateFormat = new SimpleDateFormat
10             ("yyyy/MM/dd HH:mm:ss");
11             Date date = new Date();
12             ((RiSEventFacade)thisJoinPoint.getThis())
13             .logs.insert (RiSEEventLoginScreen.getLoggedUser()
14             .getIdUser(),name,dateFormat.format(date));
15         }
16 }
```

[1] projects.spring.io/spring-security

5.2 Software Security

Although software architecture is considered the central part and is being widely discussed (ALMEIDA; OQUENDO, 2013; AMORIM; ALMEIDA; MCGREGOR, 2013; AMORIM et al., 2014; AMORIM; ALMEIDA; MCGREGOR, 2014) by the research community, few attention is given to its security (FERNANDEZ; ASTUDILLO; PEDREZA-GARCIA, 2015), especially involving software product lines. A useful software architecture approach for building secure system is based on tactics, which were originally introduced in 2003 (BASS; CLEMENTS; KAZMAN, 2003). By tactic, we mean a design decision with respect to a quality factor. Throw these architectural tactics, it is possible to codify and record best practices for achieving some quality attributes (FERNANDEZ; ASTUDILLO; PEDREZA-GARCIA, 2015).

The research community has spent considerable effort to benchmark, examine and classify security tactics (FERNANDEZ; ASTUDILLO; PEDREZA-GARCIA, 2015; BASS; CLEMENTS; KAZMAN, 2003; RYOO; LAPLANTE; KAZMAN, 2012). In (FERNANDEZ; ASTUDILLO; PEDREZA-GARCIA, 2015), the authors made a valuable contribution to this area (see Figure 25), by reclassifying and pruning the existing security tactics and providing important information in order to perform each of the tactics. It is important since the tactics do not provide enough guidance to architects in order to achieve the desired objective.

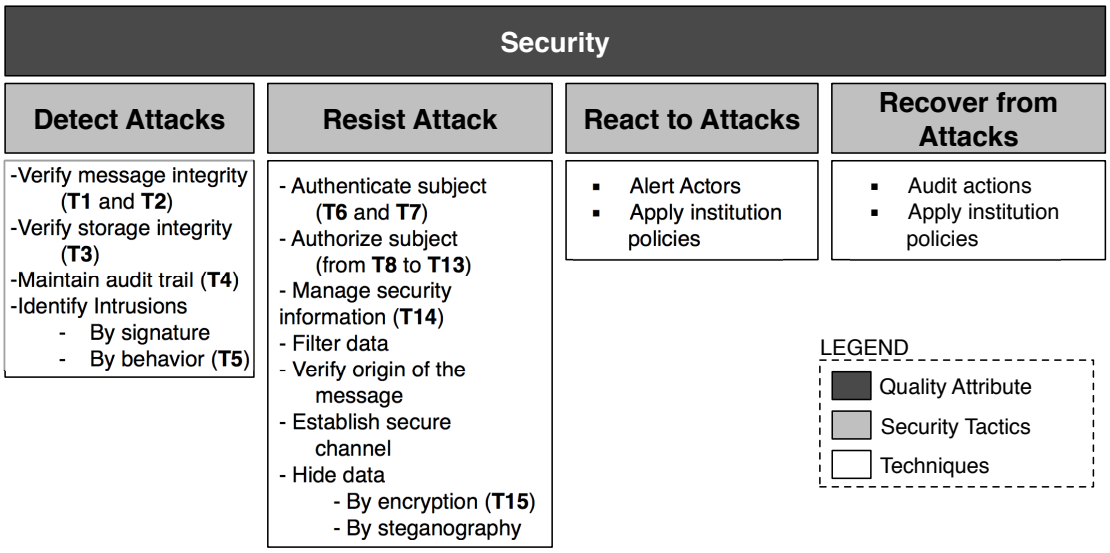

Figura 25 – Security Tactics and Techniques (FERNANDEZ; ASTUDILLO; PEDREZA-GARCIA, 2015).

Besides the use of tactics, the security can rely on patterns and frameworks each of which employed during architectural design. There are catalogs of patterns for specific quality attributes: availability, interoperability, performance, usability, security,

and so on (CERVANTES et al., 2016). Regarding to security, they can be classified as technology-agnostic and technology-specific. A technology-agnostic pattern catalog describes security patterns for concerns such as identity management, authentication, access control, secure process management, web services security, and cloud computing (FERNANDEZ-BUGLIONI, 2013). While a catalog, which is Java specific, describes 23 security patterns associated with different application layers (STEEL; NAGAPPAN; LAI, 2006). Based on recurring concerns the frameworks provide a generic solution to be reusable in different contexts (CERVANTES; VELASCO-ELIZONDO; KAZMAN, 2013). Some examples of security frameworks are: Spring Security[2], Apache Shiro[3] and JGuard[4]. Once we are interested in evaluating how CC and AOP impact on maintainability when used to implement security techniques. We chose not to use any framework due to: *(i)* we would like to analyze each solution in detail, *(ii)* the framework can hide some implementation details, *(iii)* may use implementation mechanisms different from what we would evaluate, and *(iv)* did not consider the code from basic entities to graphical user interface.

In this thesis, a subset of tactics refinement performed by (FERNANDEZ; ASTUDILLO; PEDREZA-GARCIA, 2015) was selected based on the SPL domain constraints and technology, such as: *(i)* the security tactic should be achieved code implementation and *(ii)* the implementation should make sense in the domains of the three created SPLs. Next, we describe each tactic used in our study (from T1 to T15), and detailing how it was implemented. The implementation of the security techniques was based on class diagrams (FERNANDEZ-BUGLIONI, 2013; D.; M.; C., 2006, 2006), techniques narrative description (BASS; CLEMENTS; KAZMAN, 2012) and interview with two developers with seven years of experience in software for the Brazilian government. In order to better present and keep track of our results, all techniques to implement a tactic were identified as T_n.

- *Verify Message Integrity:* It stands to the derived products to have its own way of applying cryptography. This tactic employs techniques such as a checksum or hash values to verify the integrity of messages, resource files, deployment files and configuration files. It allows the software to detect attacks even if a slight change is performed in the original files or messages. In order to achieve it, we developed an algorithm in order to generate checksums (T1) and hash values (T2) for each transaction performed with the database and configuration files. If the database suffers some attack, the system will detect and signal the attack.
- *Verify Storage Integrity (T3):* It indicates the need to define measures to make

[2] projects.spring.io/spring-security
[3] shiro.apache.org
[4] jguard.xwiki.com

sure that databases have not been modified. In our context, it was performed by coding a function to compare the current database state with the last valid database state. If some difference is detected, the system indicates a probable attack.

- *Maintain Audit Trail (T4):* It is a system function that can be used to detect attacks. The system audits, keeps a record of users and system actions and their effects to help trace the actions of, and to identify, an attacker. If the attack is confirmed, an appropriate action should be taken to recover from the attack. It can be achieved by restoring the database to the last valid state.

- *Identify intrusion by behavior (T5):* Every time when a user performs some operation in the system, a weight is assigned to it. Then this behavior is documented to be further analyzed (GAO; REITER; SONG, 2006), in order to detect a deviation from the utilization pattern of that user. This way, based on a sensibility parameter, the software architect can identify if a certain activity is suspect or not.

- *Authenticate subject:* It ensures that an authenticated actor (a user or a remote computer) has the rights to access and modify either data or services to guarantee that an actor is actually who what is supposed to be. Passwords, on-time passwords, digital certificates, and biometric identification can be used for the authentication. This tactic was performed in two different ways, as following:

  - *Login/Password (T6):* the identification and authentication service obtains an identity from the actor, translates the identity to an ID, and authenticates the ID using an authenticator. It is the most common log-on scenario for computer systems (SCHUMACHER et al., 2005).
  - *Login/Password/Machine Token (T7):* one of the alternatives largely used to increase the system security is the combination of Login/Password approach with a Machine token. It is achieved by registering a unique identification of the machine which will be used to access the system. This way, a machine token is generated and registered with its correspondent Login/Password, avoiding not allowed access.

- *Authorize subject:* It describes who is authorized to access the resources of the system (D.; M.; C., 2006). This mechanism is usually enabled by providing some access control mechanisms for a system (SCHUMACHER et al., 2005). In order to achieve it, we implemented six different security patterns (SCHUMACHER et al., 2005; D.; M.; C., 2006; FERNANDEZ-BUGLIONI, 2013), as following:

- *Authorization Pattern (T8):* The active entities are subjects which interact with the system in order to access passive resources. This relationship between subject and object describes which subject is authorized (i.e. have rights) to access certain objects.
- *Role Based Access Control(RBAC) Pattern (T9):* Users and Roles are represented by classes which describe registered users and predefined roles, respectively. This way, the authorization pattern is represented by the combination of role, protection object and rights.
- *Multilevel Security Design Pattern (T10):* It is applied when we need several levels of security classifications to be applied for both subjects, and objects. This way, the subjects and objects can be categorized into different levels and then classified. It can be expensive since subjects and objects need to be classified into certain levels of sensitiveness.
- *Session Pattern (T11):* A subject can be in several sessions at the same time and it has a limited lifetime. When we start a session, the subject only activates a set of authorization contexts assigned to him/her, then, making only his/her rights available at that session.
- *Reference Monitor Pattern (T12):* This pattern enforces declared access restrictions when an active entity requests resources. This way, it describes how to define an abstract process which intercepts all requests for resources and checks them for compliance with authorizations. These authorizations rules indicate a collection of authorization rules organized as ACLs (Access Control Lists).
- *Single Access Point (SAP) and CheckPoint (T13):* The application of SAP avoids external entities from communicating directly with components in the system. All incoming traffic is routed through one channel, where monitoring can be easily performed. It enables the capture of information regarding to the parties (subject) currently accessing the system. In order to analyze all incoming petitions and messages, the Check Point pattern was applied. For this reason, SAP is predestined to be combined with Check Point for all messages to be supervised (D.; M.; C., 2006).

An important framework which also implements security is the Spring Security a powerful and highly customizable authentication and access-control framework used in web applications [5].

- *Manage security information (T14):* It includes the management of keys for cryptography, the secure storage of authorization rules, and other ways to manipulate

[5] <http://projects.spring.io/spring-security/>

security information (FERNANDEZ; ASTUDILLO; PEDREZA-GARCIA, 2015). It was achieved by encrypting only the security information such as rules, rights, login, and password.

- *Hide Data by encryption (T15):* Data should be protected from unauthorized access. Confidentiality is usually achieved by applying some form of encryption. According to (FERNANDEZ; ASTUDILLO; PEDREZA-GARCIA, 2015), it can be achieved by cryptography and steganography. In our context, all the data stored in the database was previously encrypted, except the primary and foreign keys.

5.3 The Family of experiments

It is a consensus that to achieve greater validity and generalization of experimental results replications are essential. The concept of replication is smoothly extended to that of "family of experiments"(BASILI; SHULL; LANUBILE, 1999; HADAR et al., 2013). In (SHULL et al., 2008), the authors identified two types of replications: *exact replications* when the procedures of an experiment are followed as closely as possible, and *conceptual replications*, in which the same research questions are evaluated by using a different experimental procedure. Our study reuses the original procedures, e.g., the study design and steps, but change the subject in order to gain insight into the original results, which makes it be classified as an exact replication. It is important to mention that replications which provide similar results to those found in the original experiment are as useful to the community as a replication that produces results different from those of the original experiment. The replications with different results are important since can help the community to understand why the results were different (SHULL et al., 2008).

Figure 26 shows the main characteristics of the family of experiments, including the context of each experiment, the numbers of each Testbed used, the attributes evaluated and the order in which it was carried out. It is important to mention that the testbeds were developed by different developers with different skills and know how, as well as, different technologies. In addition, the replications are important to avoid any kind of threat to validity regarding the way in which the testbed or security techniques were implemented, adding a certain reliability to the results.

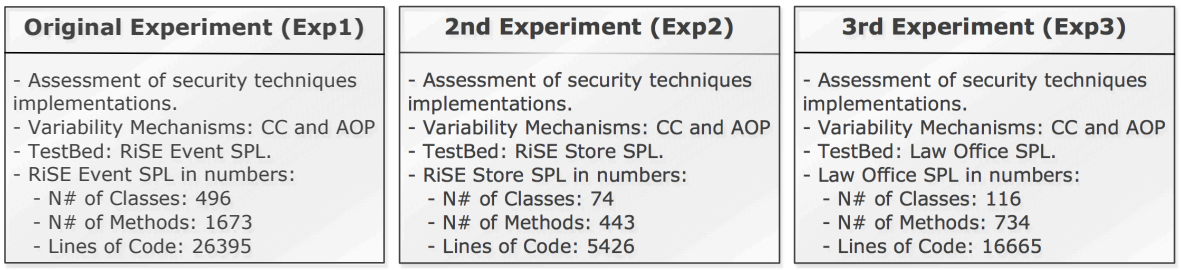

Figura 26 – The family of conducted experiments.

5.4 Experiments Definition

All three experiments were carried out by following the guidelines provided by (JEDLITSCHKA; PFAHL, 2005; WOHLIN et al., 2012). Section 5.5 details the planning of the original experiment, while the replications are presented in terms of differences with respect to the original. All experiment material and results are available at Appendix 7.

5.5 Planning

In order to setup our quantitative analysis, it is important to systematically build its planning and execution to allow further replications and extensions. This way, controlled experiments guidelines (IIT; SINGER, 1999; WOHLIN et al., 2012; KITCHENHAM et al., 2002; JEDLITSCHKA; PFAHL, 2005; JURISTO; MORENO, 2010) applied by the Software Engineering community were used to support this systematization. Next Sections describe how the studies were conducted. Table 7 summarizes the main elements.

5.5.1 Goal

The Goal/Question/Metric (GQM) method proposed by (BASILI; SELBY; HUTCHENS, 1986; BASILI; CALDIERA; ROMBACH, 1994) was adopted in order to state the goal of the quantitative study (WOHLIN et al., 2012), as described following:

The goal of this study is to ***evaluate*** different techniques to implement security tactics ***for the purpose of*** assessing conditional compilation and aspect-oriented programming as variability mechanisms ***with respect to*** maintainability through size, separation of concerns, cohesion and coupling attributes ***from the point of view of***software architects ***in the context of*** Software Product Lines projects.

Moreover, this study is also important to identify the most suitable security technique which implies on the reduction on code size, separation of concerns, coupling and cohesion increasing the benefits of reuse and maintenance activities of components,

Tabela 7 – A summary of the experiments.

Goal	The goal of this study is to ***evaluate*** different techniques to implement security tactics ***for the purpose of*** assessing conditional compilation and aspect-oriented programming as variability mechanisms ***with respect to*** maintainability by accessing code size, separation of concerns, coupling and cohesion ***from the point of view of*** software architects ***in the context of*** Software Product Lines projects.
Question	What is the impact on Maintainability when evaluating security techniques implemented using conditional compilation and aspect-oriented programming?
Metrics	- Size - Separation of Concerns - Coupling - Cohesion.

beyond the implementation of security. To assess the measurement object, a set of metrics was defined for both variability mechanisms to attend the following requirements: *(i)* measure quality factors (size, separation of concerns, cohesion and coupling), *(ii)* identify advantages and drawbacks from the use of aspect in comparison to object-oriented design and *(iii)* classify the security techniques regarding their impact on quality factors.

5.5.2 Research Questions (RQs)

In order to achieve the stated goal, some *quantitative* questions were defined and described in the following:

RQ1. What is the impact on Maintainability when evaluating security techniques implemented using conditional compilation and aspect-oriented programming? It aims to understand how security techniques impact and how they are distributed regarding Size, Separation of Concerns, coupling, and Cohesion to support software architects design decisions. In addition, it provides important insights regarding the most suitable way to implement the security techniques.

The results reported in this study can be useful for both researchers and practitioners. Researchers can identify useful research directions and get guidance on how the security techniques impact on maintainability. On the other hand, practitioners can benefit from this study by identifying the variability implementation mechanism with low impact on maintainability, as well as, learning concrete techniques to implement security tactics at the code level.

5.5.3 Metrics

In order to answer the previously defined research question, a set of metrics must be computed to understand the size, separation of concerns, coupling and cohesion of the security techniques when using conditional compilation and aspect-oriented programming. According to (JURISTO; MORENO, 2010) a more effective way of software engineers understand and interpret the collected data is associating the internal and external attributes with metrics through a measurement framework. Figure 27 shows the measurement framework used in our research which was adapted based on definitions of quality attributes (IEEE. . . , 1990; SOFTWARE. . . , 2001; BASS; CLEMENTS; KAZMAN, 2012) and existing quality models (SANT'ANNA et al., 2003; CARVALHO, 2016).

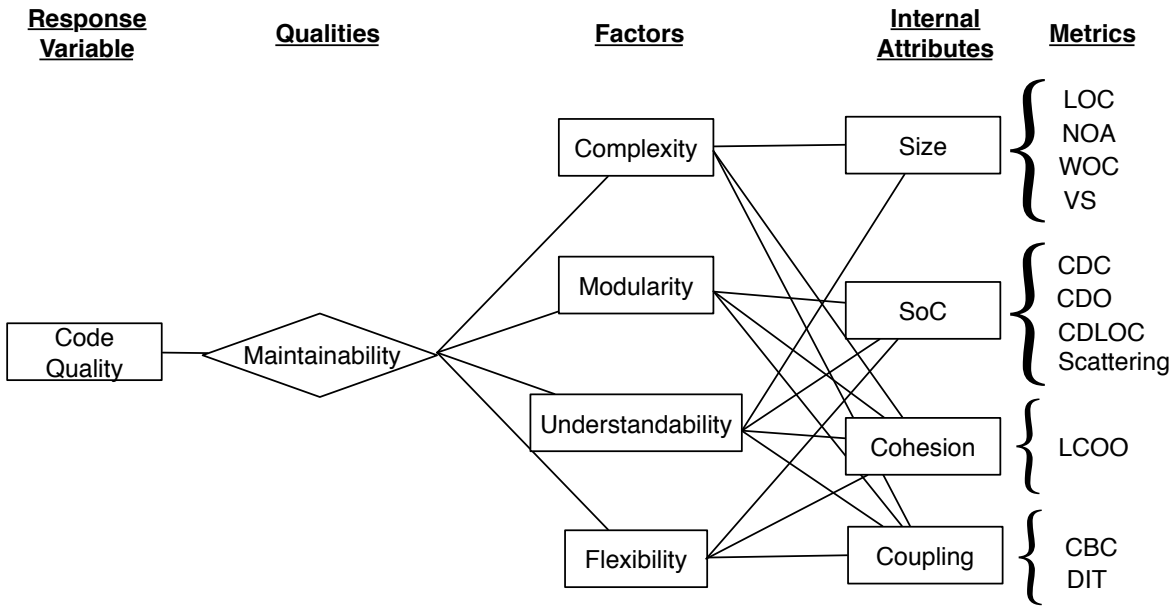

Figura 27 – The Quality Model.

Although the literature provides a large number of metrics associated with software quality (DEPT, 1998; NUNES et al., 2009; SARAIVA et al., 2015), most of the assessments of quality attributes is usually performed by a single metric, rather than a combination of multiple metrics (ARVANITOU et al., 2017). Based on the internal attributes derived found in the measurement framework, we identified a set of metrics which captures information about the design and code in terms of *coupling*, *cohesion*, *size* and *separation of concerns*. We decided to apply some well-known metrics and already empirically validated in previous study (SANT'ANNA et al., 2003; GARCIA et al., 2005; KULESZA et al., 2006a; GREENWOOD et al., 2007; EADDY et al., 2008; FIGUEIREDO

et al., 2008; FIGUEIREDO et al., 2009; DANTAS; GARCIA, 2010; CARVALHO, 2016; ARVANITOU et al., 2017; NUñEZ-VARELA et al., 2017). It is also important to mention that to evaluate modularity factor three internal attributes (separation of concerns, coupling, and cohesion) were evaluated since cohesion and coupling alone are not enough to evaluate modularization (CANDELA et al., 2016). They were also adapted and validated to aspect-oriented programing, allowing us to perform comparative studies (SANT'ANNA et al., 2003). It is important to mention that some metrics were improved from the original definition (e.g., CDLOC and CDO) by considering hook methods in their calculation and other (e.g., Scattering) was added since it brings the number of line of code perspective for separation of concerns when compared to the metric proposed in the original quality model. Next, the metrics are described regarding to their definition and importance (SANT'ANNA et al., 2003).

Separation of Concerns (SoC): it is a fundamental principle related to the decomposition mechanism used for both software design and implementation. Concerns are a known way to decompose the software into smaller units, and at the same time more manageable and comprehensible.

In this study we used three metrics related to the concerns (SANT'ANNA et al., 2003), they are:

- *Concern Diffusion over Components(CDC):* it counts the number of classes, interfaces, and aspects whose main purpose is to contribute to the implementation of a concern. Furthermore, it counts the number of components that access the primary components by using them in attribute declarations, formal parameters, return types, throws declarations and local variables, or method calls. It indicates how much spread is the concerns over the project code. The less scattered the concern is, the easier it will be to understand it.

- *Concern Diffusion over Operations(CDO):* it counts the number of methods and advices whose main purpose is to contribute to the implementation of a concern (SANT'ANNA, 2004). In addition, it counts the number of methods and advices that access any primary component by calling their methods or using them in formal parameters, return types, throws declarations and local variables. Constructors and abstract methods also are counted as operations. As the previously described metric, the CDO also indicates the concern code scattered degree in terms of operations. This way, as more operations affected by the concern, as more difficult it will be to understand it. It is important to mention that this metric suffers an improvement since the hook methods used in aspect-oriented programming were also computed. Hook methods are empty methods placed in the base code for later extension. They have no purpose other

than to provide a join point inside a method that can be extended by an aspect (KASTNER; APEL; BATORY, 2007).

- *Concern Diffusion over LOC (CDLOC)* it captures how the code of a concern is mixed with the code of another concern. It counts the number of transition points for each concern to another concern through the lines of code. Transition are points in the code where there is a "concern switch". The more the code of a concern is mixed and entangled with another concern, the harder will be to understand it. It is important to mention that this metric suffers an improvement since the hook methods used in aspect-oriented programming were also computed.

- *Scattering:* It captures how the code of a concern (security technique) is spread over the code of another concern (SPL code). It counts the number of lines of code which does not belong to the security technique class or aspect. It brings a number of lines of code scattered dimension for CDLOC metric.

Coupling: it is related to how closely connected two components are, it captures the strength of the relationships among components. The two metrics used to evaluate this characteristic were defined as follows:

- *Coupling Between components(CBC):* it measures the number of other components with which it is coupled. This metric is an extension of coupling metric (CBO) defined by (CHIDAMBER; KEMERER, 1994) and adapted by (SANT'ANNA et al., 2003) to deal with new coupling dimensions in aspect-oriented software design. The adaptation aims to: *(i)* access aspect methods and attributes introduced by inter-type declarations; *(ii)* and the relationships between aspects and classes or other aspects defined in the pointcuts (SANT'ANNA et al., 2003). For each component (classes or aspects), it counts the number of other classes used in attributes declaration, formal parameters, return types and local variables. In addition, for each aspect, it also counts: the classes in which attributes and methods are injected by using inter-type declaration, the classes, and aspects intercepted by the aspect through pointcut definitions and the components in which their attributes and methods are introduced by inter-type declaration and accessed by the aspect. In summary, the greater the number of couplings of a component, the harder it will be for it to be understood. In order to improve code modularity, the coupling among components should be reduced.

- *Depth Inheritance Tree (DIT):* it calculates the distance from class object/aspect in the inheritance hierarchy. This metric originally defined by (CHIDAMBER; KEMERER, 1994), and improved by (SANT'ANNA et al., 2003) to consider the

inheritance between aspects. It is defined as the maximum length from the node to the root of the tree, by measuring how many ancestor classes/aspects can potentially affect this class/aspect. The deeper is the inherit tree the greater complexity design the software has, difficulting the understanding and maintenance.

Cohesion: measures the strength of relationship among pieces of functionality within a given component. This property is assessed in our study by means of the following metric:

- *Lack of Cohesion over Operations(LCOO):* Adapted from (CHIDAMBER; KEMERER, 1994), LCOO metric measures the lack of cohesion over operations. This metric deal with aspect advices and methods in the same way as LCOM deals with methods which belong to a class. Considering all pairs of class's methods LCOM measures the lack of cohesion in methods by calculating the difference between the number of method pairs that do not share a field access with the number of method pairs that do. In addition to the advices and internal methods to the aspect, it also considers the methods that the aspect introduces in the classes affected by inter-type declaration (SANT'ANNA et al., 2003). The lack of cohesion increases complexity and means that the component should be split in two or more sub-components.

Size Metrics: These metrics show the different facets of system size. They are following described:

- *Lines of code (LOC):* it counts the total lines of code considering non-blank and non-comment lines in a compilation unit. The greater the number of lines of code more difficult to maintain and understand the code.
- *Number of attributes (NOA):* it measures the internal vocabulary of a component, by counting the number of attributes of each class and aspect. Regarding to the internal attributes inside an aspect, this metric also considers the attributes introduced by inter-type declarations. The higher the number of attributes per component, more difficult to understand the system.
- *Weighted Operations per Component(WOC):* it measures the complexity of a component in terms of its operations. The complexity of an operation is given by the number of parameters of that operation. The complexity of an operation without parameters is one, an operation with one parameter is two and so on. It means that operations with more parameters than another are more complex. In the aspect, in addition to advices and methods inside the aspect, it also counts

methods injected by the aspect through inter-type declaration. The greater the number and complexity of operations component, more difficult it is to understand and maintain it.

- *Vocabulary Size (VS):* Counts the number of classes, interfaces, and aspects of the system. Each component name is counted as part of the system vocabulary. The instances are not considered in this score. The greater the vocabulary size, more difficult to maintain and understand it.

5.5.4 Hypotheses and variables

The basis for the statistical analysis of an experiment is the hypothesis testing. If the hypothesis can be rejected, then conclusions can be drawn based on the hypothesis testing under given risks (WOHLIN et al., 2012). Based on the informal statement of the hypotheses presented by the research questions, as well as, the metrics previously presented, we are now able to state them formally and also define what measures we need to evaluate the hypotheses. Thus, three hypotheses were setup to formed the basis for the design of this quantitative study. Each hypothesis defined is responsible to analyze the comparison of internal quality attributes (Size, Separation of concerns, coupling, and cohesion) regarding to CC vs. AOP, Authorization Techniques and Detect Attack vs. Resist Attack Techniques respectively. While the *Null Hypotheses (H_{0n})* define that the experiment did not produce any real trends or patterns (WOHLIN et al., 2012), the *Alternative Hypotheses (H_{1n})* are those that reject the null hypotheses (WOHLIN et al., 2012). Due to size limitation, the hypotheses were grouped using the mathematical module symbol. This way, to the (H_{01}) we are interested in evaluating if the security techniques implementations using CC had no difference compared to AOP implementation when considering each of the internal attributes (Size, Separation of Concerns, Cohesion, and Coupling). They are described following:

Null Hypotheses (H_{01}): There is no statistical significant difference in size, separation of concerns, coupling and cohesion when comparing the security techniques implemented using conditional compilation (CC) and aspect-oriented programing (AOP) (from T1 to T15).

$$H_{01}: \left| \begin{array}{c} Size_{CC\ (T1...T15)} \\ Separation of Concerns_{CC\ (T1...T15)} \\ Cohesion_{CC\ (T1...T15)} \\ Coupling_{CC\ (T1...T15)} \end{array} \right| = \left| \begin{array}{c} Size_{AOP\ (T1...T15)} \\ Separation of Concerns_{AOP\ (T1...T15)} \\ Cohesion_{AOP\ (T1...T15)} \\ Coupling_{AOP\ (T1...T15)} \end{array} \right|$$

Alternative Hypotheses (H_{11}): the alternative hypotheses determine that there is the statistical significant difference in size, separation of concerns, coupling and cohe-

sion when comparing security techniques implemented using conditional compilation (CC) and aspect-oriented programing (AOP).

$$H_{11}: \left| \begin{array}{c} Size_{CC\ (T1...T15)} \\ Separation of Concerns_{CC\ (T1...T15)} \\ Cohesion_{CC\ (T1...T15)} \\ Coupling_{CC\ (T1...T15)} \end{array} \right| \neq \left| \begin{array}{c} Size_{AOP\ (T1...T15)} \\ Separation of Concerns_{AOP\ (T1...T15)} \\ Cohesion_{AOP\ (T1...T15)} \\ Coupling_{AOP\ (T1...T15)} \end{array} \right|$$

Null Hypotheses (H_{02}): There is no statistically significant difference on size, separation of concerns, coupling and cohesion when comparing the authorization techniques (from T8 to T13).

$$H_{02}: \left| \begin{array}{c} Size_{CC\ and\ AOP\ (T8...T13)} \\ Separation of Concerns_{CC\ and\ AOP\ (T8...T13)} \\ Cohesion_{CC\ and\ AOP\ (T8...T13)} \\ Coupling_{CC\ and\ AOP\ (T8...T13)} \end{array} \right| = \left| \begin{array}{c} Size_{CC\ and\ AOP\ (T8...T13)} \\ Separation of Concerns_{CC\ and\ AOP\ (T8...T13)} \\ Cohesion_{CC\ and\ AOP\ (T8...T13)} \\ Coupling_{CC\ and\ AOP\ (T8...T13)} \end{array} \right|$$

Alternative Hypotheses (H_{12}): There is a statistically significant difference on size, separation of concerns, coupling and cohesion when comparing the authorization techniques (from T8 to T13).

$$H_{12}: \left| \begin{array}{c} Size_{CC\ and\ AOP\ (T8...T13)} \\ Separation of Concerns_{CC\ and\ AOP\ (T8...T13)} \\ Cohesion_{CC\ and\ AOP\ (T8...T13)} \\ Coupling_{CC\ and\ AOP\ (T8...T13)} \end{array} \right| \neq \left| \begin{array}{c} Size_{CC\ and\ AOP\ (T8...T13)} \\ Separation of Concerns_{CC\ and\ AOP\ (T8...T13)} \\ Cohesion_{CC\ and\ AOP\ (T8...T13)} \\ Coupling_{CC\ and\ AOP\ (T8...T13)} \end{array} \right|$$

Null Hypotheses (H_{03}): There is no statistically significant difference on size, separation of concerns, coupling and cohesion when comparing the Detect attack tactics (from T1 to T5) and Resist attack tactics (from T6 to T15).

$$H_{03}: \left| \begin{array}{c} Size_{CC\ and\ AOP\ (T1...T5)} \\ Separation of Concerns_{CC\ and\ AOP\ (T1...T5)} \\ Cohesion_{CC\ and\ AOP\ (T1...T5)} \\ Coupling_{CC\ and\ AOP\ (T1...T5)} \end{array} \right| = \left| \begin{array}{c} Size_{CC\ and\ AOP\ (T6 to T15)} \\ Separation of Concerns_{CC\ and\ AOP\ (T6 to T15)} \\ Cohesion_{CC\ and\ AOP\ (T6 to T15)} \\ Coupling_{CC\ and\ AOP\ (T6 to T15)} \end{array} \right|$$

Alternative Hypotheses (H_{13}): There is a statistically significant difference on size, separation of concerns, coupling and cohesion when comparing the Detect attack tactics (from T1 to T5) and Resist attack tactics (from T6 to T15).

$$H_{13}: \left| \begin{array}{c} Size_{CC\ and\ AOP\ (T1...T5)} \\ Separationof Concerns_{CC\ and\ AOP\ (T1...T5)} \\ Cohesion_{CC\ and\ AOP\ (T1...T5)} \\ Coupling_{CC\ and\ AOP\ (T1...T5)} \end{array} \right| \neq \left| \begin{array}{c} Size_{CC\ and\ AOP\ (T6toT15)} \\ Separationof Concerns_{CC\ and\ AOP\ (T6toT15)} \\ Cohesion_{CC\ and\ AOP\ (T6toT15)} \\ Coupling_{CC\ and\ AOP\ (T6toT15)} \end{array} \right|$$

A treatment is something that researchers manage to experimental units, while a factor is a controlled independent variable; a variable whose levels are set by the experimenter. In this context, our experiment is composed of the conditional compilation and aspect oriented programming treatments, as well as, the fifteen security techniques evaluated. The factors are the variability implementation mechanisms and security. The independent variables are those that can be controlled and changed in the empirical investigation. Changing the independent variables should produce some effect on the dependent variables (WOHLIN et al., 2012). In our study, there are two sets of independent variables, since we are interested in investigating both the security techniques (from T1 to T15) and the variability implementation mechanism (CC and AOP). The *dependent variables* are the size, separation of concerns, cohesion, and coupling evaluated for security techniques and variability implementation mechanisms. One subject per experiment was used to analyze the behavior of the dependent variables (metrics directly related to the response variable).

5.6 Operation

In this section, we described how the operation was conducted detailing the resources used and insights regarding to how the activities have been carried out. The execution of the experiment should also be presented and how data was collected during the experiments. It is essential to facilitate further replications.

5.6.1 Experiments Material

The objects of the experiments are the three SPLs developed, taken from different application domains. They were built using the proactive approach since to perform the comparison presented in this study, it is fundamental that they support the implementation of the security techniques under evaluation.

RiSE Event SPL. It was originally developed to be a benchmark for studies on functional and nonfunctional properties implementations at RiSE Labs [6] and to support the research group activities in order to manage the events organized by RiSE[7].

[6] www.rise.com.br

[7] The group has organized national and international events such as RiSS, SPLC, VAMOS and ICSR

The RiSE Event SPL (SILVEIRA et al., 2016) comprises the papers submission in conferences, journals, and related events, and its management, including the control over the review life-cycle, as well as, the management of activities (workshop, tutorial, panels), users (speakers, organizers, reviewers), registrations, payments and certificates. It was built based on the main features found on largely used conference management systems, such as: EasyChair[8], JEMS[9] and CyberChair[10]. It constitutes a core asset base integrating many features to make it suitable for several conferences. Thus, based on this common base, the products can be derived.

The SPL was developed by a PH.D. and master students using the JAVA language following the MVC architectural pattern and a remote instance of MySQL database. The SPL has 34 functional properties totalizing 26.395 lines of code, 1673 methods and 496 classes. A complete view of the collected data can be seen in the Appendix 7, Table 32 and Table 33.

RiSE Store SPL. Developed to support the major features of an interactive webstore. It is composed of features to make payment and sales as well as inventory control, product categories, FAQ and bug reports.

It was developed as part of post-graduation course by a group of eleven post-graduate students (master and PH.D. students) as part of the reuse course. They used JAVA language, adapting the MVC architectural pattern (see Figure 17) and a remote instance of MySQL database with an object-relational mapping framework for mapping an object-oriented domain model to a relational database (called hibernate[11]). It resulted in 40 functional features totalizing 5426 lines of code, 443 methods and 74 classes. A complete view of the collected data can be seen in the Appendix 7, Table 35 and Table 36.

Law Office SPL. It manages the essential activities involved in law offices routines, by managing its lawyers and their associated trainees, the process that can be raised by different costumers and companies. It also manages the office departments, finance, and several law reports.

The SPL was developed from scratch by three under-graduate students during Database and Object-Oriented Programming course. They used JAVA language and a remote instance of MySQL database without any support of frameworks. After a domain analysis, the requirements and entities were identified and modeled, starting the development process resulting in 34 functional properties totalizing 16665 lines of code, 734 methods and 116 classes. A complete view of the collected data can be seen

[8] www.easychair.org/
[9] jems.sbc.org.br/
[10] www.borbala.com/cyberchair/
[11] http://hibernate.org/

in the Appendix 7, Table 38 and Table 39.

It is important to mention that the SPLs were developed by different people containing different expertizes mitigating any possible threat regarding to SPL development style.

5.6.2 Execution

Once the SPLs were finished, it was necessary to implement the security techniques (see Table 8) using both conditional compilation and aspect-oriented programming. In order to facilitate the data extraction, ninety-three different releases were created and had data extracted from all of them. Firstly, the security techniques using conditional compilation were developed in the RiSE Event SPL resulting in fifteen versions. Next, the same procedure was undertaken to aspect-oriented mechanism to implement the same set of security techniques in each SPL. Figure 28 summarizes how the releases were built. It is important to mention that different database models were created based on different security techniques needs. In order to ease the analysis and mitigate possible threats, the security techniques had their code treated as independent portions even if it shared common lines of code with others techniques. As the development maturity on security techniques increased and/or some bugs were found in both techniques code or SPL code without the security techniques, a refactoring and/or correction were performed and the changes were propagated for all impacted versions. The same procedure was performed in the Exp2 and Exp3 (Defined in Figure 26).

Tabela 8 – Security Techniques.

ID	Security Technique	Security Tactic
T1	CheckSum	Detect Attack
T2	Hash Values	Detect Attack
T3	Verify Storage Integrity	Detect Attack
T4	MainTain Audit Trail	Detect Attack
T5	Identify Intrusion by Behavior	Detect Attack
T6	Authenticate Subject - Login/PassWord	Resist Attack
T7	Authenticate Subject - Login/Password/MachineToken	Resist Attack
T8	Authorize Subject - Authorization Pattern	Resist Attack
T9	Authorize Subject - Role Based Access Control (RBAC) Pattern	Resist Attack
T10	Authorize Subject - Multilevel security Design Pattern	Resist Attack
T11	Authorize Subject -Session Pattern	Resist Attack
T12	Authorize Subject -Reference Monitor Pattern	Resist Attack
T13	Authorize Subject - Single Access Point and CheckPoint Pattern	Resist Attack
T14	Manage Security Information	Resist Attack
T15	Hide Data by encryption	Resist Attack

5.6.3 Data collection

Accurate measurement is a prerequisite for all engineering disciplines, and software engineering is not an exception. A variety of software quality metrics has

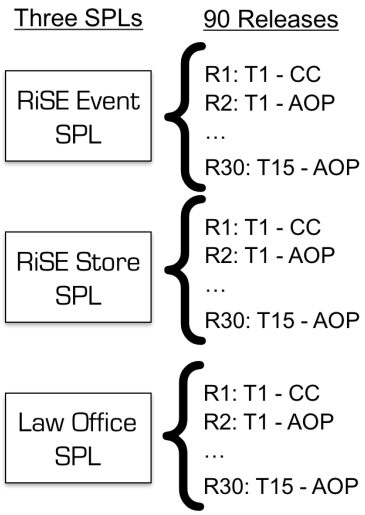

Figura 28 – Number of releases developed during experiments execution.

been developed (SARAIVA et al., 2015), and several tools exist to collect metrics from software representations (LINCKE; LUNDBERG; LöWE, 2008). As already known by the research community, there is a threat due to the inconsistency among the results when using different metrics tools (LINCKE; LUNDBERG; LöWE, 2008). In order to collect our data, we selected two metric tools: Metrics [12] and ckjm [13](SPINELLIS, 2005). This selection was based on some constraints, such as: *(i)* the tool must be compatible with JAVA language and version, *(ii)* it should extract the metrics previously described *(iii)* the tool should be compatible with any technology used in SPLs, *(iv)* it must provide its documentation and *(v)* the tools should be already known by the research community. Moreover, a total of eleven Python scripts [14] were developed to crosscheck the tools results and facilitate the data extraction.

Once the tools were selected, the data collection process was started. Firstly, several data must be collected (using Metrics tool) to further compose the metrics used in our study. These data are: *№ of classes, № of interfaces, № of methods, № of static methods, № of static attributes, № of attributes and № of parameters*. Next, ckjm tool was used to extract the LCOO metric.Appendix 7 from Table 32 to Table 39 shows these values for each SPL and security techniques implemented using conditional compilation and aspect-oriented programming. Finally, the metrics had their values calculated

[12] http://metrics.sourceforge.net/
[13] http://www.dmst.aueb.gr/dds/sw/ckjm/
[14] <https://github.com/pamsn/RiSEEventSPL/tree/master/pythonscripts>

making use of the preciously extracted data. Since the tools do not hold aspect-oriented code, all metrics collected from `.aj` files were extracted manually or using a python script <https://github.com/pamsn/RiSEEventSPL/tree/master/pythonscripts>. The Appendix 8, Table 40, Table 42 and Table 44 show the metrics results for each SPL used in our study. The Tables show all numerical data for each security technique, as well as, the difference between the SPL code without techniques (called Base) and the SPL with a given techniques. It allowed us to understand what is the impact for each security technique in the SPL under study. In addition, some metrics (e.g., CDC, CDO, CDLOC, CBC, and Scattering) were extracted using the techniques code, since they are concern related metrics. All data extracted are available at: <https://github.com/pamsn/RiSEEventSPL/>

5.7 Analysis procedure

The purpose of this Section is to present the appropriate descriptive statistics and the statistical tests used to analyze the data based on study design and goals previously established (JEDLITSCHKA; PFAHL, 2005; WOHLIN et al., 2012). When the study aims to evaluate a quantitative variable, it is important to verify whether it comes from a population that obeys a normal distribution. This verification has an impact on both how the distribution is described and which hypotheses tests should be used (WOHLIN et al., 2012).

To avoid data normalization and standardization, since it could negatively impact on results, we used the differences between the SPL with the security techniques and the default SPL (without security techniques code), which represent the security technique isolated. It was used since some metrics treat/evaluates only techniques code characteristics, such as CDC, CDO, CBC, CDLOC, and Scattering. This way, the variable regarding to SPL size was isolated from our analysis.

In this context, the data were analyzed using Shapiro-Wilk normality test to verify its normality. The test null hypothesis is that the variable to be tested is normally distributed. Thus, if the p-value is less than the chosen alpha level, then the null hypothesis is rejected and there is evidence that the data tested are not normally distributed.

The test results show that our data definitely does not follow a normal distribution. Indeed, it does not follow a symmetric distribution. Thus, we choose two non-parametric tests to analyze the results, they are following described:

Kruskal-Wallis. We used the non-parametric Kruskal-Wallis, which does not require normality assumption (CONOVER, 1999). The Kruskal-Wallis test or analysis of variance by order numbers ("ranks") can be used in cases where using the parametric ANOVA test with only slightly less potent. Furthermore, it should be used in situations

where the parametric ANOVA cannot be used in particular when the k samples do not come from normal populations, or when the variances are very heterogeneous. When k = 2, the Kruskal-Wallis test is identical to the Mann-Whitney-Wilcoxon. The hypotheses test is: H_0: The distributions of the k samples are identical; and H_1: The distributions of k samples differ in location.

Mann-Whitney-Wilcoxon Test. It is a nonparametric statistical test also known as M-W-W, applicable when data do not follow a normal distribution (CONOVER, 1999). This test is nonparametric and an alternative to t-student used to compare means of two independent samples. The underlying hypotheses test is: H_0: the two samples have identical distributions; and H_1: the two samples have different distributions.

Randomization Test. During the planning of completely randomized experiments, with two independent groups, the sample extractions from well-defined populations are rare. For example, a politician analysis, who performs an electoral survey, can enumerate the entire population and extract a random sample, but an investigator in the areas of Health or even in any other area, does not often have the possibility to randomly select samples of the population in which he is interested. (COTTON, 1967; KIRK, 1968; KEPPEL, 1973; SPENCE et al., 1976) advocates the idea that non-random samples are prevalent in experimental studies. In (SPENCE et al., 2007), the authors state that the population in which the researcher is interested is almost always a population that does not allow the extraction of a random sample.

If the samples are not random, it does not seem justifiable to use the statistical data of the experimental data, such as first choice techniques, which are based on the random sampling model.

One way to investigate whether or not a certain pattern in the data is a random effect is through the randomization test (ROBINSON, 2007; VIOLA, 2007). This test is based on the assumption that if the null hypothesis is true, all possible orders of the data are equally likely. The randomization test is a procedure in which values of an observed statistic for the data in the original arrangement are compared with the values of this statistic after the randomization of the observations.

The randomization tests have a solid theoretical basis initially formulated by (PITMAN, 1937a; PITMAN, 1937b; PITMAN, 1938; L., 1938) followed by (FISHER, 1936; FISHER, 1971) and developed by a variety of researchers, such as (KALMUS, 1952; KEMPTHORNE, 1955; EDGINGTON, 1969b; EDGINGTON, 1966; EDGINGTON, 1969a; EDGINGTON, 1969; EDGINGTON, 1995).

The advantages of randomization tests are: the data do not need to be a random sample; The data can come from any population; and extremely versatile, and can be used to calculate the significance of any test statistic (BRANCO, 2010). Stands out

the use of small samples and does not present as many constraints as conventional methods.

The main disadvantage of this test is that the conclusions obtained are restricted to each set of data and problem, and it is not possible to generalize to the population (VIOLA, 2007). Therefore, it is not necessary to calculate effect size, nor to calculate the power of the test.

The null hypothesis says that there is no pattern in the data, or it exists, it is due to the effect of change when the observations are randomized; the alternative hypotheses states that the data follow a certain pattern (ROBINSON, 2007). The decision rule is based on p-value, defined as the ratio of times the test statistic with the randomized ones is greater than or equal to the test statistic with the original data arrangement. If the p-value is less than the level of significance, we reject H_0. A statistical test is valid if the probability of rejecting the null hypothesis when it is true is less than or equal to α and is also valid when that probability is equal to α.

It is important to choose the test statistic appropriately, in this study the means of two small independent samples were compared and the requirements for the use of parametric tests were not met, the Wilcoxon-Mann-Whitney non-parametric test was used. The application of the randomization test considered the Wilcoxon-Mann-Whitney (or Kruskal-Wallis) test statistic and 10,000 randomizations. (WESTFALL; YOUNG, 1993) state that in order to analyze a real set of data, the number of readings should be as large as possible (10000 or more). Thus, the data analyzed by different researchers will show little variation in the results. This process was performed 30 times, storing the p-value. So the average of these 30 results will be our p-value. This methodology was adopted to ensure that the result was not always favorable or unfavorable depending on the seed chosen. In order to perform our analysis, some R project and shell scripts [15] were created and the overall execution spent 23 hours.

Test Power. The power of a statistical test is the probability of rejecting a false null hypothesis. When a statistical test is performed, correct decisions are made when a false null hypothesis is rejected or a true null hypothesis is not rejected. But incorrect decisions can be made when rejecting a true null hypothesis (Type I error) or rejecting a false null hypothesis (Type II error). The probability of making a Type I error is designated by α and the probability of making a Type II error is designated by β. Thus, the potency of a test is 1-β.

The three determinants of the power of a test are: the level of significance; The sample size n; and the size of the effect δ. The power will be higher the higher the values of α, of n and of δ.

[15] <https://github.com/pamsn/RiSEEventSPL>

The power of the test is used to plot the sample size and is also useful for interpreting inferential statistical analysis results in which the difference found was not statistically significant. Generally, the power of the test should be greater than or equal to 80%, that is, an 80% or more probability of finding a statistically significant difference when it actually exists, although there are no formal standards of test power (ELLIS, 2010). Statistical power is not used as a reference for control of the test, since the control must be done in relation to the value of α, since committing a type I error (rejecting the null hypothesis when it is true) in an analysis is very more serious than to make a Type II error (do not reject a hypothesis when it is false).

Effect Size. The Cliff delta (δ) is the non-parametric measure that allows quantifying the magnitude of the difference between two observation groups that do not meet the normality assumptions, and its description allows to complement the interpretation of the p value associated to the corresponding test of Hypotheses used (MACBETH; RAZUMIEJCZYK; LEDESMA, 2011). The recommendation regarding the use of the Cliff delta is linked to cases where the distributions are asymmetric and violate the homogeneity of the variances or even in cases where outliers are present (SUN; PAN; WANG, 2010). The Cliff delta does not depend on the mean, but on a concept of dominance, it considers the ordinal properties rather than the interval properties of the data (CLIFF, 1993).

The delta ranges from -1 to +1, and can assume any value for this range. A delta of -1 or +1 indicates an absence of overlap, while a delta of 0.0 indicates that the groups are completely overlapping (KROMREY; HOGARTY, 1998). In addition, (+1) indicates that group 1 is greater than group 2, and (-1) means that group 1 is smaller than group 2, whereas (0) is translated as group 1 equal to group 2. The statistic generates an array of i rows and j columns, assuming three possible values +1, -1 and 0 (MACBETH; RAZUMIEJCZYK; LEDESMA, 2011).

As an overall measure of effect size following a significant Kruskal-Wallis test result, we use a statistic known as epsilon-squared ($e\hat{}2$) as an appropriate measure (KING; MINIUM, 2003).

P-value interpretation. The meaning and interpretation of P-values which directly depends on a given sample attempts to provide a measure of the strength of the results, in contrast to a simple yes or no decision (H., 1988). In this context, our study was analyzed based on the following interpretation of p-values (see Table 9):

Finally, the ***boxplot graph*** was used to evaluate the distribution of empirical data. It is composed of the first quartile (Q1), median (Md) and third quartile (Q3). The lower and upper lines extend, respectively, from the bottom quartile to the lowest value, not less than the lower limit and the upper quartile to the highest value not greater than the upper limit. The limits are calculated as follows: lower limit: HQ_1 - 1,5 (Q_3 - Q_1) and

Tabela 9 – Measure of the p-value strength of the results.

P-value	Interpretation
P < 0.01	very strong evidence against H0
0.01 ⩽ P < 0.05	moderate evidence against H0
0.05 ⩽ P < 0.10	suggestive evidence against H0
0.10 ⩽ P	little or no real evidence against H0

upper limit: Q_3 + 1,5(Q_3 - Q_1). Appendix 8 (from Figure 34 to Figure 55) shows the variation among different security techniques using CC and AOP implementations.

In addition, our statistical analysis considered the numeric as well as the variance among the results of the three experiments. It is important since the test beds have different size regarding to lines of code and number of entities.

5.8 Analysis and Interpretation

This section reports the data analysis of the family of experiments in order to understand the data collected to derive conclusions about the relationships between factors, the aspects that improve the values of the response variables, and the influence of factors on the response variables (JURISTO; MORENO, 2010). Based on previously defined quality model (Figure 27), the following rules (KULESZA et al., 2006b; FIGUEIREDO et al., 2008) were applied during the analysis procedure: *(i)* the smaller the value of the metrics of size (LOC, NOA, WOC, VS), the less complex and easier to understand will be the technique; *(ii)* the lower the value of Separation of Concerns (SoC) metrics (CDC, CDO, CDLOC, Scattering), less widespread will be the code of the technique making it more modular, easy to understand and with better flexibility; *(iii)* the lower the value of Lack of Cohesion over operations (LCOO), the greater the cohesion, thus reducing the complexity, making the software more modular and increasing its flexibility and understandability; and *(iv)* the lower the coupling metrics (CBC, DIT) value, the less coupled will be the components making the code to be more flexible and modular, ease to understand and less complex. It is important to mention that the analysis makes use of three samples which corresponds the metric value for the security technique implemented in each product line (testbed).

RQ - What is the impact on maintainability when evaluating security techniques implemented using conditional compilation and aspect-oriented programming?

The security techniques implementation were compared to identify how they are distributed regarding to maintainability cost. In addition, we are also interested in understanding how the variability implementation mechanism affects those factors when

implementing security techniques in SPL. This way, we present the engineering point of view in which the AOP and CC implementations are compared and researcher points of view in which different security techniques are classified and evaluated.

In the next sections, an analysis is performed considering the different maintainability internal attributes (Size, Separation of Concerns, Coupling, and Cohesion) and how they behavior for both points of view.

5.8.1 Size

As previously stated in Section 5.5.3, the internal attribute size was assessed through the following metrics: Lines of Code (LOC), Number of Attributes (NOA), Weighted Operations per Component (WOC) and Vocabulary Size (VS). Table 10 shows the p-value results regarding to the statistical analysis performed to compare both AOP and CC by applying the statistical methods previously mentioned and confirmed by randomization test p-value. Values less than or equal to 0.05 means that there is a significant statistical difference between CC and AOP. Table 11 shows the power effect for each metric. For more details regarding to cliff's (δ), magnitude and Cohen's d, see Appendix 8 Tables 46 and 47.

Tabela 10 – Size comparison between CC vs AOP implementations.

Techniques	LOC			NOA			WOC			VS		
Techniques	w	p-value	Rand. p-value	w	p-value	Rand. p-value	w	p-value	Rand. p-value	w	p-value	Rand. p-value
T01	9	0.0495	0.0499	6	0.3173	0.4993	7	0.2752	0.2002	9	0.0253	0.0496
T02	9	0.0495	0.0499	6	0.3173	0.4997	9	0.0495	0.0497	9	0.0253	0.0488
T03	9	0.0495	0.0493	6	0.3173	0.5004	9	0.0463	0.0499	9	0.0253	0.0496
T04	9	0.0495	0.05	5.5	0.6579	0.3994	7	0.2752	0.1997	9	0.0431	0.0496
T05	9	0.0495	0.0494	5.5	0.6579	0.4001	7	0.2752	0.2	9	0.0431	0.0498
T06	6	0.5127	0.3491	5.5	0.6579	0.4005	7	0.2752	0.1994	9	0.0431	0.0496
T07	6	0.5127	0.3496	6	0.5002	0.399	7	0.2683	0.1498	9	0.0431	0.05
T08	5	0.8273	0.5003	6	0.4867	0.3492	7	0.2752	0.1989	9	0.0431	0.0499
T09	5	0.8273	0.5003	5	0.8248	0.4997	5	0.8273	0.5004	9	0.0431	0.0497
T10	5	0.8273	0.4999	5.5	0.6579	0.3994	7	0.2752	0.1994	8	0.099	0.15
T11	7	0.2683	0.2009	6	0.4867	0.3508	7	0.2752	0.2001	9	0.0431	0.0495
T12	5	0.8273	0.5009	6	0.4867	0.3501	7	0.2752	0.2006	9	0.0431	0.05
T13	7	0.2752	0.1997	5	0.8248	0.5002	7	0.2752	0.1991	9	0.0431	0.0492
T14	6	0.5127	0.3482	6	0.4867	0.3498	7	0.2752	0.2013	9	0.0431	0.0496
T15	6	0.5127	0.3495	4.5	NaN	1	9	0.0463	0.0498	9	0.0253	0.0499

Tabela 11 – Power effect for Size Metrics.

Techniques	LOC			NOA			WOC			VS		
Techniques	(δ) of Cliff	Magnitude	Power Test	(δ) of Cliff	Magnitude	Power Test	(δ) of Cliff	Magnitude	Power Test	(δ) of Cliff	Magnitude	Power Test
T01	1	Large	0.8285	0.3333	medium	0.1165	0.5556	large	0.2757	1	large	0.05
T02	1	large	0.8652	0.3333	medium	0.1165	1	large	0.4768	1	large	0.05
T03	1	large	0.9888	0.3333	medium	0.1165	1	large	0.7067	1	large	0.05
T04	-1	large	0.9552	0.2222	small	0.086	0.5556	large	0.0609	1	large	0.9915
T05	-1	large	0.5701	0.2222	small	0.086	0.5556	large	0.069	1	large	0.9915
T06	0.3333	medium	0.0959	0.2222	small	0.086	0.5556	large	0.0647	1	large	0.8495
T07	0.3333	medium	0.089	-0.3333	medium	0.0764	0.5556	large	0.0645	1	large	0.8495
T08	-0.1111	negligible	0.0501	0.3333	medium	0.0818	0.5556	large	0.056	1	large	0.8495
T09	-0.1111	negligible	0.0501	0.1111	negligible	0.0673	-0.1111	negligible	0.0711	1	large	0.8495
T10	-0.1111	negligible	0.05	0.2222	small	0.0717	0.5556	large	0.0515	0.7778	large	0.3441
T11	0.5556	large	0.0746	0.3333	medium	0.0818	0.5556	large	0.0579	1	large	0.8495
T12	-0.1111	negligible	0.0501	0.3333	medium	0.0818	0.5556	large	0.0573	1	large	0.8495
T13	-0.5556	large	0.1532	0.1111	negligible	0.0673	0.5556	large	0.057	1	large	0.8495
T14	0.3333	medium	0.1267	0.3333	medium	0.0818	0.5556	large	0.0522	1	large	0.8495
T15	-0.3333	medium	0.067	0	negligible	0.05	1	large	0.4684	1	large	0.05

Security Technique	Size	
Verify Message Integrity (T1 and T2)	LOC	There is a suggestive evidence that CC implementation is better for T1 and T2.
	NOA	There is a suggestive evidence that AOP and CC results have no statisticall significant difference for T1 and T2.
	WOC	There is a suggestive evidence that CC implementation is better for T2.
	VS	There is a suggestive evidence that CC implementation is better for T1 and T2.
Verify Storage Integrity (T3)	LOC	There is a suggestive evidence that CC implementation is better for T3.
	NOA	There is a suggestive evidence that AOP and CC results have no statisticall significant difference for T3.
	WOC	There is a suggestive evidence that CC implementation is better for T3.
	VS	There is a suggestive evidence that CC implementation is better for T3.
Maintain Audit Trail (T4)	LOC	There is a suggestive evidence that AOP implementation is better for T4.
	NOA	There is a suggestive evidence that AOP and CC results have no statisticall significant difference for T4.
	WOC	There is a suggestive evidence that AOP and CC results have no statisticall significant difference for T4.
	VS	There is a suggestive evidence that CC implementation is better for T4.
Identify intrusion by behavior (T5)	LOC	There is a suggestive evidence that AOP implementation is better for T5.
	NOA	There is a suggestive evidence that AOP and CC results have no statisticall significant difference for T5.
	WOC	There is a suggestive evidence that AOP and CC results have no statisticall significant difference for T5.
	VS	There is a suggestive evidence that CC implementation is better for T5.
Authenticate subject (T6 e T7)	LOC	There is a suggestive evidence that AOP and CC results have no statisticall significant difference for T6 and T7.

Continued on next page

Tabela 12 – *Continued from previous page*

Security Technique	**Size**	
	NOA	There is a suggestive evidence that AOP and CC results have no statisticall significant difference for T6 and T7.
	WOC	There is a suggestive evidence that AOP and CC results have no statisticall significant difference for T6 and T7.
	VS	There is a suggestive evidence that CC is better for T6 and T7.
Authorize subject (from T8 to T13)	LOC	There is a suggestive evidence that AOP and CC results have no statisticall significant difference from T8 and T13.
	NOA	There is a suggestive evidence that AOP and CC results have no statisticall significant difference from T8 and T13.
	WOC	There is a suggestive evidence that AOP and CC results have no statisticall significant difference from T8 and T13.
	VS	There is a suggestive evidence that CC is better for T8. There is a suggestive evidence that CC is better for T9. There is a suggestive evidence that AOP and CC results have no statisticall significant difference for T10. There is a suggestive evidence that CC is better for T11. There is a suggestive evidence that CC is better for T12. There is a suggestive evidence that CC is better for T13.
Manage security information (T14)	LOC	There is a suggestive evidence that AOP and CC results have no statisticall significant difference for T14.
	NOA	There is a suggestive evidence that AOP and CC results have no statisticall significant difference for T14.
	WOC	There is a suggestive evidence that AOP and CC results have no statisticall significant difference for T14.
	VS	There is a suggestive evidence that CC is better for T14.

Continued on next page

Tabela 12 – *Continued from previous page*

Security Technique	**Size**	
Hide Data by encryption (T15)	LOC	There is a suggestive evidence that AOP and CC results have no statisticall significant difference for T15.
	NOA	There is a suggestive evidence that AOP and CC results have no statisticall significant difference for T15.
	WOC	There is a suggestive evidence that CC is better for T15.
	VS	There is a suggestive evidence that CC is better for T15.

Tabela 12 – Summary table size.

AOP vs CC. Based on the data gathered, the investigation of security techniques implementation can be classified in three groups: *(i)* represents the security techniques that the CC solution provide better results; *(ii)* represents the security techniques solutions in which AOP have shown as superior and *(iii)* involves the security techniques in which the variability implementation mechanism does not impact the results. All three groups are shown in Table 12.

The **null hypotheses** H_{01} was rejected in the following manner: CC techniques implementation presented better results when compared with AOP implementations, when considering most of the metrics (LOC, WOC and VS) responsible to evaluate size internal attribute. The group includes Verify Storage Integrity (T3), Verify Message Integrity (T2 and T1), Hide Data by encryption (T15), Authenticate subject (T6 e T7), Identify intrusion by behavior (T5), Maintain Audit Trail (T4) and Authorize subject (T8, T11, T12, T13 and T14) (Starting from the security technique that presents the best results for size attribute). In the next paragraphs, we provide an in depth analysis.

Considering AOP implementations, there are some techniques that exhibited a small value for all size metrics. This group includes the techniques Maintain Audit Trail (T4) and Identify intrusion by behavior (T5), which presented better results for LOC metric. It may be due to its code that is spread and repeated over the class when using CC solution. While T4 keeps a record of users and system actions which are basically a log implementation, the T5 makes use of this log and the user behavior to detect a deviation from the user utilization pattern. This way, all code responsible for user execution and behavior analysis, as well as, the crosscutting code can be isolated in the aspects. It provide insights to reject the **null hypotheses** H_{01}.

From the results provided from LOC metric, it indicates that if the security technique requires that a given data should be manipulated in a specific point of a method, a hook method needs to be created increasing the lines of code in the AOP solution. It can be viewed in the implementation of techniques T1 and T2 (see Listing 5.3). NOA metric did not present any difference for AOP and CC solutions, it happens since the implementation of the techniques is the same in both solutions. The difference is only the code related to the crosscutting concern which was isolated within the aspect. It did not happen for VS metric, as the aspect solution isolated the crosscutting concern code inside the aspect, in addition to the classes used in the CC solution, new aspects were created to isolate this code.

Código 5.3 – Hook method example.

```
1  ...
2      //#if ${ variability2 .8} == "T"
3      private void technique1_insert(Object object){
4        try {
5          hashValues.insert(object);
6        } catch (CheckSumGenerationException e) {
7          // TODO Auto-generated catch block
8          e.printStackTrace();
9        } catch (RepositoryException e) {
10         // TODO Auto-generated catch block
11         e.printStackTrace();
12       }
13     }
14     //#endif
15
16     public void insertEvent(Event event) throws EventAlreadyInsertedException,
           RepositoryException{
17       this.events.insert(event);
18       //#if ${ variability2 .8} == "T"
19         technique1_insert(event);
20       //#endif
21   }
22 ...
```

An insider view regarding to the security techniques implementation in each experiment show that SPL code size influences on the results. Table 13 shows the percentage difference between the techniques implemented when using CC and AOP. Negative values indicate that CC is less costly then AOP. On the other hand, positive values indicate that AOP is less costly than CC. From the metric with the greatest difference, we can extract insights that the lower the SPL LOC, the less costly is the maintenance when using CC. See Appendix 8 Table 41, Table 43 and Table 45 for more

details. Due to the code techniques are well spread throughout different architecture layers, the more line of code the SPL has the greater will be the code spread. In this context, the AOP solution is more appropriated.

Tabela 13 – Percentual difference among Size metrics.

	RiSE Event				RiSE Store				Office Law			
CCvsAOP	LOC	NOA	WOC	VS	LOC	NOA	WOC	VS	LOC	NOA	WOC	VS
T01	-0.94	0	-1.04	-0.38	-4.7	-0.34	-4.69	-2.35	-2.49	0	-4.42	-1.56
T02	-0.94	0	-1.05	-0.38	-4.09	-0.33	-4.92	-2.35	-2.34	0	-4.55	-1.56
T03	-0.06	0	-0.1	-0.19	-0.27	-0.34	-1.16	-1.28	-0.1	0	-0.89	-0.83
T04	0.63	-0.41	-1.17	-0.55	5.19	-0.3	-0.68	-2.94	2.68	0	-0.69	-2.07
T05	0.14	-0.41	-1.27	-0.55	4.12	-0.31	-0.91	-2.94	2.27	0	-0.58	-2.07
T06	-0.29	-0.41	-1.05	-0.37	-0.33	-0.31	-0.52	-2.06	-0.14	0	-0.46	-1.43
T07	-0.29	0.21	-1.15	-0.37	-0.3	0	-0.51	-2.04	-0.14	0	-0.39	-1.42
T08	0.1	-0.4	-0.89	-0.71	0.24	-0.29	-2.49	-3.39	-0.18	0	-0.91	-2.48
T09	0.1	-0.39	-1.12	-0.69	0.2	-0.28	-2.1	-2.92	-0.18	0.12	17.77	-2.22
T10	0.11	-0.38	-1.18	-0.65	0.15	-0.25	-1.08	-2.29	-0.2	0	-1.2	-1.83
T11	-0.16	-0.4	-1.17	-0.71	-0.53	-0.29	-2.48	-3.36	-0.47	0	-0.91	-2.47
T12	0.06	-0.4	-1.17	-0.71	0.29	-0.29	-2.47	-3.31	-0.15	0	-0.9	-2.44
T13	0.85	-0.4	-1.28	-0.71	1.42	-0.28	-2.64	-3.23	0.56	0.12	-1.08	-2.4
T14	0.1	-0.39	-1.32	-0.69	0.19	-0.26	-1.75	-2.88	-0.17	0	-0.74	-2.2
T15	-0.25	0	-7.35	-0.19	1.12	0	-3.36	-1.28	1.49	0	-13.65	-0.83

Authorization. Comparing all authorization patterns considering CC and AOP implementations, only WOC did not present statistical significant difference at 0.05 level for both AOP and CC (see Table 14). The Table 30 shows the power effect, effect size and magnitude of each metric. This way, LOC, NOA and VS values rejected the **null hypotheses** H_{02}. Regarding to LOC, NOA and VS the authorization patterns can be ordered starting from the best, such as: T8, T11, T12, T13, T14 and T10. While T8 establishes a relationship between subject and object to evaluate if a subject is authorized (i.e. have rights) to access certain objects, T10 categorizes subjects and objects in different levels and then classify it. It is also important to mention that technique T11 besides the code to control the functionalities access, it also has code to determine how long time the session will be opened (see Listing 5.4). This way, this technique can have an increase in size internal attribute if all screens and/or functionalities were controlled by this timer. A deeper view can show that the main difference among these patterns was the code responsible to implement their rules since the code spread over the SPLs are almost the same except for T11. T10 appears to be the most costly since it needs several levels of security classifications to be applied for both subjects and objects. Appendix 8 (from Figure 34 to Figure 41) also shows that the same order can be seen for CC and AOP implementations.

Código 5.4 – Session timeout functionality.

```
1  ...
2      //#if ${technique11} == "T"
3      private boolean technique11(String functionality){
```

```
4         boolean resposta = false;
5         try  {
6            int  idUser = RiSEEventLoginScreen.getLoggedUser().getIdUser();
7            if (RiSEEventLoginScreen.getFacade().checkPermission(idUser, functionality) &&
                 (Session.checkSessionTime(RiSEEventLoginScreen.getSessionStartHour()) == true)){
8              resposta = true;
9            }else
10             resposta = false;
11        } catch (PermissionDeinedException e) {
12           JOptionPane.showMessageDialog(null, "You do not have permission", "Error",
                 JOptionPane.ERROR_MESSAGE);
13           e.printStackTrace();
14        } catch (RepositoryException e) {
15           //  TODO Auto-generated catch block
16           e.printStackTrace();
17        }
18        return  resposta;
19     }
20     //#endif
21  ...
22     @Override
23     public  void  actionPerformed(ActionEvent e) {
24        //#if  ${technique11.4} == "T"
25        String   functionality   = new Object(){}.getClass().getEnclosingClass().getSimpleName();
26        if (technique11(functionality)   == true){
27        //#endif
28              dispose();
29           //#if  ${technique11.4} == "T"
30           }
31        //#endif
32
33     }
34  ...
```

Detect Attack. In the context of detect attack techniques, the results showed that Verify Storage Integrity (T3) is the less costly regarding to size attribute being followed by Verify Message Integrity (T1 and T2), Maintain Audit Trail (T4) and Identify intrusion by behavior (T5). It can be explained by the fact that T3 has a really specific implementation to verify if the database had their data corrupted (see Listing 5.5). On the other hand, T5 besides to implement user action by tracking user action and registering their log, it also evaluates the user behavior. Table 14 shows a significant statistical difference when comparing detect attack techniques for both CC and AOP implementations.

Código 5.5 – Database dump verification.

```
1  ...
```

Tabela 14 – Comparison of Size metrics from CC and AOP implementations.

	LOC			NOA			WOC			VS		
CC	k	p-value	Rand p-value	k	p-value	Rand p-value	k	p-value	Rand p-value	k	p-value	Rand p-value
Authorization	15.0585	0.0101	0	15.3445	0.009	0	9.2844	0.0982	0.0674	16.1677	0.0064	0
Detect Attack	12.6333	0.0132	0.0001	13.5727	0.0088	0	13.0567	0.011	0	13.3412	0.0097	0
Resist Attack	27.5979	0.0011	0	28.3262	0.0008	0	24.7422	0.0033	0	28.5375	0.0008	0
CC	w	p-value	Rand p-value	w	p-value	Rand p-value	w	p-value	Rand p-value	w	p-value	Rand p-value
Detect vs Resist	54	0	1	47.5	0	1	48	0	1	76.5	0.0003	0.9999
AOP	k	p-value	Rand p-value	k	p-value	Rand p-value	k	p-value	Rand p-value	k	p-value	Rand p-value
Authorization	15.0038	0.0103	0	15.0582	0.0101	0.0001	7.8538	0.1645	0.1501	16.1677	0.0064	0
Detect Attack	12.4333	0.0144	0.0001	13.3187	0.0098	0	12.5667	0.0136	0.0001	13.3412	0.0097	0
Resist Attack	27.6298	0.0011	0	28.056	0.0009	0	22.6653	0.007	0.0001	28.5375	0.0008	0
AOP	w	p-value	Rand p-value	w	p-value	Rand p-value	w	p-value	Rand p-value	w	p-value	Rand p-value
Detect vs Resist	60	0.0001	1	48	0	1	34	0	1	76.5	0.0003	0.9999

Tabela 15 – Power Effect, Effect size and magnitude of Size metrics.

	LOC			NOA			WOC			VS		
CC	Power Effect	Effect Size	Mag.	Power Effect	Effect Size	Mag.	Power Effect	Effect Size	Mag.	Power Effect	Effect Size	Mag.
Authorization	1	0.3422	high	1	0.3487	high	1	0.211	"medium"	1	0.3674	high
Detect	1	0.2871	high	1	0.3085	high	1	0.2967	high	1	0.3032	
Resist	1	0.6272	very high	1	0.6438	very high	1	0.5623	very high	1	0.6486	"very high"
AOP	Power Effect	Effect Size	Mag.	Power Effect	Effect Size	Mag.	Power Effect	Effect Size	Mag.	Power Effect	Effect Size	Mag.
Authorization	1	0.341	high	1	0.3422	high	1	0.1785	medium	1	0.3674	high
Detect	1	0.2826	high	1	0.3027	high	1	0.2856	high	1	0.3032	"high"
Resist	1	0.628	very high	1	0.6376	very high	1	0.5151	very high	1	0.6486	very high

```
2     public RiSEEventMainScreenP() {
3     //#if ${technique3} == "T"
4     Backup.databaseBackup("Release2.9", "root", "password",
          "Workspace/003_RiSEEventSPL_CC_T3/backup_OLD.sql");
5     String file1 = "Workspace/003_RiSEEventSPL_CC_T3/backup_NEW.sql";
6     String file2 = "Workspace/003_RiSEEventSPL_CC_T3/backup_OLD.sql";
7     try {
8       if (Compare.compareFile(file1, file2) == 0){
9         JOptionPane.showMessageDialog(getContentPane(),
10            "The system was hacked. Please, Contact Technical Support.", "Erro",
11            JOptionPane.INFORMATION_MESSAGE);
12      }
13    } catch (Exception e) {
14      // TODO Auto-generated catch block
15      e.printStackTrace();
16    }
17    //#endif
18 ...
```

Resist Attack. In addition to the Authorization techniques, the resist attack tactic is composed of Manage security information (T14) and Hide Data by encryption (T15).

In this context, the most efficient technique was T15 in which hides all system data using encryption. As expected, T14 presents values compatible with authorization techniques since it is composed of authorization control and also code responsible for encrypting security information. Table 14 shows a significant statistical difference when comparing resist attack techniques for both CC and AOP implementations.

Detect Attack vs Resist Attack. According to the results shown in Table 14 the **null hypotheses** H_{03} was not rejected since there is no significant statistical difference among these tactics. Although the p-values indicate that there is no significant difference, if we take a look at Appendix 8 (from Figure 34 to Figure 41) due to the values for techniques T6, T7 and T15, we can see that authorization technique has a higher cost for maintenance.

Metrics Analysis. Based on the results from Table 10 we saw that NOA metric may not be a good indicator to evaluate size attribute when comparing CC and AOP implementations for security techniques since it did not show any significant statistical difference. It can be explained by the use of intertype declarations used in the AOP implementations. On the other hand, LOC metric seems to be a good indicator to compare different implementations for detect attack techniques, for all of the techniques, the analysis point out that there is a significant statistical difference. In addition, VS metric shows a significant statistical difference for most of the security techniques except T10 which presents the largest number of classes to be implemented. A number of classes to implement impacted the comparison between CC and AOP.

Considering the three experiments and their test beds, the metric which presented the greater variation was VS, especially when considering authorization techniques. It happens since the authorization techniques have much of their code spread over different system screens or functionalities.

5.8.2 Separation of Concerns

The Separation of Concern internal attribute was evaluated through the following metrics: Concern Diffusion over Components(CDC), Concern Diffusion over Operations(CDO), Concern Diffusion over LOC (CDLOC) and Code Scattering. It is important to state that the lower the value of the metric, less costly will be the technique maintenance. Firstly, the analysis is divided into two parts. It focuses on the analysis of to what extent the aspect-oriented programming (AOP) and conditional compilation (CC) solutions provide support for the separation of security techniques concerns. Table 16 shows the results regarding to the statistical analysis performed to compare both AOP and CC with respect to separation of concerns metrics by applying the statistical methods previously mentioned and confirmed by randomization test p-value. The values below 0.05 means that there is a statistically significant difference between both solutions. Table 17 shows

the power effect for each metric. For more details regarding to cliff's (δ), magnitude and Cohen's d, see Appendix 8 Tables 48 and 49.

Tabela 16 – Separation of concerns comparison between CC vs AOP implementations.

Techniques	CDC			CDO			CDLOC			Scattering		
Techniques	w	p-value	Rand. p-value	w	p-value	Rand. p-value	w	p-value	Rand. p-value	w	p-value	Rand. p-value
T01	4.5	NaN	1	9	0.0495	0.05	9	0.0463	0.0495	9	0.0495	0.0494
T02	4.5	NaN	1	9	0.0495	0.0499	9	0.0463	0.049	9	0.0463	0.0496
T03	4.5	NaN	1	8	0.1046	0.1491	9	0.0253	0.0499	9	0.0495	0.05
T04	6	0.5002	0.3009	9	0.0495	0.0499	9	0.0463	0.05	9	0.0495	0.05
T05	6	0.5002	0.2994	8	0.1266	0.1001	6	0.5127	0.3505	9	0.0495	0.0498
T06	5	0.7963	0.5014	6	0.5127	0.3499	7	0.2612	0.151	9	0.0495	0.05
T07	5	0.7963	0.501	6	0.5127	0.3499	7	0.2612	0.1506	9	0.0495	0.0488
T08	6	0.5002	0.3004	5	0.8273	0.4998	5	0.8273	0.4992	9	0.0495	0.0497
T09	6	0.5002	0.3002	8	0.1266	0.1	5	0.8273	0.5	9	0.0495	0.0493
T10	7	0.2612	0.1503	7	0.2752	0.1994	5	0.8273	0.4989	9	0.0495	0.0498
T11	6	0.5002	0.3006	5	0.8273	0.5005	5	0.8273	0.5006	9	0.0495	0.0493
T12	8	0.099	0.1504	5	0.8273	0.5005	5	0.8273	0.5004	9	0.0495	0.0499
T13	7	0.2612	0.1507	9	0.0495	0.0496	9	0.0495	0.0496	9	0.0495	0.0493
T14	9	0.0463	0.05	6	0.5127	0.3504	5	0.8273	0.4997	9	0.0495	0.05
T15	7.5	0.1213	0.1999	7	0.2752	0.1992	5	0.8273	0.4989	9	0.0495	0.0499

Tabela 17 – Power effect for Separation of Concerns Metrics.

Techniques	CDC			CDO			CDLOC			Scattering		
Techniques	(δ) of Cliff	Magnitude	Power Test	(δ) of Cliff	Magnitude	Power Test	(δ) of Cliff	Magnitude	Power Test	(δ) of Cliff	Magnitude	Power Test
T01	0	negligible	NaN	-1	large	0.5359	-1	large	1	-1	large	0.9998
T02	0	negligible	NaN	-1	large	0.6633	-1	large	1	-1	large	0.9999
T03	0	negligible	NaN	0.7778	large	0.3133	-1	large	0.05	-1	large	1
T04	0.3333	medium	0.0571	-1	large	0.9955	-1	large	0.9492	-1	large	1
T05	0.3333	medium	0.0571	0.7778	large	0.2787	0.3333	medium	0.0667	-1	large	0.9899
T06	-0.1111	negligible	0.0948	-0.3333	medium	0.05	-0.5556	large	0.1795	-1	large	0.4497
T07	-0.1111	negligible	0.0948	-0.3333	medium	0.0501	-0.5556	large	0.1795	-1	large	0.4151
T08	0.3333	medium	0.055	-0.1111	negligible	0.0542	0.1111	negligible	0.0535	-1	large	0.6091
T09	0.3333	medium	0.055	-0.7778	large	0.1549	0.1111	negligible	0.0535	-1	large	0.7633
T10	-0.5556	large	0.0937	-0.5556	large	0.0546	0.1111	negligible	0.0539	-1	large	0.8894
T11	0.3333	medium	0.055	0.1111	negligible	0.051	0.1111	negligible	0.0535	-1	large	0.6097
T12	-0.7778	large	0.2023	-0.1111	negligible	0.0625	0.1111	negligible	0.0535	-1	large	0.6963
T13	-0.5556	large	0.1304	-1	large	0.986	-1	large	0.4146	-1	large	0.8716
T14	-1	large	0.3615	-0.3333	medium	0.0523	0.1111	negligible	0.0566	-1	large	0.7653
T15	-0.6667	large	0.2592	0.5556	large	0.1502	0.1111	negligible	0.0532	-1	large	0.5828

AOP vs CC. Differently, from Size internal attribute, the data gathered revealed only two groups: *(i)* represents the security techniques solutions in which AOP has shown as superior and *(ii)* involves the security techniques in which the variability implementation mechanism did not impact on the results. All of them can be visualized in Table 18.

Security Technique	**Separation of Concerns**	
Verify Message Integrity (T1 and T2)	CDC	There is a suggestive evidence that AOP and CC results have no statistical significant difference for T1 and T2.
	CDO	There is a suggestive evidence that AOP implementation is better for T1 and T2
	CDLOC	There is a suggestive evidence that AOP implementation is better for T1 and T2
	Scattering	There is a suggestive evidence that AOP implementation is better for T1 and T2
Verify Storage Integrity (T3)	CDC	There is a suggestive evidence that AOP and CC results have no statistical significant difference for T3.
	CDO	There is a suggestive evidence that AOP and CC results have no statistical significant difference for T3.
	CDLOC	There is a suggestive evidence that AOP implementation is better for T3.
	Scattering	There is a suggestive evidence that AOP implementation is better for T3.
Maintain Audit Trail (T4)	CDC	There is a suggestive evidence that AOP and CC results have no statistical significant difference for T4.
	CDO	There is a suggestive evidence that AOP implementation is better for T4.
	CDLOC	There is a suggestive evidence that AOP implementation is better for T4.
	Scattering	There is a suggestive evidence that AOP implementation is better for T4.
Identify intrusion by behavior (T5)	CDC	There is a suggestive evidence that AOP and CC results have no statistical significant difference for T5.

Continued on next page

Tabela 18 – *Continued from previous page*

Security Technique	**Separation of Concerns**	
	CDO	There is a suggestive evidence that AOP and CC results have no statistical significant difference for T5.
	CDLOC	There is a suggestive evidence that AOP and CC results have no statistical significant difference for T5.
	Scattering	There is a suggestive evidence that AOP implementation is better for T5.
Authenticate subject (T6 e T7)	CDC	There is a suggestive evidence that AOP and CC results have no statistical significant difference for T6 and T7.
	CDO	There is a suggestive evidence that AOP and CC results have no statistical significant difference for T6 and T7.
	CDLOC	There is a suggestive evidence that AOP and CC results have no statistical significant difference for T6 and T7.
	Scattering	There is a suggestive evidence that AOP implementation is better for T6 and T7.
Authorize subject (from T8 to T13)	CDC	There is a suggestive evidence that AOP and CC results have no statistical significant difference from T8 to T13.
	CDO	There is a suggestive evidence that AOP and CC results have no statistical significant difference from T8 to T12. There is a suggestive evidence that AOP implementation is better for T13.
	CDLOC	There is a suggestive evidence that AOP and CC results have no statistical significant difference from T8 to T12. There is a suggestive evidence that AOP implementation is better for T13.
	Scattering	There is a suggestive evidence that AOP implementation is better from T8 to T13.

Continued on next page

Tabela 18 – *Continued from previous page*

Security Technique	**Separation of Concerns**	
Manage security information (T14)	CDC	There is a suggestive evidence that AOP implementation is better for T14.
	CDO	There is a suggestive evidence that AOP and CC results have no statistical significant difference from T14.
	CDLOC	There is a suggestive evidence that AOP and CC results have no statistical significant difference from T14.
	Scattering	There is a suggestive evidence that AOP implementation is better for T14.
Hide Data by encryption (T15)	CDC	There is a suggestive evidence that AOP and CC results have no statistical significant difference from T15.
	CDO	There is a suggestive evidence that AOP and CC results have no statistical significant difference from T15.
	CDLOC	There is a suggestive evidence that AOP and CC results have no statistical significant difference from T15.
	Scattering	There is a suggestive evidence that AOP implementation is better for T15.

Tabela 18 – Summary table separation of concerns.

The **null hypotheses** H_{01} was rejected by Concern diffusion over Components (CDC) metric, i.e., to what extent the security technique is isolated through the system components in both solutions. It happened for Manage security information (T14) technique since besides the components to implement the data authorization and data encryption, it also had its code spread over the screens to control authorization. Next, the Concern Diffusion over Operations (CDO) metric presents a significant difference in techniques Maintain Audit Trail (T4) and Authorize subject (T13). While the former had their code spread over all SPL functionalities inside the facade class in order to keep track of all user actions, the latter besides to keep track of user actions it also controls

functionalities access. It is important to mention that their values are high dependent of the number of functionalities available in the system.

Regarding to code scattering, we used two metrics: the Concern diffusion over lines of code (CDLOC) and Scattering metric. As already expected, AOP solution had less code spread over the SPL for all security techniques. After a careful analysis, we noticed that Scattering metric is more sensible then CDLOC metric. It dues to the way in which they are computed, while CDLOC computes the number of shadowed blocks of code, the scattering counts the number of lines of crosscutting code.

A deeper analysis regarding to percentage difference between the techniques implemented using CC and AOP shows that SPL modularization can influence in favor or against the percentage difference between CC and AOP. The CDC metric shows that more modularized the SPL code, the greater the advantage of AOP in relation to CC. On the other hand, the use of the facade design pattern tends to equalize this percentual difference, since the system functionalities are concentrated in this facade class. It can be seen in those techniques that implement log or access control to basic system functionalities. Appendix 8 Table 41, Table 43 and Table 45 show the percentage difference between the techniques implemented when using CC and AOP for separation of concerns metrics.

Authorization. Table 19 shows that comparing CC implementations only CDLOC and Scattering metrics did not show a statistically significant difference, it happens since the difference among the authorization techniques are their specific rules, instead of code responsible to apply the authorization over the functionalities which are the same among them. It is reinforced by the significant difference among them when considering CDC and CDO metrics. Considering AOP implementations, besides to show the same behavior as previously explained for metrics CDLOC and Scattering in CC implementations, CDO also showed the absence of statistically significant difference since AOP implementation isolates the crosscutting code inside the aspect, standardizing the implementations. In summary, the **null hypothesis** H_{02} was rejected for CDC, CDO when considering CC implementations and only CDC when considering AOP implementations. Table 20 shows the power effect, effect size, and magnitude for each metric.

Detect Attack. Considering the CDC metric, we can identify three groups which maintained the same ordering for CC and AOP. The groups are composed of *(i)* Verify Storage Integrity (T3), *(ii)* Verify Message Integrity (T1 and T2) and *(iii)* Maintain Audit Trail (T4), Identify intrusion by behavior (T5). Starting from the security technique that presents the best results for separation of concerns. From the CDO point of view, the techniques T4 and T5 were ordered differently when evaluating AOP and CC. It is explained since to implement technique T5 all methods from the facade class received a unique score number, which is further used in the algorithm responsible to analyze the

Tabela 19 – Comparison of Separation of concern metrics from CC and AOP implementations.

	CDC			CDO			CDLOC			Scattering		
CC	k	p-value	Rand. p-value	k	p-value	Rand. p-value	k	p-value	Rand. p-value	k	p-value	Rand. p-value
Authorization	14.5385	0.0125	0.0002	14.4269	0.0131	0.0002	7.5535	0.1826	0.1728	6.0058	0.3057	0.3296
Detect Tec.	13.3412	0.0097	0	13.0333	0.0111	0	12.3122	0.0152	0.0002	12.3	0.0153	0.0002
Resist Tec.	27.7989	0.001	0	26.7893	0.0015	0	15.5162	0.0777	0.041	19.695	0.0199	0.0024
	w	p-value	Rand. p-value	w	p-value	Rand. p-value	w	p-value	Rand. p-value	w	p-value	Rand. p-value
Detect vs Resist	56.5	0	1	80	0.0005	0.9999	258	0.4255	0.2168	48	0	0.903
AOP	k	p-value	Rand. p-value	k	p-value	Rand. p-value	k	p-value	Rand. p-value	k	p-value	Rand. p-value
Authorization	16.1677	0.0064	0	9.2105	0.101	0.071	0.8941	0.9706	0.9803	0.3851	0.9957	0.9973
Detect Tec.	13.3412	0.0097	0	13.0567	0.011	0	9.6215	0.0473	0.0175	9.4875	0.05	0.0194
Resist Tec.	28.5375	0.0008	0	25.2323	0.0027	0	14.7347	0.0985	0.0625	15.8675	0.0697	0.0343
	w	p-value	Rand. p-value	w	p-value	Rand. p-value	w	p-value	Rand. p-value	w	p-value	Rand. p-value
Detect vs Resist	76.5	0.0003	0.9999	51	0	1	109	0.005	0.998	93.5	0.0015	0.9995

Tabela 20 – Power Effect, Effect size and magnitude of Separation of Concerns metrics.

	CDC			CDO			CDLOC			Scattering		
CC	Power Effect	Effect Size	Mag.	Power Effect	Effect Size	Mag.	Power Effect	Effect Size	Mag.	Power Effect	Effect Size	Mag.
Authorization	1	0.3304	high	1	0.3279	high	1	0.1717	medium	1	0.1365	medium
Detect	1	0.3032	high	1	0.2962	high	1	0.2798	"high"	1	0.2795	"high"
Resist	1	0.6318	very high	1	0.6088	very high	1	0.3526	high	1	0.4476	high
AOP	Power Effect	Effect Size	Mag.	Power Effect	Effect Size	Mag.	Power Effect	Effect Size	Mag.	Power Effect	Effect Size	Mag.
Authorization	1	0.3674	high	1	0.2093	medium	0.3482	0.0203	low	0.0927	0.0088	"negligible"
Detect	1	0.3032	high	1	0.2967	high	1	0.2187	medium	1	0.2156	medium
Resist	1	0.6486	very high	1	0.5735	"very high"	1	0.3349	high	1	0.3606	high

user behavior (see Listing 5.6). This way, it was not possible to isolate in the aspect all crosscutting code inserted in facade methods. It is reinforced by CDLOC and Scattering metrics results. All this information can be viewed in Appendix 8 from Figure 44 to Figure 51. In summary, Table 19 shows a statistically significant difference in all separation of concerns metrics when implementing the techniques using CC and AOP.

Código 5.6 – Portion of the implementation of technique T5.

```
1 ...
2     public void removeLogin(int idUser) throws LoginNotFoundException, RepositoryException,
          LoginAlreadyInsertedException{
3       //#if ${technique5} == "T"
4       Log.getInstance().insert(RiSEEventLoginScreen.getLoggedUser().getIdUser(),2 );
5       Log.calcProbabilidade(RiSEEventLoginScreen.getLoggedUser().getIdUser(),2);
6       //#endif
7       logins.remove(idUser);
8     }
9 ...
```

Resist Attack. Table 19 shows statistical significant difference for all metrics except for CDLOC when considering AOP implementation. It may be due to T15 in

which the CDLOC metric shows to be less efficient that Scattering metric, as reinforced by Table 16. Analyzing CDC and CDO point of views, both metrics preserved the same behavior (ordination) considering the implementations of the techniques using CC and AOP which was T15, T8, T11, T12, T13, T9, T14 and T10. Regarding to CDLOC and Scattering, AOP implementations shows to be more equally distributed, while in CC implementation, T13 shows different behavior since it implements besides the access control it also keeps track of user actions.

Detect Attack vs Resist Attack. Although Table 19 shows the absence of statistically significant difference between detect attack vs resist attack techniques, an in-depth analysis in Appendix 8 (from Figure 44 to Figure 51) show that detect attack techniques have less widespread code making it more modular, easy to understand and maintain. It is important to note that Authentication techniques (T6 and T7) and Hide Data by encryption (T15) are similar to detect attack techniques. The **null hypothesis** H_{03} was not rejected.

Metrics Analysis. CDC metric may not be a good indicator to evaluate the separation of concern attribute when comparing CC and AOP implementations since the main difference between CC and AOP was the crosscutting code which was isolated inside the aspect. This code difference was only significant for the Manage security information (T14) technique. The CDLOC metric seems to be less sensible to assess code scattering since it counts blocks of code instead of lines of code. This way, one block can have 10 lines of code or 100 lines of code.

5.8.3 Lack of Cohesion

The cohesion indicates whether a class represents a single abstraction or multiple abstractions. If the class or aspect represents more than one abstraction, it should be refactored into more than one class or aspect, each of which represents a single abstraction. In order to investigate these characteristics, the Lack of Cohesion over operations (LCOO) was used. It is important to state that the high LCOO value means low cohesion, which results in classes with more than one abstraction, difficulty to understand the code and difficulty to maintain. Table 21 shows the results from the statistical analysis performed to compare both AOP and CC with respect to lack of cohesion by applying the statistical methods previously mentioned and confirmed by randomization test p-value. Table 22 shows the power effect for each metric. For more details regarding to cliff's (δ), magnitude and Cohen's d, see Appendix 8 Tables 50 and 51.

AOP vs CC. Similar to the separation of concern internal attribute, the data gathered reveals two groups: *(i)* represents the security techniques solutions in which the AOP solution had shown as superior and *(ii)* involves the security techniques in

Tabela 21 – Lack of Coheison comparison between CC vs AOP implementations.

Techniques	LCOO		
Techniques	w	p-value	Rand. p-value
T01	8	0.1212	0.0998
T02	8	0.1212	0.0998
T03	6	0.4867	0.3487
T04	8	0.1266	0.1006
T05	7	0.2752	0.1996
T06	8	0.1266	0.0999
T07	8	0.1266	0.1
T08	8	0.1266	0.0996
T09	8	0.1266	0.0999
T10	9	0.0495	0.049
T11	8	0.1266	0.1003
T12	8	0.1266	0.0996
T13	8	0.1266	0.0991
T14	8	0.1266	0.1011
T15	7.5	0.1213	0.2004

Tabela 22 – Power effect for Cohesion Metrics.

Techniques	LCOO		
Techniques	(δ) of Cliff	Magnitude	Power Test
T01	-0.7778	large	0.1681
T02	-0.7778	large	0.1616
T03	-0.3333	medium	0.0552
T04	-0.7778	large	0.3122
T05	-0.5556	large	0.2417
T06	-0.7778	large	0.3107
T07	-0.7778	large	0.3036
T08	-0.7778	large	0.279
T09	-0.7778	large	0.2937
T10	-1	large	0.3655
T11	-0.7778	large	0.2984
T12	-0.7778	large	0.2758
T13	-0.7778	large	0.2818
T14	-0.7778	large	0.2967
T15	0.6667	large	0.1588

which the variability implementation mechanism had the same effect. The summary is showed in Table 23.

The **null hypothesis** H_{01} was only rejected by LCOO metric for the Multilevel Security Design Pattern (T10) technique. It happened because T10 is the technique with the greatest number of classes, methods, and attributes, and using the AOP implementation the code of the technique that before was spread in the facade now is the aspect of the technique. It is reinforced by Appendix 8, Figure 42 and Figure 43.

Authorization. Table 24 shows that there is no significant difference for CC or

Tabela 23 – Summary table lack of cohesion.

Security Technique	**Lack of Cohesion**	
Verify Message Integrity (T1 and T2)	LCOO	There is a suggestive evidence that AOP and CC results have no statistical significant difference for T1 and T2.
Verify Storage Integrity (T3)	LCOO	There is a suggestive evidence that AOP and CC results have no statistical significant difference for T3.
Maintain Audit Trail (T4)	LCOO	There is a suggestive evidence that AOP and CC results have no statistical significant difference for T4.
Identify intrusion by behavior (T5)	LCOO	There is a suggestive evidence that AOP and CC results have no statistical significant difference for T5.
Authenticate subject (T6 e T7)	LCOO	There is a suggestive evidence that AOP and CC results have no statistical significant difference for T6 and T7.
Authorize subject (from T8 to T13)	LCOO	There is a suggestive evidence that AOP and CC results have no statistical significant difference for T8. There is a suggestive evidence that AOP and CC results have no statistical significant difference for T9. There is a suggestive evidence that AOP implementation is better for T10. There is a suggestive evidence that AOP and CC results have no statistical significant difference for T11. There is a suggestive evidence that AOP and CC results have no statistical significant difference for T12. There is a suggestive evidence that AOP and CC results have no statistical significant difference for T13.
Manage security information (T14)	LCOO	There is a suggestive evidence that AOP and CC results have no statistical significant difference for T14.
Hide Data by encryption (T15)	LCOO	There is a suggestive evidence that AOP and CC results have no statistical significant difference for T15.

AOP, this way the **null hypothesis** H_{02} was not rejected and Table 25 shows the power effect, effect size, and magnitude of each metric. It was reinforced by the fact that the authorization techniques have really similar implementations, with classes to implement their core rules and code spread over the screens and functionalities to control the access.

Detect Attack. Although Table 24 shows that there is no statistically significant difference among the detect attack techniques, we can see two extremes. On one extreme, the Verify Storage Integrity (T3) technique with the lowest LCOO value, since

its code really specific. On the other hand, the Identify intrusion by behavior (T5) technique which besides the code to trace which functionality the user has used, it also calculates its behavior.

Tabela 24 – Comparison of Lack of Cohesion metrics from CC and AOP implementations.

	LCOO		
CC	k	p-value	Rand p-value
Authorization	4.1345	0.5302	0.5805
Detect Tec.	8.0811	0.0887	0.0588
Resist Tec.	11.6426	0.2342	0.2216
	w	p-value	Rand p-value
Detect vs Resist	62	0.0001	1
AOP	k	p-value	Rand p-value
Authorization	9.2807	0.0984	0.068
Detect Tec.	7.6607	0.1048	0.0789
Resist Tec.	24.6559	0.0034	0
	w	p-value	Rand p-value
Detect vs Resist	41	0	1

Tabela 25 – Power Effect, Effect size and magnitude of Lack os Cohesion metric.

	LCOO		
CC	Power Effect	Effect Size	Mag.
Authorization	1	0.094	medium
Detect	1	0.1837	medium
Resist	1	0.2646	high
AOP	Power Effect	Effect Size	Mag.
Authorization	1	0.2109	medium
Detect	1	0.1741	medium
Resist	1	0.5604	very high

Resist Attack. Table 24 shows the statistically significant difference only for AOP implementations. Figures 29 and Figure 30 show that Multilevel Security Design Pattern (T10) and Manage security information (T14) are contributing to this statistical difference. The reason why T14 had its median value higher than T10 is that besides it controls the user access, it also keeps track of user actions.

Detect Attack vs Resist Attack. The **null hypotheses** H_{03} was not rejected, for either variability implementation mechanism AOP and CC. Despite this, a clear difference can be noticed having the techniques of detect attack lower values related to the lack of cohesion metrics which makes the code with high cohesion less costly to be maintained.

Metric Analysis. The variation regarding to LCOO metric for CC implementation is greater than AOP implementations. It happens since most of the methods which

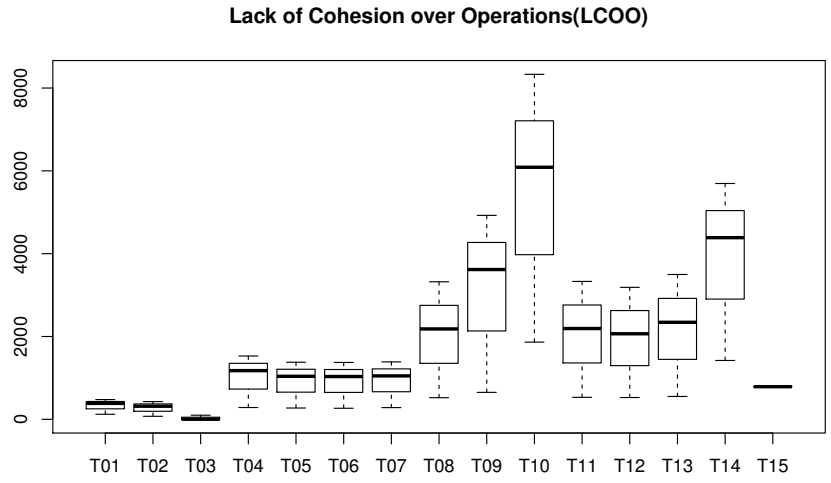

Figura 29 – Lack of Cohesion over Operations(LCOO) for CC.

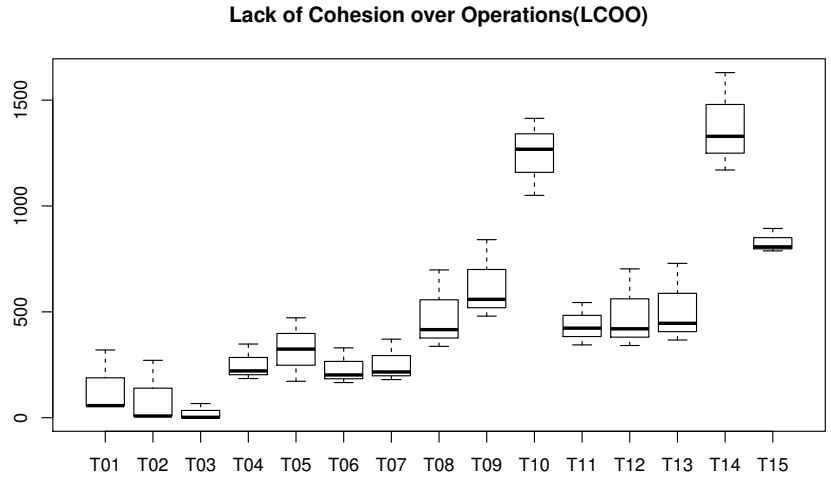

Figura 30 – Lack of Cohesion over Operations(LCOO) for AOP.

appear in CC implementation are invoked by "call"statement in the aspect pointcut. It reduced significantly the variation viewed in CC implementations.

5.8.4 Coupling

It is responsible to evaluate how closely connected two components are. In order to measure it, two metrics were used. The Coupling between components (CBC) which measures the number of other components with which it is coupled and Depth Inheritance Tree (DIT) responsible to calculate the distance from class object/aspect in the inheritance hierarchy. Table 26 summarizes the results from the statistical analysis

performed to compare AOP and CC implementations by applying the statistical methods previously mentioned and confirmed by randomization test p-value. The Table 27 shows the power effect for each metric. For more details regarding to cliff's (δ), magnitude and Cohen's d, see Appendix 8 Tables 52 and 53.

Tabela 26 – Resutls from the comparison between CC vs AOP.

Techniques	CBC			DIT		
Techniques	w	p-value	Rand. p-value	w	p-value	Rand. p-value
T01	9	0.0495	0.0494	7	0.2683	0.149
T02	9	0.0495	0.0498	7	0.2683	0.1494
T03	9	0.0339	0.0499	8	0.099	0.1501
T04	9	0.0495	0.0497	8	0.1212	0.1007
T05	9	0.0495	0.0494	8	0.1212	0.0994
T06	9	0.0495	0.0491	7	0.2612	0.1489
T07	7	0.2752	0.1998	7	0.2612	0.1505
T08	9	0.0495	0.0499	7	0.2612	0.1501
T09	9	0.0495	0.05	5	0.8248	0.5004
T10	9	0.0463	0.0496	7	0.2612	0.1498
T11	9	0.0495	0.0493	7	0.2612	0.1498
T12	9	0.0495	0.0492	7	0.2612	0.1492
T13	9	0.0495	0.0493	7	0.2612	0.1506
T14	7	0.2752	0.2004	7	0.2683	0.1504
T15	9	0.0495	0.0493	7	0.2612	0.1503

Tabela 27 – Power effect for Coupling Metrics.

Techniques	CBC			DIT		
Techniques	(δ) of Cliff	Magnitude	Power Test	(δ) of Cliff	Magnitude	Power Test
T01	1	large	0.8391	0.5556	large	0.0599
T02	1	large	0.8092	0.5556	large	0.0639
T03	1	large	1	0.7778	large	0.3741
T04	1	large	0.6823	0.7778	large	0.1336
T05	1	large	0.5181	0.7778	large	0.1306
T06	1	large	0.4029	0.5556	large	0.0507
T07	0.5556	large	0.2	0.5556	large	0.0507
T08	1	large	0.9846	0.5556	large	0.0509
T09	1	large	0.9982	-0.1111	negligible	0.0818
T10	1	large	0.9989	0.5556	large	0.0502
T11	1	large	0.9908	0.5556	large	0.0509
T12	1	large	0.9906	0.5556	large	0.0508
T13	1	large	0.9671	0.5556	large	0.0508
T14	0.5556	large	0.0674	0.5556	large	0.0504
T15	1	large	0.5183	0.5556	large	0.1249

AOP vs CC. The data gathered from the analysis revealed two distinct groups: *(i)* represents the security techniques solutions in which the CC solution had shown as superior and *(ii)* involves the security techniques in which the variability implementation mechanism had the same effect. All these groups can be viewed in Table 28.

Security Technique	**Coupling**	
Verify Message Integrity (T1 and T2)	CBC	There is a suggestive evidence that CC implementation is better for T1 and T2.
	DIT	There is a suggestive evidence that AOP and CC results have no statistical significant difference for T1 and T2.
Verify Storage Integrity (T3)	CBC	There is a suggestive evidence that CC implementation is better for T3.
	DIT	There is a suggestive evidence that AOP and CC results have no statistical significant difference for T3.
Maintain Audit Trail (T4)	CBC	There is a suggestive evidence that CC implementation is better for T4.
	DIT	There is a suggestive evidence that AOP and CC results have no statistical significant difference for T4.
Identify intrusion by behavior (T5)	CBC	There is a suggestive evidence that CC implementation is better for T5.
	DIT	There is a suggestive evidence that AOP and CC results have no statistical significant difference for T5.
Authenticate subject (T6 e T7)	CBC	There is a suggestive evidence that CC implementation is better for T6. There is a suggestive evidence that AOP and CC results have no statistical significant difference for T7.
	DIT	There is a suggestive evidence that AOP and CC results have no statistical significant difference for T6 e T7.
Authorize subject (from T8 to T13)	CBC	There is a suggestive evidence that CC implementation is better from T8 to T13
	DIT	There is a suggestive evidence that AOP and CC results have no statistical significant difference from T8 to T13.
Manage security information (T14)	CBC	There is a suggestive evidence that AOP and CC results have no statistical significant difference for T14.

Continued on next page

Tabela 28 – *Continued from previous page*

Security Technique	**Coupling**	
	DIT	There is a suggestive evidence that AOP and CC results have no statistical significant difference for T14.
Hide Data by encryption (T15)	CBC	There is a suggestive evidence that CC implementation is better for T15.
	DIT	There is a suggestive evidence that AOP and CC results have no statistical significant difference for T15.

Tabela 28 – Summary table coupling.

The **null hypothesis** H_{01} was rejected considering CBC metric since the CC implementation showed to be more effective for all studied techniques. It happens since CC implementation tends to be less coupled than AOP. The coupling in OO solution it is smaller than AOP implementation since the latter import some classes to implement the crosscutting concern code.

In the context of DIT metric, there is no difference between AOP and CC implementations. It happened because none of the security techniques implementation required inheritance during its implementation. It made AOP and CC showed similar results.

Authorization. Table 29 shows the p-values from the statistical analysis performed. Based on that, there is a statistically significant difference among the authorization techniques, which rejects the **null hypothesis** H_{02} in both AOP and CC implementations when considering CDC metric. Table 30 shows the power effect, effect size, and magnitude of each metric. It can be seen in Figure 31 and Figure 32, CC and AOP implementations had similar results, the only difference is the aspect that now isolates the code that was scattered. In the context of DIT metric, the authorization techniques did not show any difference, including maintaining a very similar variation. Regarding to coupling, the authorization techniques can be ordered as follow, starting from the less coupled: T8, T11, T13, T12, T9 and T15.

Detect Attack. For both metrics (CBC and DIT), the detect attack techniques followed the same ordering starting with the lowest CBC for the largest: T03, T02, T01, T05, T04. T03 had the lowest value, which facilitates the maintenance since its code are really punctual and its implementation practically does not depend on other classes.

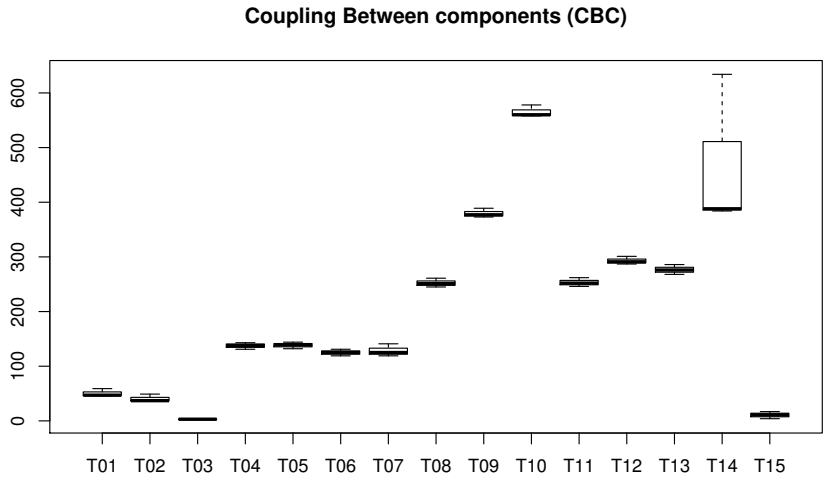

Figura 31 – Coupling Between components(CBC) for CC.

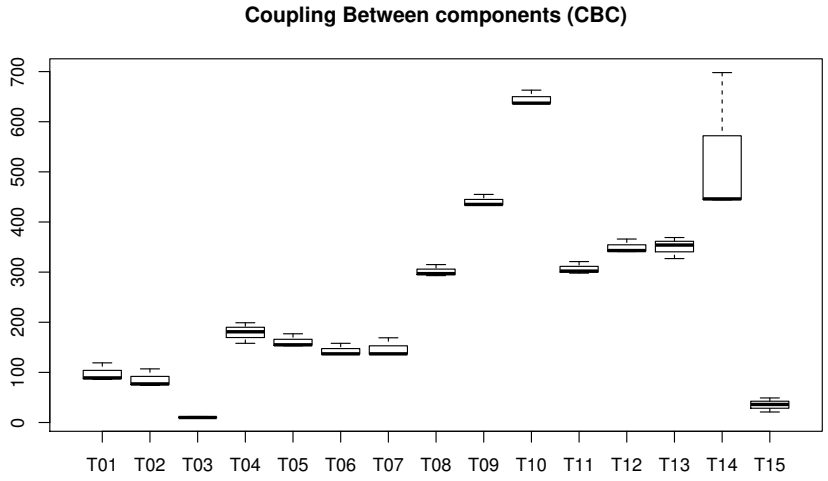

Figura 32 – Coupling Between components(CBC) for AOP.

The techniques T04 and T05 had the highest values in relation to this metric. It happens since these techniques besides to implement the algorithm responsible for keep track of user activities (T05), it also evaluates any user change behavior.

Resist Attack. Concerning to CBC metric, the results are similar for both implementations CC and AOP, as well as, they reveled few variation regarding to metric values. However, T14 shows a different behavior regarding to its variation since, besides to perform authorization control, it also implements the algorithm responsible to encrypt security information such as, email, username, and password. This way, depending on the number of entities to be controlled the higher the number of entities in the SPL

Tabela 29 – Comparison of Coupling metrics from CC and AOP implementations.

	CBC			DIT		
CC	k	p-value	Rand p-value	k	p-value	Rand p-value
Among	16.1579	0.0064	0	9.3856	0.0946	0.0638
Detect Tec.	12.5228	0.0139	0.0001	3.4395	0.4871	0.5335
Resist Tec.	28.1007	0.0009	0	20.3698	0.0158	0.0013
	w	p-value	Rand p-value	w	p-value	Rand p-value
Detect vs Resist	67.5	0.0001	1	56.5	0	1
AOP	k	p-value	Rand p-value	k	p-value	Rand p-value
Authorization	15.7999	0.0074	0	8.1062	0.1505	0.1319
Detect Tec.	12.7895	0.0124	0	12.6799	0.013	0.0001
Resist Tec.	27.9692	0.001	0	19.8009	0.0192	0.0021
	w	p-value	Rand p-value	w	p-value	Rand p-value
Detect vs Resist	66.5	0.0001	1	88.5	0.001	0.9997

Tabela 30 – Power Effect, Effect size and magnitude of Coupling metrics.

	CBC			DIT		
CC	Power Effect	Effect Size	Mag.	Power Effect	Effect Size	Mag.
Authorization	1	0.3672	"high"	1	0.2133	"medium"
Detect	1	0.2846	high	1	0.0782	low
Resist	1	0.6387	very high	1	0.463	high
AOP	Power Effect	Effect Size	Mag.	Power Effect	Effect Size	Mag.
Authorization	1	0.3591	high	1	0.1842	medium
Detect	1	0.2907	"high"	1	0.2882	"high"
Resist	1	0.6357	very high	1	0.45	high

code the greater the variance. T15 had the lowest values since it only depends on repository classes. In the context of DIT metric, the results followed the same order for both solutions CC and AOP starting from T15, T08, T09, T11, T12, T13, T14 and T10. It happens since the metric calculates the distance from class object/aspect in the inheritance hierarchy. CC and AOP solutions differ only regarding to crosscutting concern code. It also explains the variation in the metrics values.

Detect Attack vs Resist Attack. The **null hypotheses** H_{03} was not rejected, probably influenced by T15 metrics value and T08 and T09 metric values variation. Analyzing Appendix 8 from Figures 52 to Figure 55, we can see that detect attack techniques is less costly than resist attack techniques. The classes used to implement detect attack techniques are less coupled than the classes used in resist attack techniques.

Metrics Analysis. When security techniques do not need inheritance in their implementations, DIT metrics may not be a good parameter to differentiate these variability implementation mechanisms since AOP implementation will only differ regarding to crosscutting concern code. In addition, CBC metric seems to be a good indicator to differentiate AOP and CC as we can see in the results from Table 28.

5.8.5 Feature Interaction

The feature interaction happens when the behavior of one feature influences the presence of another feature or a set of features (APEL et al., 2013). In our work, this interaction can happen between security tactics and techniques, as well as, other quality attributes (such as, availability) and functional features.

Based on the implementations and the purpose of each security technique more than one security technique can be used in the same software product except that techniques which have more than one implementation, such as verify message integrity (T1 and T2), Authenticate subject (T6 and T7) and Authorize subject(from T8 to T13). Regarding to feature interaction between security techniques, there is an implicit relationship between authorize subject and authenticate subject techniques, since to authorize you need to identify and authenticate a subject (user). This way, the authorize subject techniques need to interact with one authenticate subject technique.

Considering the interaction between quality attributes there are some scenarios in which this relation can be observed. In general, recovering security tactics share many techniques as availability since in both scenarios they involve returning the system to a consistent state prior to any attack (BASS; CLEMENTS; KAZMAN, 2012). Another scenario in which some tradeoffs can happen is during the intrusion by behavior identification (T5), verify message integrity (T1 and T2) and Hide data by encryption (T15). The algorithm used to identify the intrusion can influence system performance since it needs to analyze a considerable amount of data or even use an intelligent algorithm to learn based on the user previously executions. The same happens to verifying message integrity (T1 and T2) and Hide data by encryption (T15) in which the cryptography algorithm can influence in system performance, maintainability and complexity.

5.9 Main Findings

As discussed in the previous section, it is not surprising that there are some differences regarding to security techniques and the variability mechanisms implementation. This was expected, but the results from the analysis illustrated other points that should be taken into account when implementing security techniques:

- The use of Facade design pattern may facilitate the implementation of security techniques such as: T01, T02, T04, T05, T13 and T14. Considering the CC implementations it facilitates since most of the code will be concentrated inside the facade class. Regarding to AOP implementation, it facilitates since the crosscutting technique code will be concentrated inside the aspect.

- SPL code modularity can influence the results of some metrics, such as CDLOC, Scattering, CDC and CBC. It happens when the security techniques had its code spread over different classes and packages. The less the code of the line is modularized the less the code of the technique will be spread by different classes.

- The Scattering metric shows to be more sensible than CDLOC metric since CDLOC calculates the scattered code by considering blocks of crosscutting concerns code, while Scattering counts each line of code.

- NOA metric seems to be a bad choice to differentiate AOP and CC implementations since the use of intertype declarations leaves the code very similar in relation to AOP and CC solutions.

- The results from the analysis of the three experiments reinforce the advantages of AOP regarding to Separation of concern metrics.

- The number of screens to be managed by access control has a greater impact on conditional compilation when implementing authorization techniques. This may be greater for T11 in which the code should consider timeout on different system screens.

- The use of encryption on code level on techniques T1, T2, T14 and T15 increases the implementation complexity. Part of this complexity can be decreased when the database offer and manages encrypt data.

- The Session Pattern (T11) can be more complex than other authorization techniques as the number of system screens increases since a screen timeout should be set on each system screen.

- The metrics which had the greatest impact regarding to the different SPL code were: LCOO, DIT, WOC, Scattering, CDLOC and CDC. It happens due to the number of different entities used in each SPL. Even with the variation, a pattern can be found in the results as explained in the previous section.

In order to summarize our main findings to provide a more engineering view about it, we built the conceptual map presented in Figure 33. We believe that it is a starting point towards the definition of a decision model. More studies are necessary in order to identify and confirm the existing difference between the security techniques and the variability implementation mechanisms and provide a base to define a decision model. It shows the results regarding to the different between CC and AOP implementations when exists a statistical difference between them.

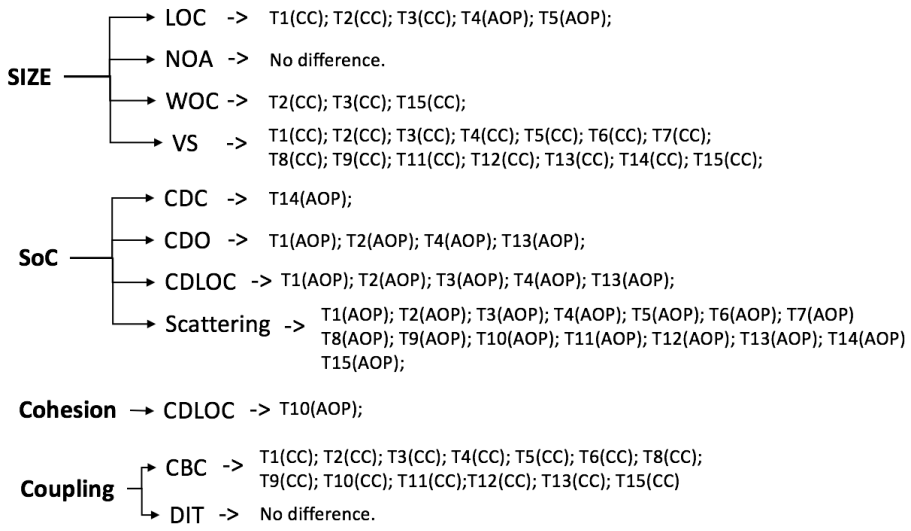

Figura 33 – A conceptual map summarizing the results would be useful for researchers and practitioners.

5.10 Threats to Validity

There are some threats to the validity of our results, briefly described along with the mitigation strategy for each are.

Metrics Selection: There are a variety of metrics to analyze maintainability as stated by (SARAIVA et al., 2015). This large amount of metrics brings many challenges to researchers and practitioners, such as scattered descriptions, lack of information regarding to what they measure and how and metrics with the same name but measure different attributes (SARAIVA et al., 2015). To mitigate these threats we choose a set of already empirically validated metrics (SANT'ANNA et al., 2003; GARCIA et al., 2005; KULESZA et al., 2006a; GREENWOOD et al., 2007; FIGUEIREDO et al., 2008; EADDY et al., 2008; FIGUEIREDO et al., 2009; DANTAS; GARCIA, 2010) to be used in our study.

Metrics Tools: From a practical point of view, software engineers need to be aware that the metrics are tool dependent, and that these differences are a threat to the study results (LINCKE; LUNDBERG; LöWE, 2008). In order to mitigate this threat most of the metrics were collected by a weel-known metric tools and recommended by other researchers in their studies.

Metric Extraction: When the metric had to be partially extracted manually for AOP implementations, they were checked twice, by an M.Sc and Ph.D. student. Moreover, A metric was collected for all releases so that another metric could be collected. In this way, we were able to maintain the same pattern and procedure in data collection to avoid any mistake or extraction errors. Finally, Python scripts[16] were also

[16] Available at: <https://github.com/pamsn/RiSEEventSPL/tree/master/pythonscripts>

used to extract the data, as well as, crosscheck the metric tools outputs. We believe that these operations were useful to minimize any bias potentially introduced in the data extraction process.

SPL TestBeds: In order to perform the maintainability analysis and implement to different security techniques, it was required construction to different TestBeds. Our work considered three different tests beds being one developed by the author of this thesis and the other two developed by different students with a different background. The threat regarding to the development was partially mitigated by asking different students to implement and the threat regarding to the number of tests beds was partially mitigated by implementing three test beds.

TestBeds Refactoring: Some code refactoring were necessary to be performed in order to allow the implementation of the security techniques in all SPL domains. For example, during the Store SPL development, an authentication mechanism to control the user access was implemented by the students. As the authentication is one of the security techniques evaluated by our study, it needed to be removed. Thus, the user access code was removed, and a refactoring was performed both in the SPL code and database scripts.

TestBeds Testing: Although the SPL has been tested throughout the development process, the code has not undergone a test phase which may have left implementation errors. These bugs may have been propagated as security techniques have been implemented.

Architectural and Implementation Style: It happens due to learning and fatigue effects experienced by the participants. The learning effect was mitigated adopting the procedure to spread corrections and refactoring for all already implemented security techniques. This way, as the maturity level of the developer (researcher) increased, refactoring was performed in all releases accordingly. In addition, the way in which the security techniques were developed was checked by another Ph.D. researcher.

Security Technique Implementation: Different security metrics were selected to be implemented in different SPL domains (TestBeds). As previously mentioned, there is a semantic gap between the tactics description and the way used to implement it. In order to mitigate this threat, the class diagram was used to support the tactic and techniques implementations.

Trade-offs Analysis: Our study was performed considering the security point of view without analyzing how they impact on different quality attributes, such as performance and availability.

Statistical Analysis: The statistical analysis was performed using mainly non-parametric tests that do not assume data normality and are also well-suited for use

on small samples (HADAR et al., 2013). Moreover, the randomization test was used in order to confirm the methods precision and reliability. In addition, a strength p-value scale was used in order to show how strong are the conclusions, the power effect of statistical methods used in the study was also shown.

Randomization test: Although the randomization test was used to confirm the statistical methods precision, the main disadvantage of this test is that the conclusions obtained are restricted to each set of data and type of problem, and it is not possible to generalize to the population (VIOLA, 2007)

Data Aggregation: Another concern related to the conclusion validity in this research is the analysis and interpretation of the data aggregated from all three experiments. The motivation for this aggregation is that while each experiment was performed using one SPL, aggregating data to include results obtained from the implementation of each security techniques in each SPL, it contributed to the statistical robustness of the conclusions. However, aggregating data from different security technique implementation may threat validity due to the way in which the SPLs were developed. This threat was mitigated in this research by selecting different test beds in different domains. Moreover, the results obtained in the three experiments were similar in their direction, and the only difference (if any) was in the metric values, which is to be expected when using different SPLs.

It is important to mention that although some action was undertaken to mitigate the threats they continue to exist to a greater or lesser extent. New studies are recommended to reinforce our results.

5.11 Related Work

In this section, we provide an overview of some related work.

The authors in (GEORG; FRANCE; RAY, 2002) described how design-level aspects can be used to encapsulate security concerns that can be woven into models of software design. The authors limited the study analyzes the impact of security concerns on other functional concerns. In addition, it was performed at design level using a toy example, without considering the impact at the code level, how it interferes in quality attributes or even another variability implementation mechanism.

The authors in (D.; M.; C., 2006), compared several security patterns (Authorization and Authentication) following an approach for describing the security degree of the patterns, and indicating a fulfillment or not of the properties and attributes common to all security systems. The study evaluated different ways to implement the authorization patterns and qualitatively assessed the patterns regarding to availability, auditability, performance, confidentiality, and so on. The study is restricted to a qualitative analysis

of these patterns regarding to most commonly used attributes and security properties in the security domain.

In (DANIEL; EDUARDO; MARIO, 2010), the authors defined a security requirement framework to support the development of security SPLs and their derived products. They considered the most relevant security standards with regard to the management of security requirements. Then, the process was formally specified with SPEM 2.0 and the repository was specified with an XML grammar. The authors did not consider neither design nor implementation, as well as, did not evaluate the impact on quality attributes.

(AYED et al., 2013) described how to manage security policies within distributed systems based on the Aspect-Oriented Approach. The authors showed how the security policies are modeled and how an interpreter module is able to manage and use the knowledge derived from the policies. In addition, it also makes the link between the knowledge defined by the policy interpreter and the aspect generation module which gives the set of aspects to be defined and then weaved. Although to consider security and aspect-oriented approach the authors did not performed a qualitative analysis regarding the impact of such aspect on quality attributes.

In (GAIA et al., 2014), the authors investigated whether the use of aspects and features, through Aspectual Feature Modules (AFM) approach, facilitates the evolution of SPLs. They used two SPLs in which four different variability mechanisms were applied, such as: *(i)* feature modules aspects and aspects of refinements of AFM, *(ii)* aspects of aspect-oriented programming (AOP), *(iii)* feature modules of feature-oriented programming (FOP), and *(iv)* conditional compilation (CC) with object-oriented programming. Next, metrics for change propagation and modularity were calculated in the context where the SPL has been evolved with addition and modification of crosscutting concerns. In (FERREIRA et al., 2014), the authors also investigated how FOP, conditional compilation and object-oriented design patterns impacts on change propagation and modularity when used to implement crosscutting concerns in two different SPLs. Both studies there is no quality attributes assessment, the studies analyze how different variability implementation mechanisms impact on modularity throughout functional features evolution.

In (MYLLARNIEMI, 2015), the authors investigated why and how to vary quality attributes in SPL. Instead of focusing on how to represent quality attribute variabilities, they focused on understanding the phenomenon of how specific quality attributes vary. Thus, they conducted a systematic literature review (MYLLäRNIEMI; RAATIKAINEN; MäNNISTö, 2012) on quality attribute variability, and performed two case studies (MYLLäRNIEMI; RAATIKAINEN; MäNNISTö, 2006; MYLLäRNIEMI; RAATIKAINEN; MäNNISTö, 2015) on performance variability, and constructed a design theory and artifacts addressing security variability. Compared to our study, the authors focused on

understanding how some attributes of quality vary and based on the results of two case studies the authors constructed an approach that considers modeling such attributes still in the design phase. No comparison is made between these different ways of variation nor how they can impact on the code.

In (RYOO; KAZMAN; ANAND, 2015), the authors combine architectural analysis techniques based on tactics, patterns, and vulnerabilities in order to understand how to create a secure architecture. They proposed three novel approaches to architectural analysis for security: vulnerability-oriented architectural analysis (VoAA), patterns-oriented architectural analysis (PoAA), and tactic-oriented architectural analysis (ToAA). The authors did not evaluate the impact of such security in the architecture or code, neither provide evidence or any assessment regarding how to implement (code level) security.

In (HORCAS; PINTO; FUENTES, 2016), the authors provided means for software architects to focus only on application functionality, without having to worry about functional quality attributes. It is achieved by modeling quality attributes separately from the application functionality using SPL approach. This way, they defined a generic process to model and automatically inject quality attributes into the application without breaking the base architecture. It is important to mention that the authors did not provide any information regarding to how to implement (code level) the quality attributes neither their quantitative assessment.

(CERVANTES et al., 2016), recognized that the architecture alone is insufficient, just as coding or process alone is insufficient to ensure secure software. In order to support the assertion that frameworks are essential to a secure system, they studied three industrial projects and one open source project to gather empirical evidence about security practices. The analysis aimed to understand how secure these systems were, how difficult and costly it was to create the security controls, and how much it would cost to maintain them in the future. Despite to evaluate the trade-off space between the costs and benefits of different architectural approaches to security, the study did not indicate how the tactics were implemented in each case study, as well as did not evaluate through code metrics the security maintenance.

5.12 Conclusion

In this chapter, we presented the results of a family of experiments regarding to implementation of security techniques in SPL architectures.

Initially, we provided a way to implement at code level different security tactics reducing the existing gap between security tactics and their implementation. Next, the security techniques were assessed and classified regarding to maintainability (size,

separation of concerns, coupling, and cohesion). Finally, the conditional compilation and aspect-oriented programming were also evaluated in order to understand which variability mechanism are less costly regarding to maintainability. In addition, to be an important step in the maturation of the research area, it also provides important design decision which has its cost decreased the sooner the decision is taken.

According to the analysis performed from the aggregation of the results from the three experiments, in general, AOP implementation had better results regarding to separation of concerns, while conditional compilation had better results for size and cohesion. The coupling internal attribute had similar results for both variability mechanisms. In the tactics perspective, detect attack techniques are less costly than resist attack techniques regarding to maintainability. The detect attack techniques obtained better results for all internal attributes.

Regarding to the implementation the use of facade pattern facilitates the implementation of techniques to verify message integrity, maintain audit trail and identify intrusion by behavior since this pattern concentrates most of the system functionalities in one class. The authentication techniques are less costly than authorization techniques since besides the user control, the authorization techniques also manage the objects (functionalities) control. Finally, our results may provide important insights and evidence towards the choose of the most suitable technique to implement security at SPL architectures with respect to low impact on maintainability. Although we identified important evidence, future studies are required to increase the reliability and generalization of this evidence.

5.13 Chapter Summary

A useful way to achieve secure systems is based on tactics, the design decisions that influence the achievement of quality attribute response (BASS; CLEMENTS; KAZMAN, 2012). All these decisions should be considered in early design phases since wrong decisions can impact on the overall project and increase its cost.

In this Chapter, three experiments were carried out in order to understand the impact of security techniques on maintainability with respect to size, separation of concerns, coupling, and cohesion. It described possible ways to implement each security technique using aspect-oriented programming and conditional compilation. Moreover, our results also show that some metrics may not be adequate to assess the maintainability of security techniques. While even these results are not enough to define a decision model, it might be used as evidence towards its definition.

Next Chapter presents the conclusions and future work of this thesis.

6 CONCLUSION AND FUTURE WORK

Software Product Lines practices in industry have decreased implementation costs, reduced time-to-market and improved product quality (CLEMENTS; NORTHROP, 2001). However, all these benefits do not come for free, they demand planed and managed reuse, adequate practices for its management and development of commonalities and variabilities, and being capable of dealing with technical and organizational issues. The common and variable properties of the SPL are managed into a rational, though and complex environment. The organization of such properties must enable the derivation of different products through a Product Line Architecture to enable organizations to respond quickly to a redefined mission or to new and changing markets (DIKEL et al., 1997).

Nowadays, with the increasing number of service-oriented applications and a large amount of user data to be managed remotely, it is even more important that the SPL architecture holds security requirements. This security is considered a cross-cutting concern affected by a wide range of architectural decisions, that must be addressed earlier to avoid costly changes in the future. In this context, the most suitable architecture approach to address security is based on tactics, basically defined as design decisions (FERNANDEZ; ASTUDILLO; PEDREZA-GARCIA, 2015). Although design tactics are considered an important approach to improve system quality factors (BASS; CLEMENTS; KAZMAN, 2012), there is a lack of studies regarding how to implement this design decisions and the impact on maintainability. Besides, evidence regarding to the difference of the most suitable way to implement security techniques.

Hence, this present work is an attempt to fix this gap, by providing insights and evidence in order to define a decision model to support software architects on design decisions, based upon architectural design knowledge. It can also be useful for companies developing product lines in which security is an important quality attribute, since security aspects should be treated in early phases, otherwise its cost can overcome the defined budget.

In order to achieve it, three SPL test beds were developed in different domains using different groups of developers. Next, the security techniques were implemented using conditional compilation and aspect-oriented programming as variability mechanism resulting in 90 different versions. After that, eleven metrics were used to assess maintainability through complexity, modularity, understandability, and flexibility. Finally, a family of experiments was performed in order to evaluate and better understand the most suitable variability mechanism to implement SPL security, as well as, classify the existing security tactics and techniques regarding their impact on complexity, modularity,

understandability, and flexibility.

According to the assessment, the aspect-oriented solution presented better results for modularity, understandability, and flexibility since it improved separation of concerns and cohesion. It is important to mention that the difference was not big enough to justify its use. Although it had better results in separation of concerns and cohesion, the conditional compilation showed to be less complex and considering size and coupling easier to understand. Regarding the engineering point of view, these differences can not justify the use of aspect-oriented programming to implement security in SPL since the cost to train the developers can overcome its benefits. It is reinforced by the three experiments results since for all test beds the results followed the same pattern.

Nonetheless, we understand that other domains with different architecture styles could be used in order to assess the application of the security techniques in SPL. It means that other experiments could improve our results and provide more conclusive results. Another aspect to be considered is the maintenance point of view, the experience gained during the implementation of the techniques and further evaluation provide important insights regarding the less costly maintenance to aspect-oriented programming.

The remainder of this chapter is organized as follows: Section 6.1, the concluding remarks are presented. Section 6.2 we sketch the main contributions achieved so far with this investigation. Finally, Section 6.3 discusses future research directions.

6.1 Concluding Remarks

The study analyzed different ways to implement software security tactics considering three SPLs implemented in different domains. It also analyzed how the security techniques are distributed regarding to maintainability when implemented using conditional compilation and aspect-oriented programming as variability implementation mechanisms.

The test beds are important to research since it allows the assessment of new techniques, processes, approaches and so on. It is also essential in studies replications and generalization. In this context, our test beds did not use any development framework since its use can be considered a constraint for other researchers especially when working with metrics extraction. In addition, the test beds developed in our work considered different domains, technology, and code size.

The SPLs were developed by different students, while the security techniques were developed by the author of this thesis in order to minimize any threat regarding to different programming styles. Although some data showed a significant difference,

this difference cannot be greater enough to advocate or justify the use of conditional compilation against aspect-oriented programming.

The Mann-Whitney-Wilcoxon and Kruskal-Wallis methods when purely applied during the hypotheses testing were not sensible enough to show the differences, for this reason, the randomization test was used. It provided a more sensible comparisons allowing us to draw concrete conclusions about which solution is better. We believe that the sample size was a possible reason for the need to apply randomization tests. The Mann-Whitney Test, for instance, has little power with small samples and will always give a p-value greater than 0.05 no matter how much the groups differ.

The quantitative analysis showed some differences between authorization subject techniques, with the Authorization Pattern (T8) technique being the least costly and Multilevel Security Design Pattern (T10) being the most costly when evaluating maintainability. When evaluating the maintainability cost between detect a attack and resist attack for both variability implementation mechanisms, detect attack is less costly than resist attack.

6.2 Main Contributions

The main contributions of this thesis can be summarized as folllows:

- *Test Beds.* Three test beds were developed considering the JAVA programming language and desktop user interface. It is important to mention that none of them makes use of SPL development frameworks which can allow researchers to use it without needing to learn about a specific framework. In addition, it also allows existing metrics tools to extract code parameters with no interference of frameworks characteristics. These test beds have a considerable size and were build considering different developers reducing any bias. As consequence, these test beds are being used by the overall research group and has been essential to establish external research collaborations.

- *Security Techniques implementation.* As previously mentioned, it is important to map the tactics, their respective techniques and the way in which these techniques are realized on software. It is considered an important step to reduce any misunderstanding regarding to how to implement a security technique providing support to software architecture decisions. Our study can be considered an initial attempt in order to reduce this misunderstanding by providing the techniques and how they can be implemented in SPL code.

- *Family of Experiments.* We performed three experimental studies considering the test beds and the application of each security technique. These studies

yielded important insights regarding to the most suitable way to implement software security in SPL considering two variability implementation mechanisms, aspect-oriented programming, and conditional compilation.

Regarding to academic contributions, this thesis generated the following paper:

- RiSE Events - A Testbed for Software Product Lines Experimentation. Published at: The First International Workshop on Variability and Complexity in Software Design.

6.3 Future Work

Due to time constraints and complexity involved to provide the material necessary to perform the research, this work can be seen as an initial research towards to find the most suitable way to implement software security in SPL. Thus, the following issues should be investigated as future work:

- *Test Beds.* Our study was restricted to three test beds, it is important to perform the same assessment in a greater variety of test beds, considering different domains, architecture styles, and sizes. It can provide better insights regarding to the influence that the SPL code has on the implementation of the security techniques.
- *Software security evolution.* The evolution regarding to a new cryptography algorithm, the change of authentication mechanisms such as: biometric, magnetic card or even a new way to manage the user authorization in the system can have a huge impact on the SPL. It is important to understand the impact of those changes to provide guidelines to the architects responsible for software security. In addition, it is important to know which variability implementation mechanisms is most suitable considering the maintenance and evolution costs.
- *Security Degree.* Even subjective it is important to define a set of criteria to evaluate the security techniques regarding to its security degree (e.g., Low, Medium and High). It provides important insights to software architects to decide the most suitable way to implement software security, considering its impact on maintenance aspects and security degree (D.; M.; C., 2006).
- *Combination of Security Techniques.* The combination of security techniques can also be adopted in order to achieve a better level of security. This way, it is important to understand how these security techniques can be combined and how the combination influence on SPL maintenance.

- *Mapping among tactics, techniques and their implementation.* Although this work partially addressed this issue, it is important to define guidelines or patterns for implementation of security tactics. It can reduce the errors generated by different interpretations given to each of tactic and technique descriptions.
- *Variability Mechanisms.* The experiments were restricted to two variability mechanisms that are among the most suitable to implement SPL variability. Indeed, the consideration of other mechanisms such as Object Orientation and Dynamic class loading would enrich the current results.
- *Qualitative Analysis.* A survey with experts is an important step to qualitative analysis the security techniques and the variability implementation mechanisms.
- *Dynamic SPL Analysis.* The evaluation of security techniques on dynamic SPL can provide important insights regarding to how the context may influence on software security options.

7 TESTBED DATA

Tabela 31 – Functional Properties.

ID	Description
F1	*Speaker:* it aims to describe the people responsible for talks during different activities.
F2	*Organizer:* It aims to describe the people responsible for organizing the event activities. It can assume different roles such as: General Chair, Program Chair, Workshop Chair, Tutorial Chair, Proceedings Chair and Panel Chair.
F3	*Reviewer:* It describes the people responsible for reviewing the manuscripts submitted to the event main track, as well as, their workshops.
F4	*Event Program:* This feature generates the event program, composed of activities and its respective schedules.
F5	*Event Important Dates:* It generates the event important dates, such as: Abstracts, Full papers, Notifications and so on.
F6	*Activity Workshop:* It enables the creation and management of workshop tracks.
F7	*Activity Tutorial:* It enables the creation and management of tutorials.
F8	*Activity Panel:* It enables the creation and management of panels.
F9	*Activity Course:* It enables the creation and management of courses.
F10	*Activity Main Track:* It enables the creation and management of event main track.
F11	*Round of Review:* The manuscript review can happens in different rounds (reply and rejoinder), enabling the interaction between authors and reviewers.
F12	*Simple Review:* The manuscript only receives one review which is sent to the authors.
F13	*Registration - User/Activity:* It enables the user registration in event activities.
F14	*Registration - Speaker/Activity:* it enables the speaker registration in some event activities.
F15	*Registration - Organizer/Activity:* it enables the organizer registration in some event activities.
F16	*Frequency per Event:* it generates the list of participants per event.
F17	*Frequency per Activity:* it generates the list of participants per activity.
F18	*List of Authors:* it generates the list of authors per event, to enable the proceedings generation.
F19	*Checking Copy:* it enables the option to print a checking copy to show that a given participant was in a specific activity.
F20	*Certificate:* it prints a certificate regarding to presentation, participation, organization and so on.
F21	*Payment in Cash:* it enables the payment in cash.
F22	*Payment Deposit:* it enables the deposit payment.
F23	*Payment Credit Card:* it enables the credit card payment.
F24	*Partial Submission:* it enables the submission of abstracts and after a defined date, the system enables the attachment uploading.
F25	*Complete Submission:* it enables the abstract and attachment submission.
F26	*Assignment Chair Indication:* it enables the program chair to choose which manuscript will be reviewed by which reviewer, considering its conflict of interest.
F27	*Automatic Assignment:* the system automatically assign the manuscripts to their reviewers.
F28	*Automatic Conflict of Interest:* during the manuscripts assignment, the conflict of interest is automatically solved. It indicates which reviewers cannot review a specific manuscript, based on the reviewer knowledge and authors filiation.
F29	*Notification Deadline:* it notifies the reviewers about the review due date.
F30	*Notification Acceptance/Rejection:* it notifies the authors about the manuscript reviews results.
F31	*Notification Paper Assignment:* it notifies the reviewer about which are the manuscripts that they should review.
F32	*Authors:* it enables the insertion of authors that are not necessarily registered in the system.
F33	*Bugs:* it enables the system users to raise change request to the system support.
F34	*Receipt:* it enables the generation of participants receipts for different event activities.

Tabela 32 – Data set of RiSE Event SPL vs. Security Techniques vs. Conditional Compilation.

RiSE Event SPL	№ of Classes	№ of Interfaces	№ of Methods	№ of Static Methods	№ of Static Attributes	№ of Attributes	№ of Parameters(Value)
Base	496	20	1493	180	199	737	1219
T1	503	21	1514	190	200	743	1247
T01	503	21	1514	190	200	743	1247
T02	503	21	1509	183	202	743	1234
T03	498	20	1493	182	199	737	1225
T04	519	22	1560	192	207	770	1288
T05	519	22	1550	194	208	765	1279
T06	516	21	1545	191	206	764	1273
T07	517	21	1547	192	206	766	1277
T08	534	23	1609	195	211	779	1328
T09	551	25	1678	197	215	795	1378
T10	587	29	1785	207	230	822	1452
T11	535	23	1609	197	212	779	1329
T12	537	23	1614	195	211	780	1328
T13	539	24	1628	198	212	788	1348
T14	553	25	1699	220	226	807	1434
T15	498	20	1514	207	210	749	1279
T01- BASE	7	1	21	10	1	6	28
T02- BASE	7	1	16	3	3	6	15
T03- BASE	2	0	0	2	0	0	6
T04- BASE	23	2	67	12	8	33	69
T05- BASE	23	2	57	14	9	28	60
T06- BASE	20	1	52	11	7	27	54
T07- BASE	21	1	54	12	7	29	58
T08- BASE	38	3	116	15	12	42	109
T09- BASE	55	5	185	17	16	58	159
T10- BASE	91	9	292	27	31	85	233
T11- BASE	39	3	116	17	13	42	110
T12- BASE	41	3	121	15	12	43	109
T13- BASE	43	4	135	18	13	51	129
T14- BASE	57	5	206	40	27	70	215
T15- BASE	2	0	21	27	11	12	60

Tabela 33 – Data set of RiSE Event SPL vs. Security Techniques vs. AspectJ.

RiSE Event SPL	№ of Classes	№ of Interfaces	№ of Methods	№ of Static Methods	№ of Static Attributes	№ of Attributes	№ of Parameters(Value)
Base	496	20	1493	180	199	737	1219
T01	505	21	1530	190	200	743	1262
T02	505	21	1525	183	202	743	1249
T03	499	20	1495	182	199	737	1226
T04	522	22	1578	193	210	771	1305
T05	522	22	1569	195	211	766	1298
T06	518	21	1561	192	209	765	1288
T07	519	21	1565	193	207	763	1293
T08	538	23	1619	196	214	780	1345
T09	555	25	1697	198	218	796	1395
T10	591	29	1806	208	233	823	1471
T11	539	23	1628	198	215	780	1346
T12	541	23	1633	196	214	781	1345
T13	543	24	1649	199	215	789	1367
T14	557	25	1718	225	229	808	1455
T15	499	20	1584	207	210	749	1447
T01- BASE	9	1	37	10	1	6	43
T02- BASE	9	1	32	3	3	6	30
T03- BASE	3	0	2	2	0	0	7
T04- BASE	26	2	85	13	11	34	86
T05- BASE	26	2	76	15	12	29	79
T06- BASE	22	1	68	12	10	28	69
T07- BASE	23	1	72	13	8	26	74
T08- BASE	42	3	126	16	15	43	126
T09- BASE	59	5	204	18	19	59	176
T10- BASE	95	9	313	28	34	86	252
T11- BASE	43	3	135	18	16	43	127
T12- BASE	45	3	140	16	15	44	126
T13- BASE	47	4	156	19	16	52	148
T14- BASE	61	5	225	45	30	71	236
T15- BASE	3	0	91	27	11	12	228

Tabela 34 – Functional Properties.

ID	Description
F1	*User notification:* it aims to notify the users regarding different actions.
F2	*Insert User:* it aims to insert different users in the system.
F3	*Remove User:* it aims to remove users from the system.
F4	*Update User:* it aims to update different users attributes.
F5	*Change User Account:* it aims to change a user among different users permission groups.
F6	*User Status:* it shows the user status in the system.
F6	*User profile display:* it shows the user profile information previously stored in the system.
F7	*Insert Product:* it aims to add different products in the system.
F8	*Remove Product:* it aims to remove products from the system.
F9	*Update Product:* it aims to update different product attributes.
F10	*Sell Product:* it is responsible to sell the product and all the tasks impacted with the sale of that product.
F11	*Product Visualization:* it works such as a showcase with all products available to sell.
F12	*Buy Product:* it is responsible for all functionalities involved to buy products to replenish stock.
F13	*Product Search:* it is used to search different products in the system store.
F14	*Insert suggestion:* it is used to the customer or user insert any suggestion to be futher considered by the company.
F15	*Reply suggestion:* it is used to send a reply to the customer or user which make a suggestion.
F16	*Remove suggestion:* once the suggestion is aswered it can be removed from the system.
F17	*Shopping Cart:* it allows the customer to group different products to make the purchase.
F18	*Remove Product from Shopping Cart:* it removes the selected product from the shopping cart.
F19	*Insert Product from Shopping Cart:* it adds the selected product to the shopping cart.
F20	*Buy:* once the products were inserted in the shopping cart, the users can now proceed to buy all of them.
F21	*Payment in cash:* it enables the payment in cash.
F22	*Debit:* it enables the deposit payment.
F23	*Credit:* it enables the credit card payment.
F24	*Receipt:* it enables the generation of customer receipts, for different event buys.
F25	*Discount* it allows the store to generate different discounts.
F26	*Tracking* it allows the customer to keep track of all products bought in the store.
F27	*Insert FAQ:* it aims to add different frequent asked questions in the system.
F28	*Remove FAQ:* it aims to remove frequent asked questions from the system.
F29	*Update FAQ:* it aims to update different frequent asked questions attributes.
F30	*Search FAQ:* it aims to search different frequent asked questions.
F31	*List FAQ:* it list all frequent asked questions already stored in the system.
F32	*Insert Bugtrack:* it aims to add different bug reports in the system.
F33	*Remove Bugtrack:* it aims to remove bug reports from the system.
F34	*Update Bugtrack:* it aims to update different bug reports attributes.
F35	*Search Bugtrack:* it aims to search different bug reports.
F36	*Insert Category:* it is used to group the store products in different categories.
F37	*Remove Category:* it enables the manager to remove product categories.
F38	*Update Category:* it enables the manager to update product categories attributes.
F39	*Search Category:* it enables the customers to search different product categories.
F40	*Contact us:* it works as a communication channel between costumer and system store.

Tabela 35 – Data set of RiSE Store SPL vs. Security Techniques vs. Conditional Compilation.

RiSE Store SPL	№ of Classes	№ of Interfaces	№ of Methods	№ of Static Methods	№ of Static Attributes	№ of Attributes	№ of Parameters(Value)
Base	74	1	437	6	72	219	318
T01	81	2	475	16	73	224	363
T02	81	2	465	9	75	224	337
T03	76	1	436	8	72	218	325
T04	96	3	560	19	80	250	438
T05	96	3	541	20	81	245	420
T06	93	2	531	17	79	244	408
T07	94	2	535	18	79	247	415
T08	110	4	646	22	84	258	506
T09	127	6	774	24	88	274	598
T10	161	10	968	34	103	299	731
T11	111	4	646	24	85	258	508
T12	113	4	657	22	84	259	508
T13	115	5	683	25	85	267	545
T14	129	6	816	47	99	286	711
T15	76	1	479	33	83	231	438
T01- BASE	7	1	38	10	1	5	45
T02- BASE	7	1	28	3	3	5	19
T03- BASE	2	0	-1	2	0	-1	7
T04- BASE	22	2	123	13	8	31	120
T05- BASE	22	2	104	14	9	26	102
T06- BASE	19	1	94	11	7	25	90
T07- BASE	20	1	98	12	7	28	97
T08- BASE	36	3	209	16	12	39	188
T09- BASE	53	5	337	18	16	55	280
T10- BASE	87	9	531	28	31	80	413
T11- BASE	37	3	209	18	13	39	190
T12- BASE	39	3	220	16	12	40	190
T13- BASE	41	4	246	19	13	48	227
T14- BASE	55	5	379	41	27	67	393
T15- BASE	2	0	42	27	11	12	120

Tabela 36 – Data set of RiSE Store SPL vs. Security Techniques vs. AspectJ.

RiSE Store SPL	№ of Classes	№ of Interfaces	№ of Methods	№ of Static Methods	№ of Static Attributes	№ of Attributes	№ of Parameters(Value)
Base	74	1	437	6	72	219	318
T01	83	2	497	16	73	225	383
T02	83	2	487	9	75	225	357
T03	77	1	439	8	72	219	331
T04	99	3	565	18	80	251	441
T05	99	3	546	20	81	246	424
T06	95	2	534	17	79	245	410
T07	96	2	538	18	79	247	417
T08	114	4	663	21	84	259	520
T09	131	6	791	23	88	275	612
T10	165	10	985	33	103	300	734
T11	115	4	663	23	85	259	522
T12	117	4	674	21	84	260	522
T13	119	5	702	24	85	268	561
T14	133	6	833	46	99	287	723
T15	77	1	495	33	83	231	455
T01- BASE	9	1	60	10	1	6	65
T02- BASE	9	1	50	3	3	6	39
T03- BASE	3	0	2	2	0	0	13
T04- BASE	25	2	128	12	8	32	123
T05- BASE	25	2	109	14	9	27	106
T06- BASE	21	1	97	11	7	26	92
T07- BASE	22	1	101	12	7	28	99
T08- BASE	40	3	226	15	12	40	202
T09- BASE	57	5	354	17	16	56	294
T10- BASE	91	9	548	27	31	81	416
T11- BASE	41	3	226	17	13	40	204
T12- BASE	43	3	237	15	12	41	204
T13- BASE	45	4	265	18	13	49	243
T14- BASE	59	5	396	40	27	68	405
T15- BASE	3	0	58	27	11	12	137

Tabela 37 – Functional Properties.

ID	Description
F1	*Documentation:* the lawyers during its activity can search for laws and documents to attach or facilitate the writting process.
F2	*Insert Customer:* it aims to insert different customers in the system.
F3	*Remove Customer:* it aims to remove customers from the system.
F4	*Update Customer:* it aims to update different customers attributes.
F5	*Search Customer:* it search for any customer already stored in the system.
F6	*Activate/Deactivate Customer:* it enables the user to activate/deactivate customers, without losing any data.
F7	*Insert Enterprise:* it aims to add different companies in the system.
F8	*Remove Enterprise:* it aims to remove companies from the system.
F9	*Update Enterprise:* it aims to update different companies attributes.
F10	*Activate/Deactivate Enterprise:* it enables the user to activate/deactivate companies, without losing any data.
F11	*Insert Lawyer:* it it aims to insert different lawyers in the system.
F12	*Remove Lawyer:* it aims to remove lawyers from the system.
F13	*Update Lawyer:* it aims to update different lawyers attributes.
F14	*Lawyer assignment:* it is responsible to assign a lawyer to a process/task.
F15	*Activate/Deactivate Lawyer:* it enables the user to activate/deactivate lawyers, without losing any data.
F16	*Report:* The user can extract different kind of information by generating reports.
F17	*Insert Trainee:* it aims to add different trainees in the system and assign it to a lawyer tutor.
F18	*Remove Trainee:* it aims to remove trainees from the system.
F19	*Update Trainee:* it aims to update different trainees attributes.
F20	*Activate/Deactivate Trainee:* it enables the user to activate/deactivate trainees, without losing any data.
F21	*Insert Process:* it aims to add different process in the system.
F22	*Remove Process:* it aims to remove process from the system.
F23	*Update Process:* it aims to update different process attributes.
F24	*Process assignment:* it is responsible to assign a process to a lawyer and customer.
F25	*Schedule* it allows the lawyers to build their on agenda/schedule.
F26	*Input Finantial* it enabels the departament manager to manage input finantials.
F27	*Output Finantial:* it enabels the departament manager to manage input finantials.
F28	*Insert Departament:* it aims to insert a new departament/office to be managed by the system.
F29	*Send Email:* it enables the lawyers to send emails to trainees or customers.
F30	*Laws:* it generates different reports regarding to law types.
F31	*Cash desk:* it calculates the finantial based on Input/Output finantials.
F32	*Server Configuration:* it enables the office to setup its server in different locations.
F33	*Backup:* it aims to backup all data stored in a specific office.
F34	*Restore:* it aims to restore all data stored in a specific office.

Tabela 38 – Data set of Law Office SPL vs. Security Techniques vs. Conditional Compilation.

Law Office SPL	№ of Classes	№ of Interfaces	№ of Methods	№ of Static Methods	№ of Static Attributes	№ of Attributes	№ of Parameters(Value)
Base	116	2	724	10	100	643	595
T01	123	3	763	20	101	649	645
T02	123	3	753	13	103	649	619
T03	118	2	722	12	100	643	601
T04	138	4	846	22	108	673	714
T05	138	4	827	24	109	668	697
T06	135	3	817	21	107	667	685
T07	136	3	821	22	107	673	692
T08	152	5	938	25	112	682	783
T09	169	7	1066	27	116	698	875
T10	203	11	1260	37	131	725	1008
T11	153	5	938	27	113	682	785
T12	155	5	949	25	112	683	785
T13	157	6	975	28	113	691	822
T14	171	7	1108	50	127	710	987
T15	118	2	766	37	111	655	715
T01- BASE	7	1	39	10	1	6	50
T02- BASE	7	1	29	3	3	6	24
T03- BASE	2	0	-2	2	0	0	6
T04- BASE	22	2	122	12	8	30	119
T05- BASE	22	2	103	14	9	25	102
T06- BASE	19	1	93	11	7	24	90
T07- BASE	20	1	97	12	7	30	97
T08- BASE	36	3	214	15	12	39	188
T09- BASE	53	5	342	17	16	55	280
T10- BASE	87	9	536	27	31	82	413
T11- BASE	37	3	214	17	13	39	190
T12- BASE	39	3	225	15	12	40	190
T13- BASE	41	4	251	18	13	48	227
T14- BASE	55	5	384	40	27	67	392
T15- BASE	2	0	42	27	11	12	120

Tabela 39 – Data set of Law Office SPL vs. Security Techniques vs. AspectJ.

Law Office SPL	№ of Classes	№ of Interfaces	№ of Methods	№ of Static Methods	№ of Static Attributes	№ of Attributes	№ of Parameters(Value)
Base	116	2	724	10	100	643	595
T01	125	3	804	20	101	649	670
T02	125	3	794	13	103	649	644
T03	119	2	727	12	100	643	608
T04	141	4	851	22	108	673	720
T05	141	4	831	24	109	668	702
T06	137	3	820	21	107	667	689
T07	138	3	824	22	107	673	695
T08	156	5	945	25	112	682	792
T09	173	7	897	27	116	697	747
T10	207	11	1271	37	131	725	1025
T11	157	5	945	27	113	682	794
T12	159	5	956	25	112	683	794
T13	161	6	984	28	113	690	833
T14	175	7	1115	50	127	710	996
T15	119	2	799	37	111	655	922
T01- BASE	9	1	80	10	1	6	75
T02- BASE	9	1	70	3	3	6	49
T03- BASE	3	0	3	2	0	0	13
T04- BASE	25	2	127	12	8	30	125
T05- BASE	25	2	107	14	9	25	107
T06- BASE	21	1	96	11	7	24	94
T07- BASE	22	1	100	12	7	30	100
T08- BASE	40	3	221	15	12	39	197
T09- BASE	57	5	173	17	16	54	152
T10- BASE	91	9	547	27	31	82	430
T11- BASE	41	3	221	17	13	39	199
T12- BASE	43	3	232	15	12	40	199
T13- BASE	45	4	260	18	13	47	238
T14- BASE	59	5	391	40	27	67	401
T15- BASE	3	0	75	27	11	12	327

8 DATA COLLECTION

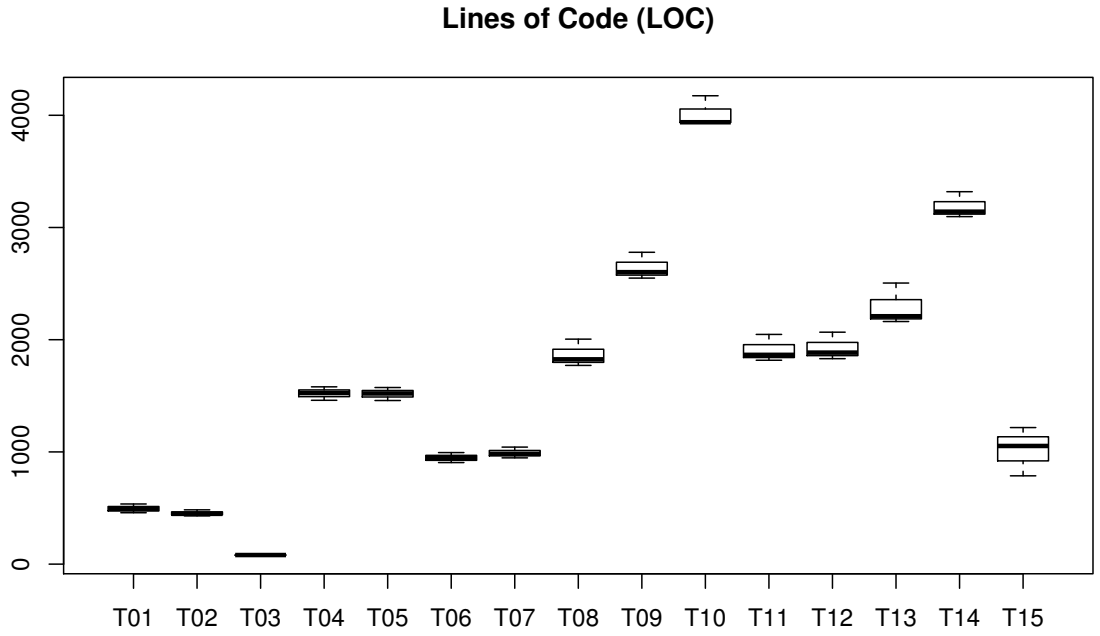

Figura 34 – Lines of code (LOC) for CC.

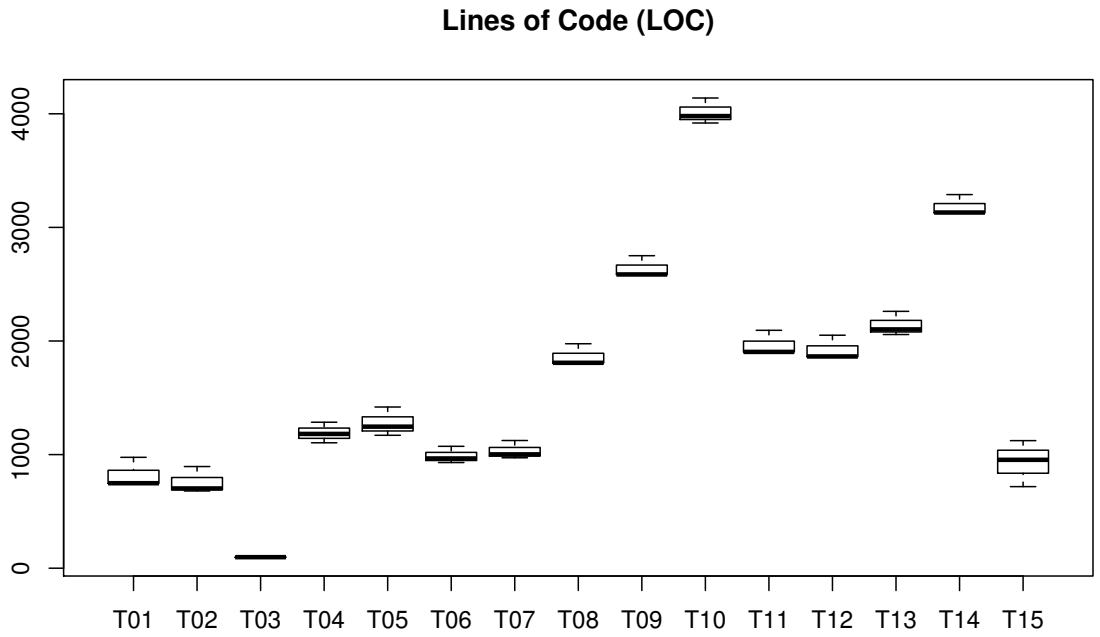

Figura 35 – Lines of code (LOC) for AOP.

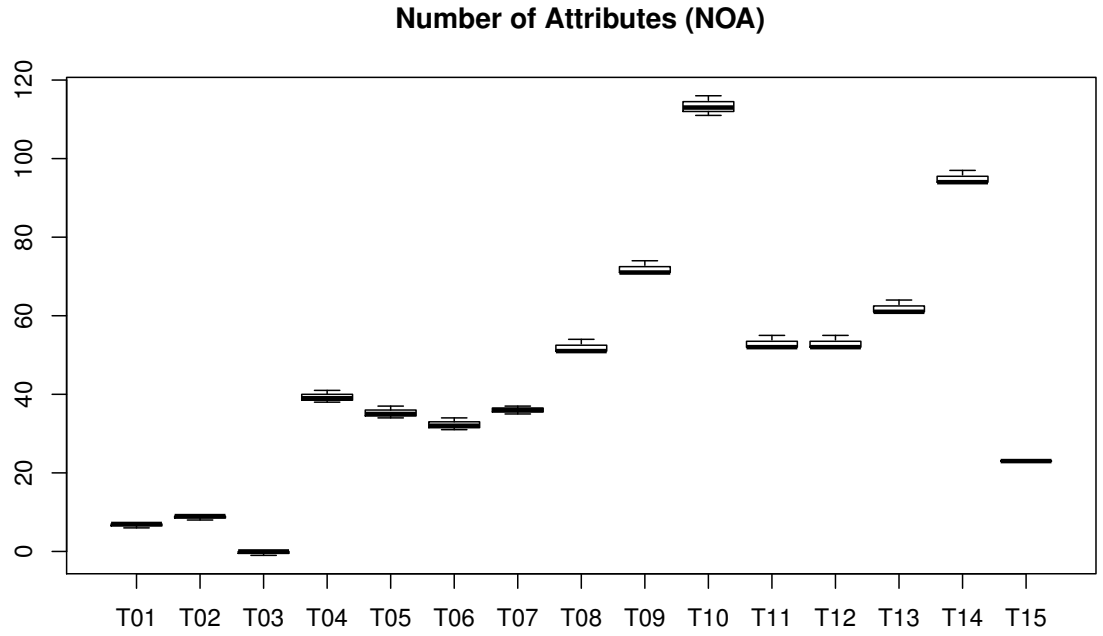

Figura 36 – Number of attributes (NOA) for CC.

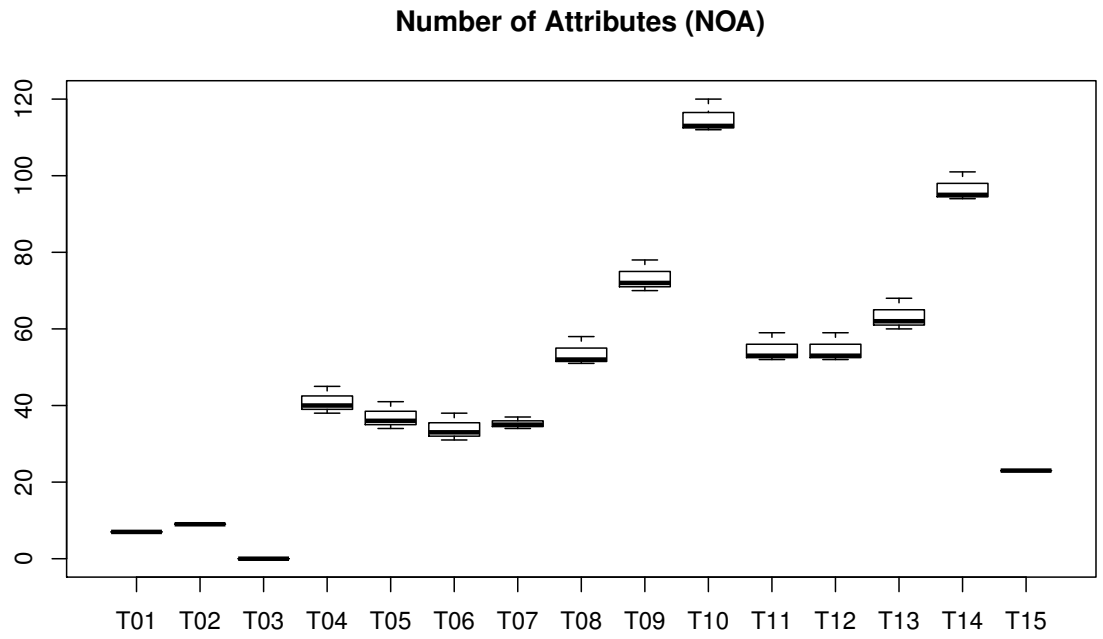

Figura 37 – Number of attributes (NOA) for AOP.

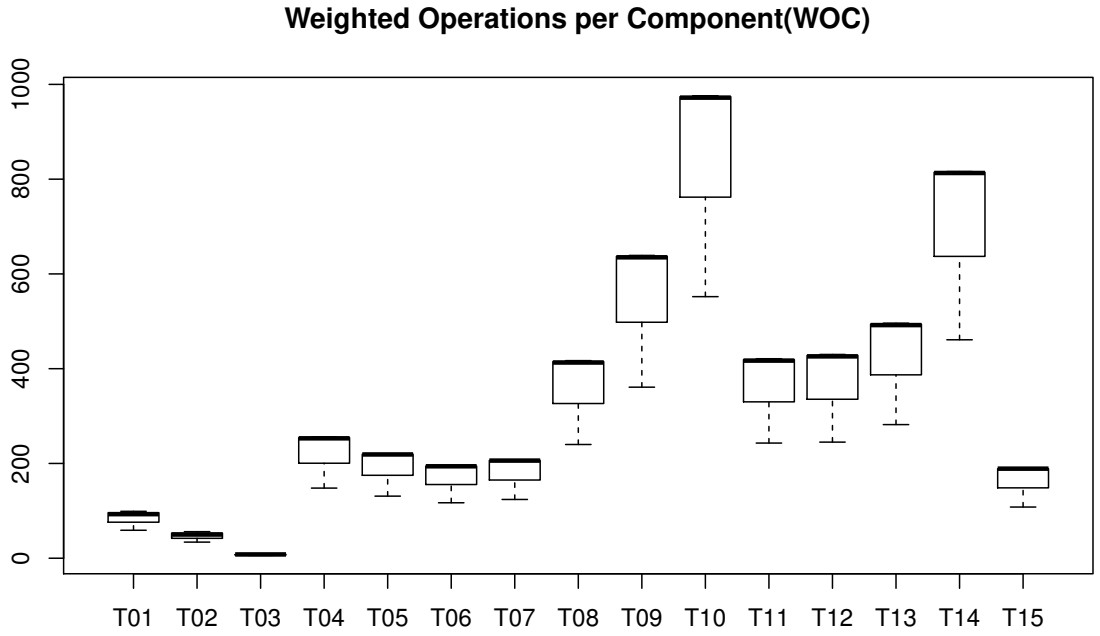

Figura 38 – Weighted Operations per Compoenent (WOC) for CC.

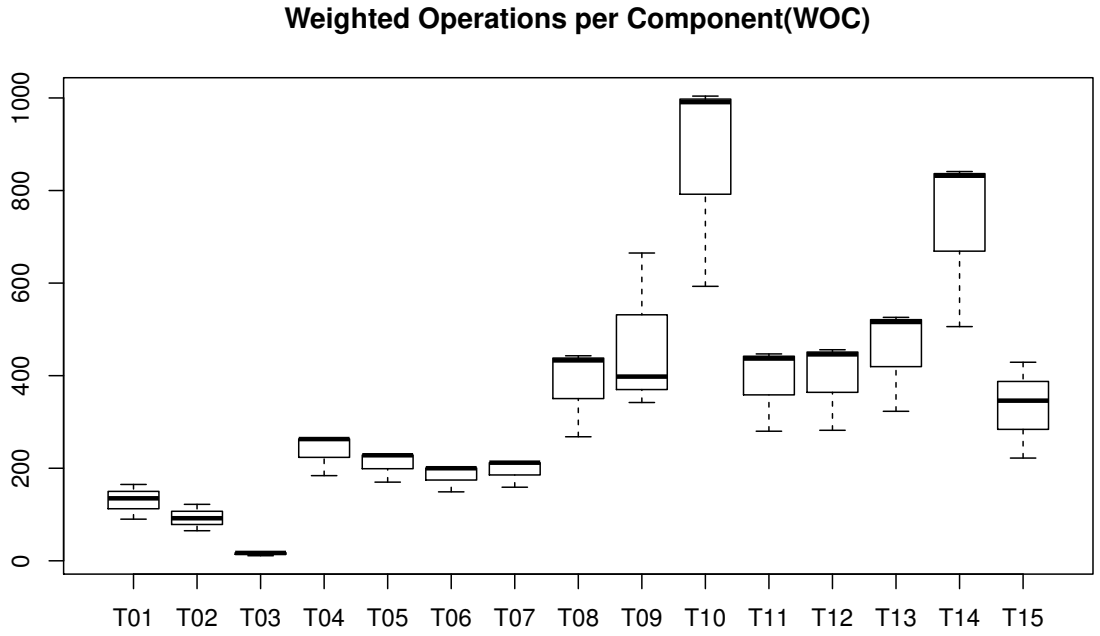

Figura 39 – Weighted Operations per Compoenent(WOC) for AOP.

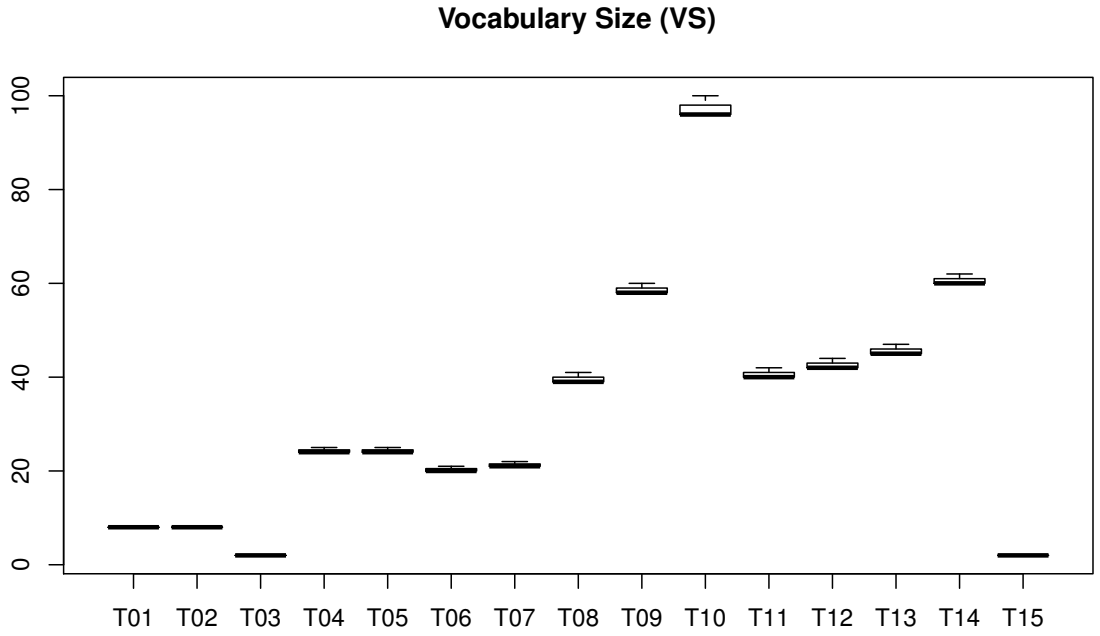

Figura 40 – Vocabulary Size (VS) for CC.

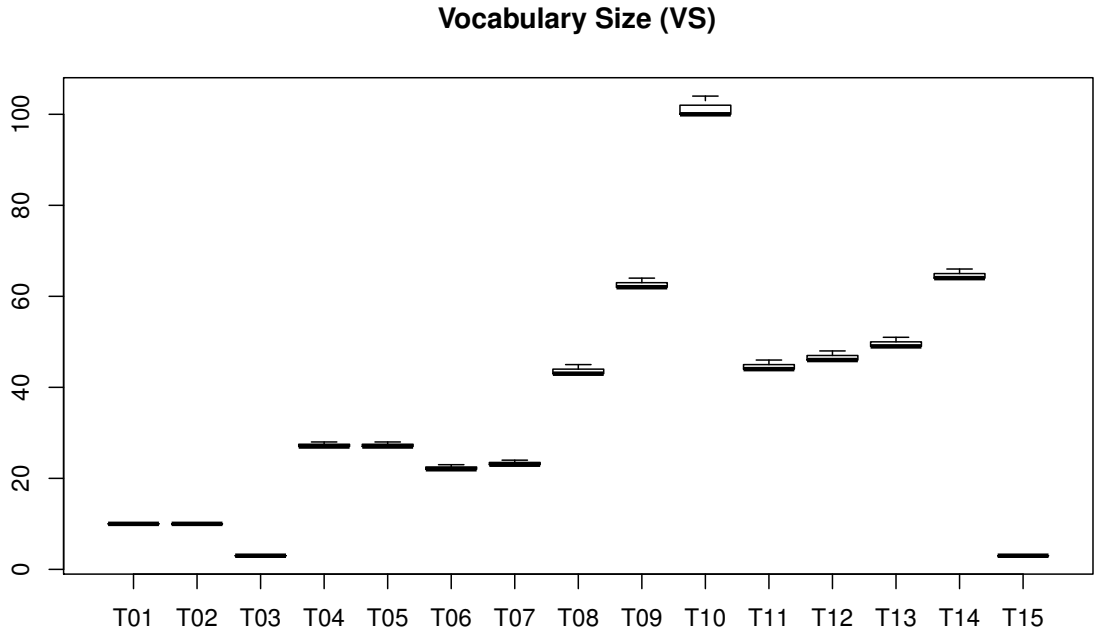

Figura 41 – Vocabulary Size (VS) for AOP.

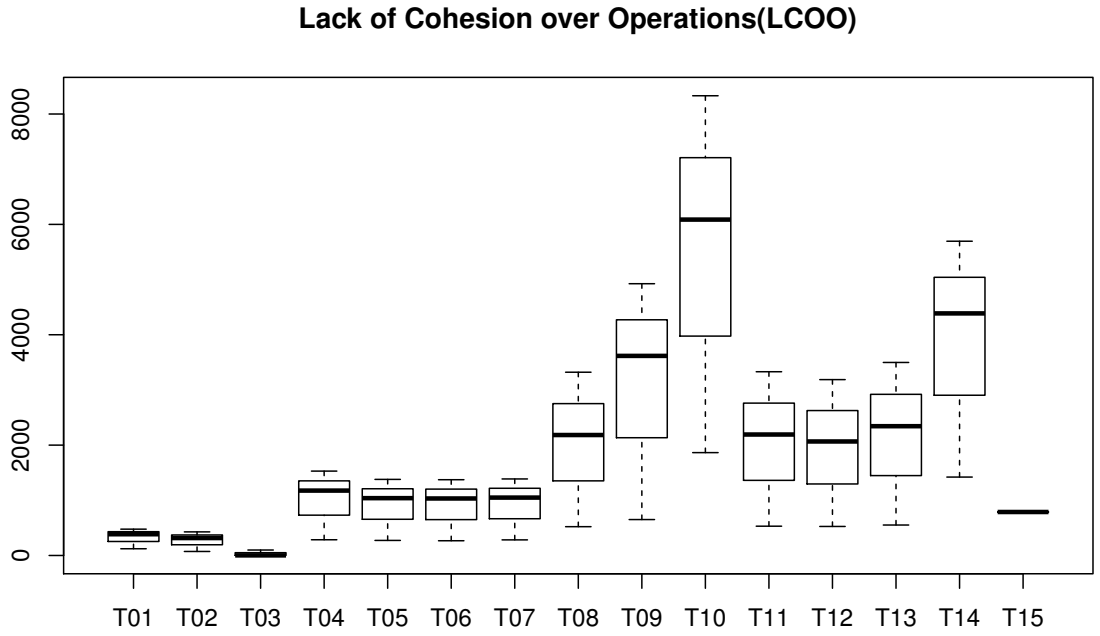

Figura 42 – Lack of Cohesion over Operations(LCOO) for CC.

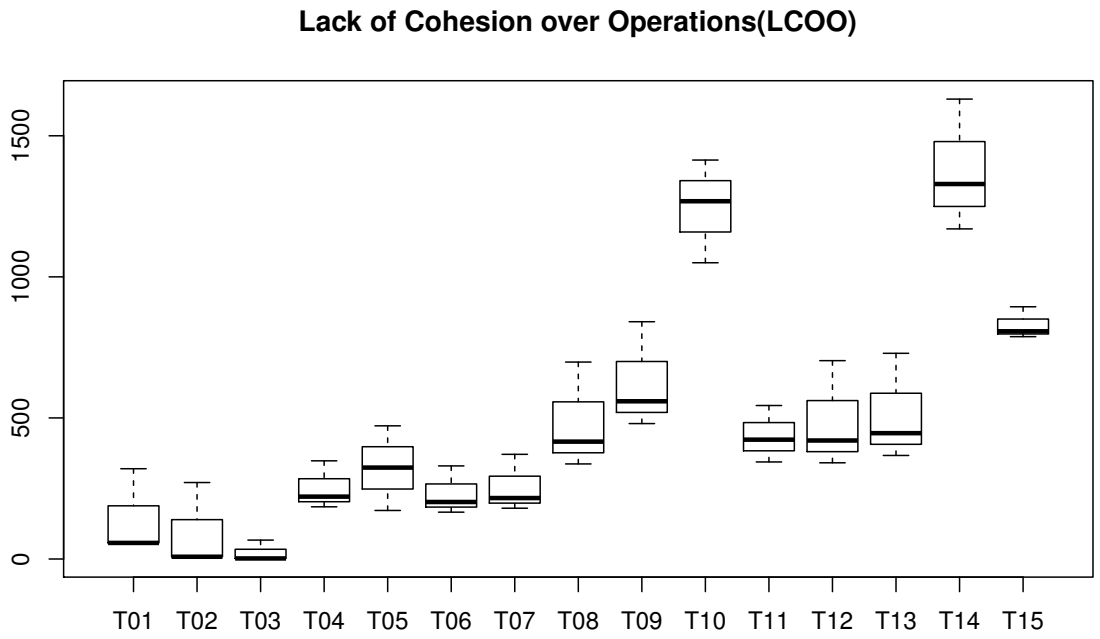

Figura 43 – Lack of Cohesion over Operations(LCOO) for AOP.

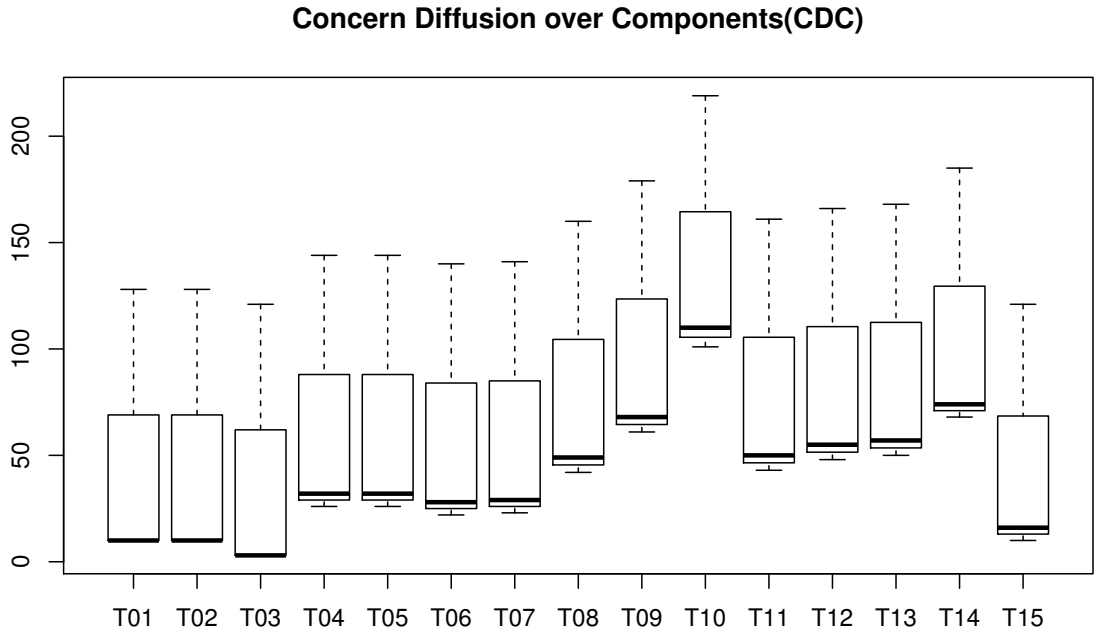

Figura 44 – Concern Diffusion over Components(CDC) for CC.

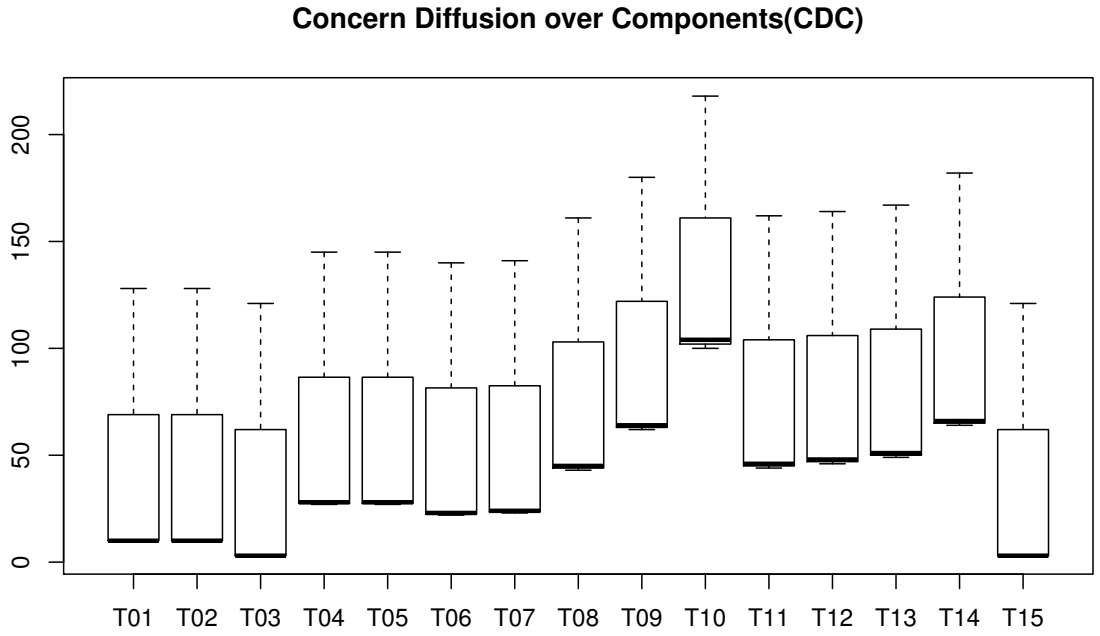

Figura 45 – Concern Diffusion over Components(CDC) for AOP.

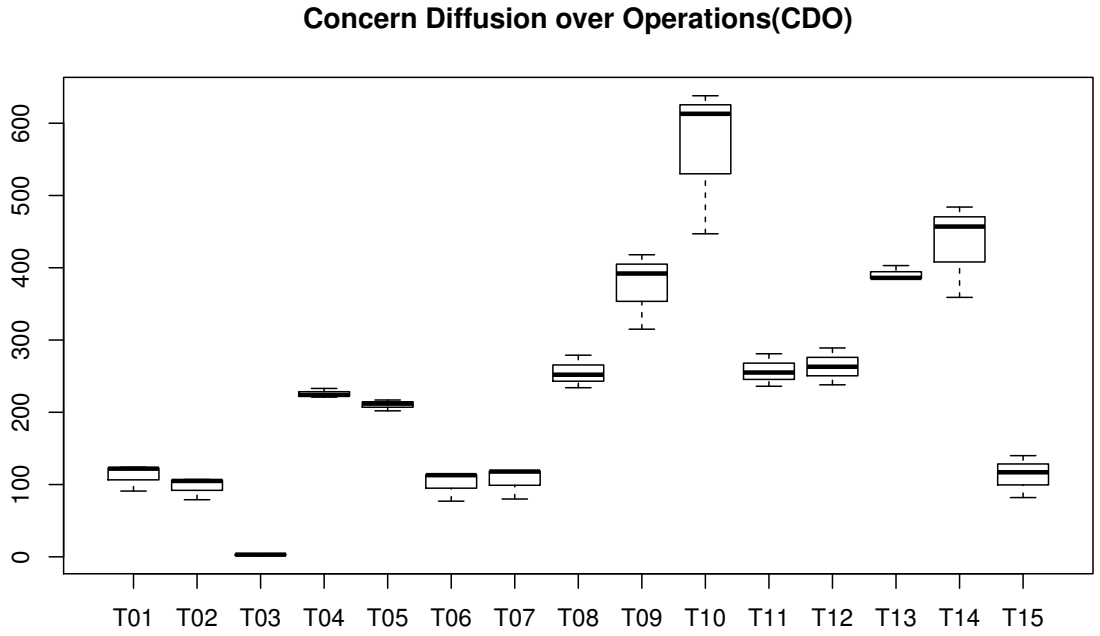

Figura 46 – Concern Diffusion over Operations(CDO) for CC.

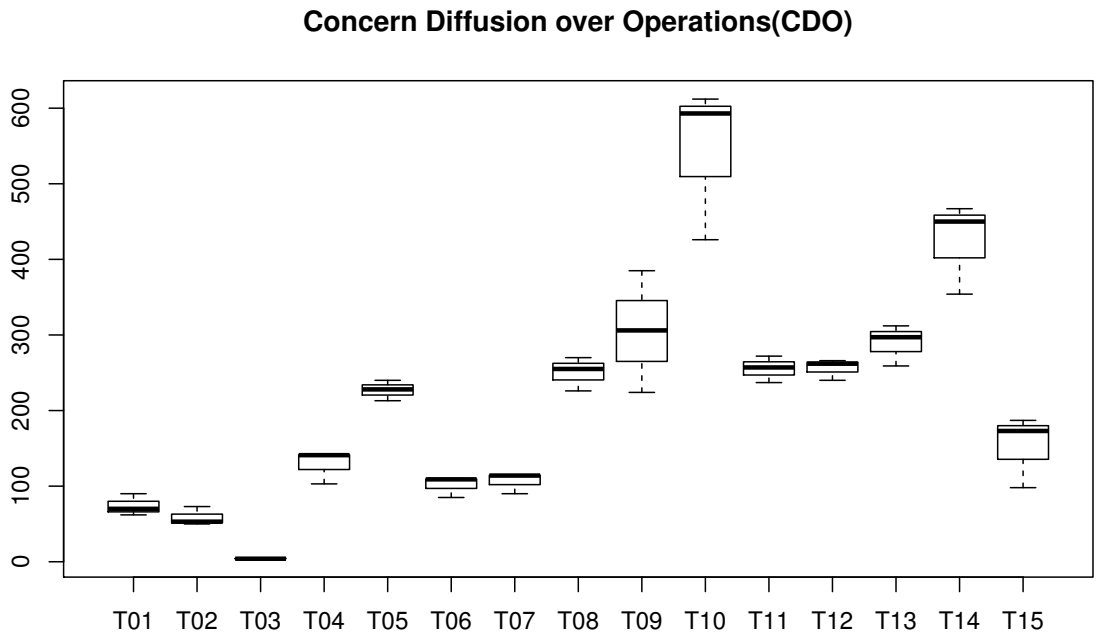

Figura 47 – Concern Diffusion over Operations(CDO) for AOP.

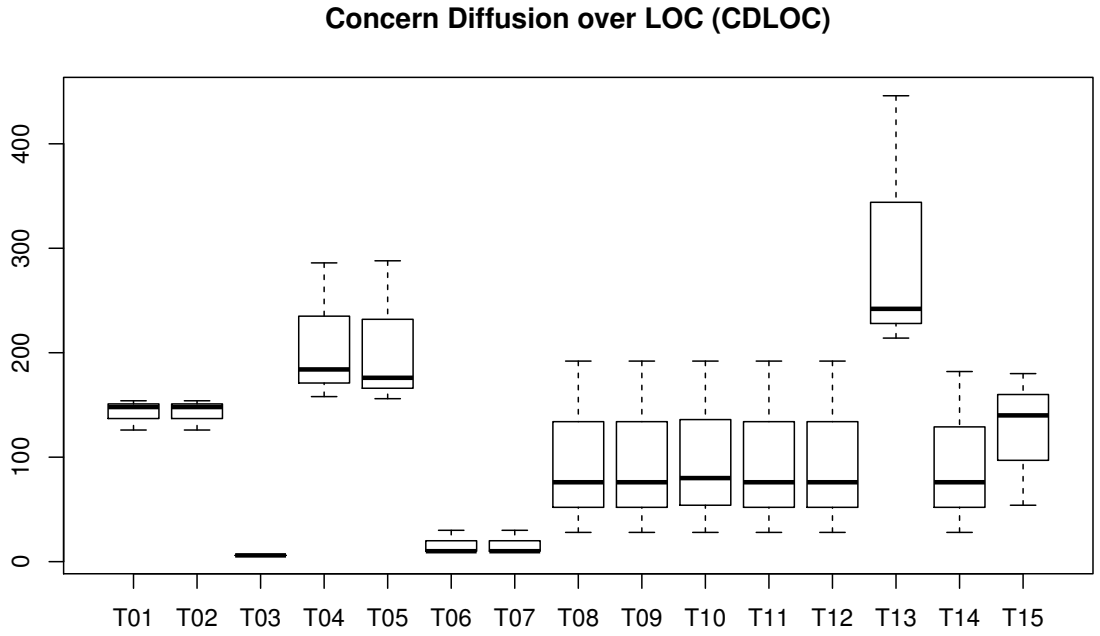

Figura 48 – Concern Diffusion over LOC (CDLOC) for CC.

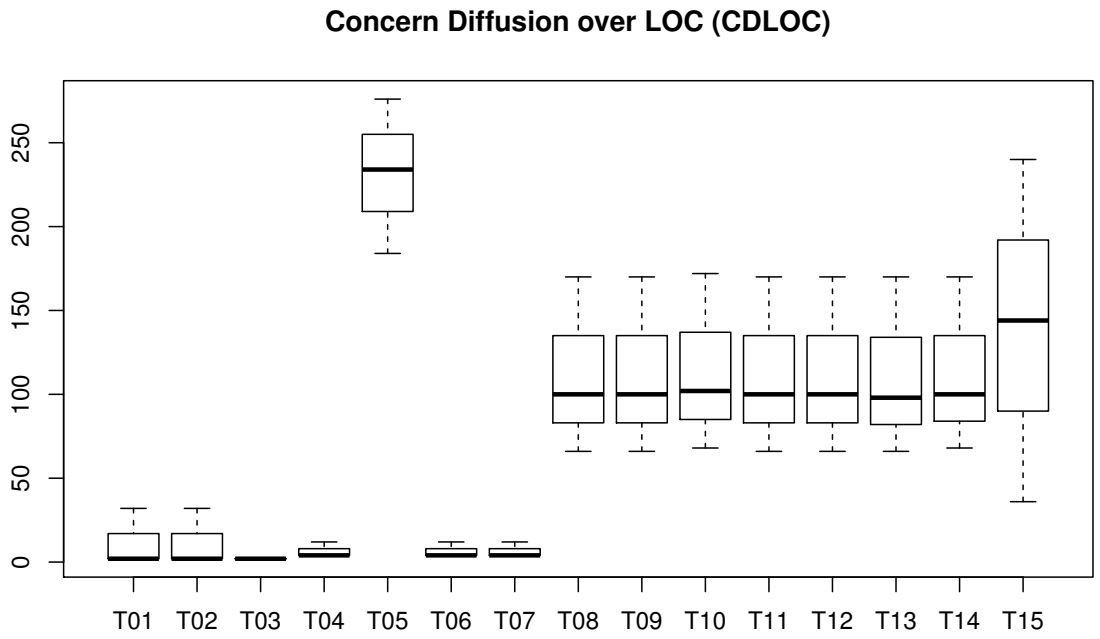

Figura 49 – Concern Diffusion over LOC (CDLOC) for AOP.

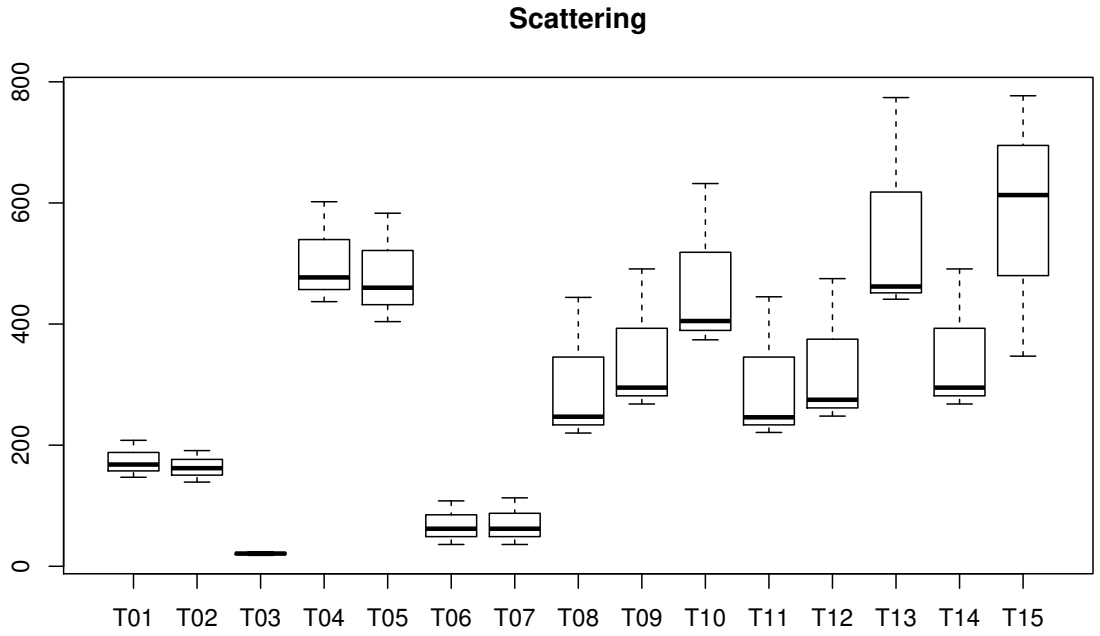

Figura 50 – Scattering for CC.

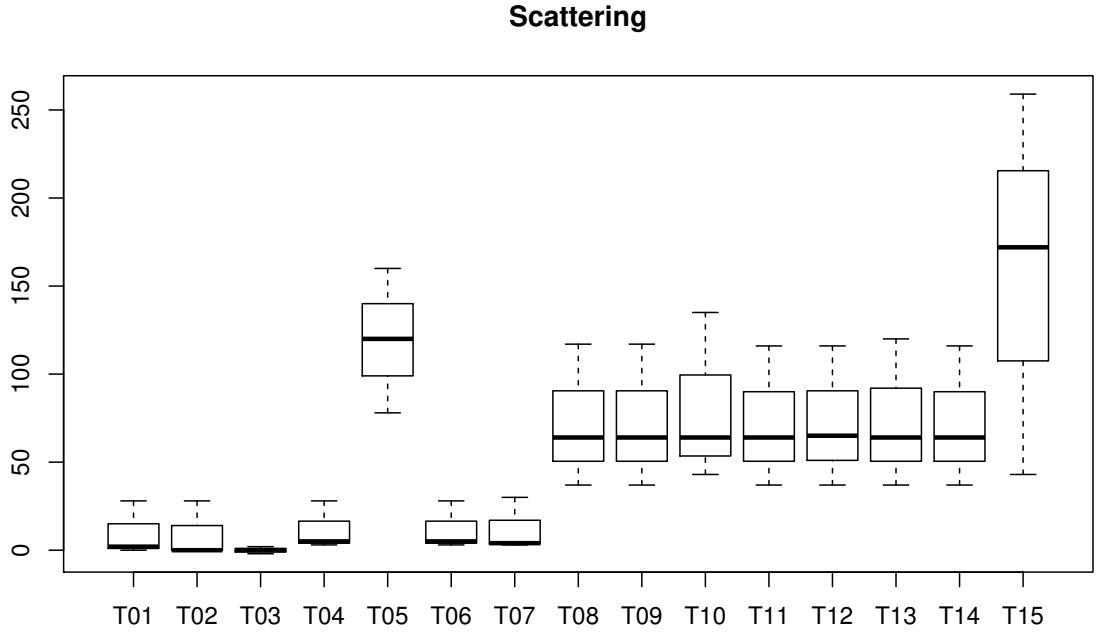

Figura 51 – Scattering for AOP.

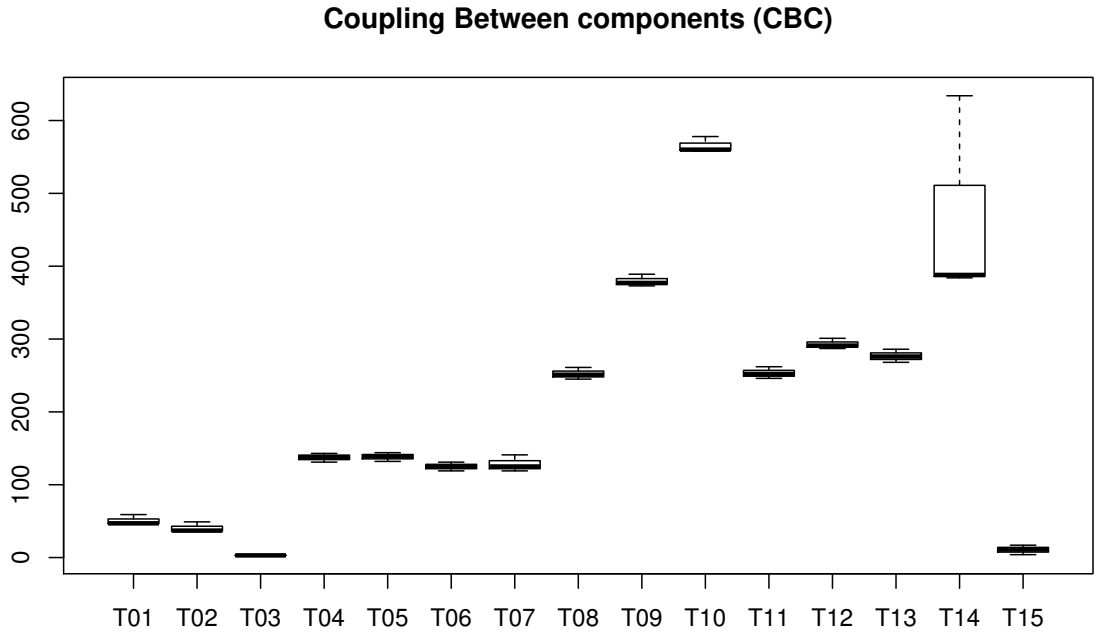

Figura 52 – Coupling Between components(CBC) for CC.

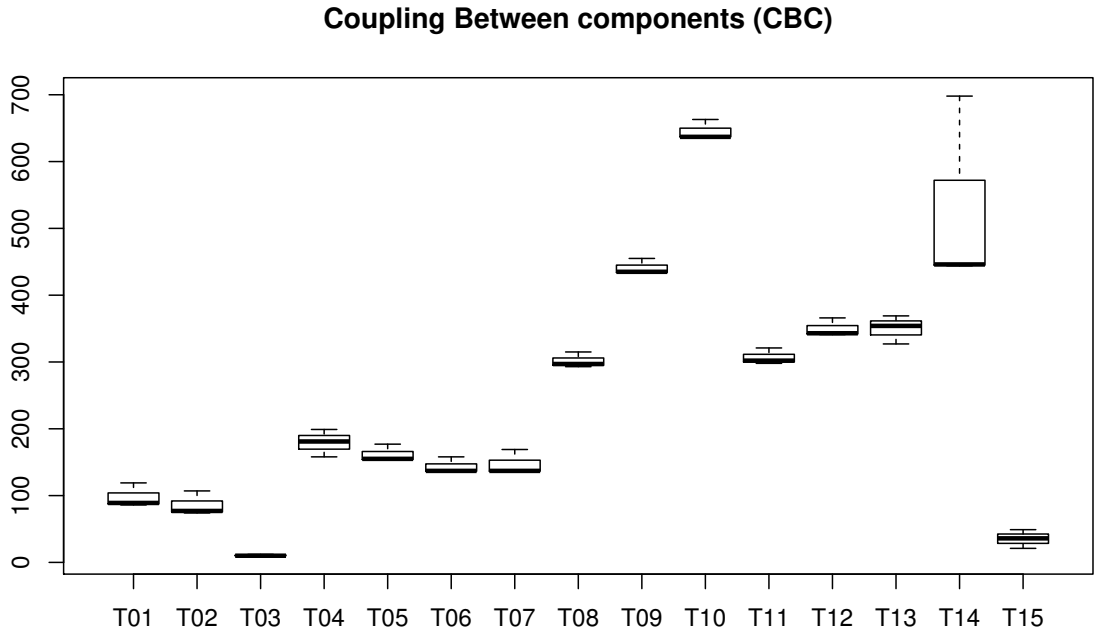

Figura 53 – Coupling Between components(CBC) for AOP.

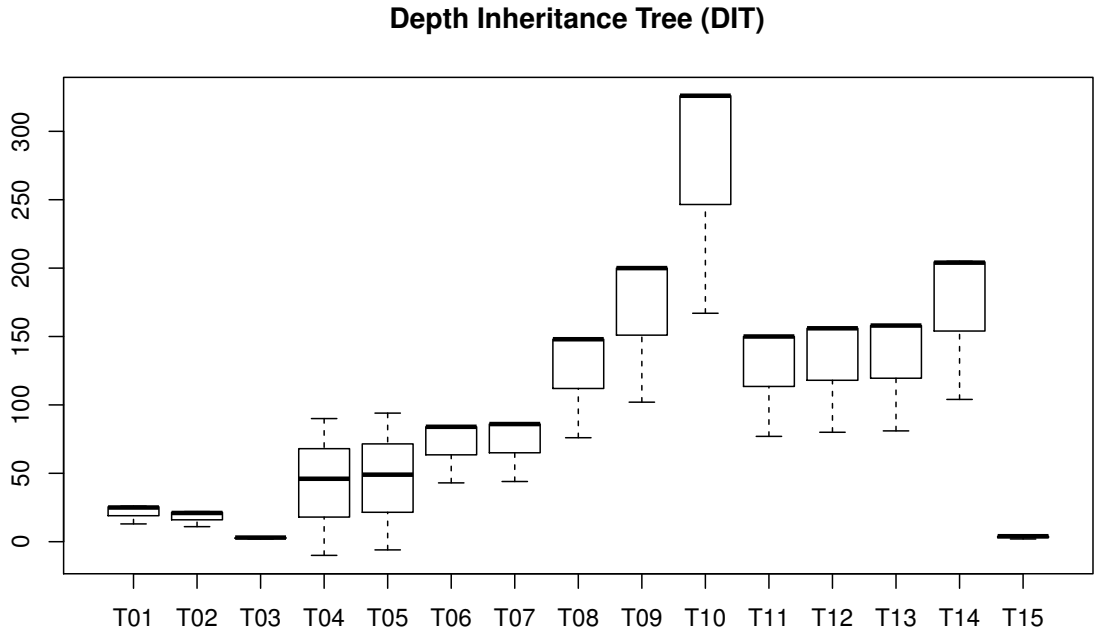

Figura 54 – Depth Inheritance Tree (DIT) for CC.

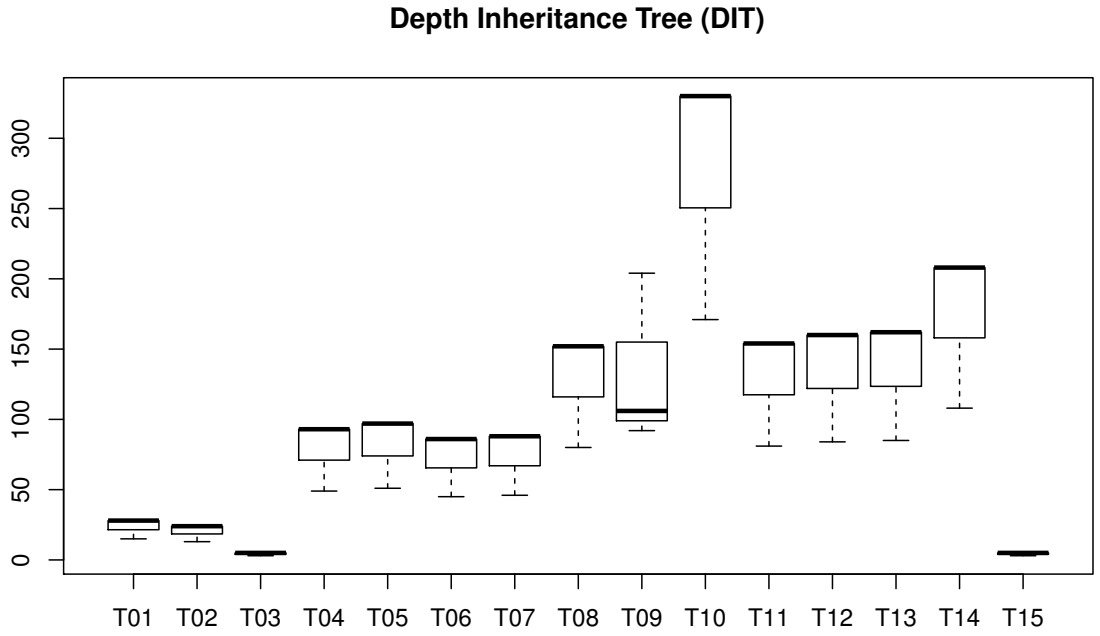

Figura 55 – Depth Inheritance Tree (DIT) for AOP.

Tabela 40 – Metrics for RiSE Event SPL

RiSE Event SPL	LOC	NOA	WOC	VS	LCOM	CDC	CDO	CDLOC	CBC	DIT	Scattering
Base	26395	936	2892	516	25335					969	
Conditional Compilation											
T01	26886	943	2951	524	25812	0	0	0	0	982	0
T02	26843	945	2926	524	25763	0	0	0	0	980	0
T03	26473	936	2900	518	25337	0	0	0	0	971	0
T04	27855	977	3040	541	26864	0	0	0	0	1015	0
T05	27853	973	3023	541	26714	0	0	0	0	1018	0
T06	27389	970	3009	537	26708	0	0	0	0	1012	0
T07	27438	972	3016	538	26722	0	0	0	0	1013	0
T08	28400	990	3132	557	28656	0	0	0	0	1045	0
T09	29174	1010	3253	576	30260	0	0	0	0	1071	0
T10	30569	1052	3444	616	33666	0	0	0	0	1136	0
T11	28442	991	3135	558	28665	0	0	0	0	1046	0
T12	28462	991	3137	560	28522	0	0	0	0	1049	0
T13	28900	1000	3174	563	28833	0	0	0	0	1050	0
T14	29714	1033	3353	578	31030	0	0	0	0	1073	0
T15	27448	959	3000	518	26123	0	0	0	0	971	0
T01- BASE	491	7	59	8	477	10	91	126	59	13	147
T02- BASE	448	9	34	8	428	10	79	126	49	11	139
T03- BASE	78	0	8	2	2	3	4	6	3	2	18
T04- BASE	1460	41	148	25	1529	32	224	286	143	46	437
T05- BASE	1458	37	131	25	1379	32	217	288	144	49	404
T06- BASE	994	34	117	21	1373	28	77	30	131	43	108
T07- BASE	1043	36	124	22	1387	29	80	30	141	44	113
T08- BASE	2005	54	240	41	3321	49	234	192	261	76	444
T09- BASE	2779	74	361	60	4925	68	315	192	389	102	491
T10- BASE	4174	116	552	100	8331	110	447	192	578	167	632
T11- BASE	2047	55	243	42	3330	50	236	192	262	77	445
T12- BASE	2067	55	245	44	3187	55	238	192	301	80	475
T13- BASE	2505	64	282	47	3498	57	385	446	286	81	774
T14- BASE	3319	97	461	62	5695	74	359	182	634	104	491
T15- BASE	1053	23	108	2	788	16	117	140	17	2	613
AspectJ											
T01	27140	943	2982	526	25655	0	0	0	0	984	0
T02	27097	945	2957	526	25606	0	0	0	0	982	0
T03	26488	936	2903	519	25336	0	0	0	0	972	0
T04	27680	981	3076	544	25683	0	0	0	0	1018	0
T05	27814	977	3062	544	25807	0	0	0	0	1020	0
T06	27468	974	3041	539	25665	0	0	0	0	1014	0
T07	27519	970	3051	540	25706	0	0	0	0	1015	0
T08	28371	994	3160	561	26033	0	0	0	0	1049	0
T09	29146	1014	3290	580	26176	0	0	0	0	1075	0
T10	30534	1056	3485	620	26749	0	0	0	0	1140	0
T11	28489	995	3172	562	25879	0	0	0	0	1050	0
T12	28446	995	3174	564	26038	0	0	0	0	1053	0
T13	28656	1004	3215	567	26064	0	0	0	0	1054	0
T14	29684	1037	3398	582	26965	0	0	0	0	1077	0
T15	27518	959	3238	519	26229	0	0	0	0	972	0
T01- BASE	745	7	90	10	320	10	62	32	119	15	28
T02- BASE	702	9	65	10	271	10	50	32	107	13	28
T03- BASE	93	0	11	3	1	3	4	2	10	3	-2
T04- BASE	1285	45	184	28	348	28	103	12	199	49	28
T05- BASE	1419	41	170	28	472	28	228	276	177	51	160
T06- BASE	1073	38	149	23	330	23	85	12	158	45	28
T07- BASE	1124	34	159	24	371	24	90	12	169	46	30
T08- BASE	1976	58	268	45	698	45	226	170	315	80	117
T09- BASE	2751	78	398	64	841	64	306	170	455	106	117
T10- BASE	4139	120	593	104	1414	104	426	172	663	171	135
T11- BASE	2094	59	280	46	544	46	237	170	321	81	116
T12- BASE	2051	59	282	48	703	48	240	170	366	84	116
T13- BASE	2261	68	323	51	729	51	259	170	369	85	120
T14- BASE	3289	101	506	66	1630	66	354	170	698	108	116
T15- BASE	1123	23	346	3	894	3	187	144	49	3	259

Tabela 41 – Difference between AOP and CC considering RiSE Event SPL.

CC vs AOP	LOC	NOA	WOC	VS	LCOO	CDC	CDO	CDLOC	CBC	DIT	Scattering
T01	-0.94	0	-1.04	-0.38	0.61	0	46.77	293.75	-50.42	-0.2	425
T02	-0.94	0	-1.05	-0.38	0.61	0	58	293.75	-54.21	-0.2	396.43
T03	-0.06	0	-0.1	-0.19	0	0	0	200	-70	-0.1	800
T04	0.63	-0.41	-1.17	-0.55	4.6	14.29	117.48	2283.33	-28.14	-0.29	1460.71
T05	0.14	-0.41	-1.27	-0.55	3.51	14.29	-4.82	4.35	-18.64	-0.2	152.5
T06	-0.29	-0.41	-1.05	-0.37	4.06	21.74	-9.41	150	-17.09	-0.2	285.71
T07	-0.29	0.21	-1.15	-0.37	3.95	20.83	-11.11	150	-16.57	-0.2	276.67
T08	0.1	-0.4	-0.89	-0.71	10.08	8.89	3.54	12.94	-17.14	-0.38	279.49
T09	0.1	-0.39	-1.12	-0.69	15.6	6.25	2.94	12.94	-14.51	-0.37	319.66
T10	0.11	-0.38	-1.18	-0.65	25.86	5.77	4.93	11.63	-12.82	-0.35	368.15
T11	-0.16	-0.4	-1.17	-0.71	10.77	8.7	-0.42	12.94	-18.38	-0.38	283.62
T12	0.06	-0.4	-1.17	-0.71	9.54	14.58	-0.83	12.94	-17.76	-0.38	309.48
T13	0.85	-0.4	-1.28	-0.71	10.62	11.76	48.65	162.35	-22.49	-0.38	545
T14	0.1	-0.39	-1.32	-0.69	15.08	12.12	1.41	7.06	-9.17	-0.37	323.28
T15	-0.25	0	-7.35	-0.19	-0.4	433.33	-37.43	-2.78	-65.31	-0.1	136.68

Tabela 42 – Metrics for RiSE Store SPL vs. Security Techniques

RiSE Store SPL	LOC	NOA	WOC	VS	LCOM	CDC	CDO	CDLOC	CBC	DIT	Scattering
Base	5426	291	761	75	6822	0	0	0	0	214	0
Conditional Compilation											
T01	5884	297	854	83	7207	0	0	0	0	239	0
T02	5856	299	811	83	7138	0	0	0	0	235	0
T03	5510	290	769	77	6921	0	0	0	0	217	0
T04	6951	330	1017	99	7998	0	0	0	0	204	0
T05	6947	326	981	99	7861	0	0	0	0	208	0
T06	6371	323	956	95	7855	0	0	0	0	298	0
T07	6409	326	968	96	7871	0	0	0	0	300	0
T08	7251	342	1174	114	9005	0	0	0	0	362	0
T09	8028	362	1396	133	10439	0	0	0	0	414	0
T10	9359	402	1733	171	12909	0	0	0	0	540	0
T11	7291	343	1178	115	9014	0	0	0	0	364	0
T12	7311	343	1187	117	8888	0	0	0	0	370	0
T13	7635	352	1253	120	9165	0	0	0	0	372	0
T14	8567	385	1574	135	11209	0	0	0	0	419	0
T15	6213	314	950	77	7610	0	0	0	0	218	0
T01- BASE	458	6	93	8	385	10	124	154	47	25	168
T02- BASE	430	8	50	8	316	10	107	154	37	21	162
T03- BASE	84	-1	8	2	99	3	3	6	3	3	24
T04- BASE	1525	39	256	24	1176	26	221	158	138	-10	477
T05- BASE	1521	35	220	24	1039	26	202	156	139	-6	460
T06- BASE	945	32	195	20	1033	22	114	10	125	84	36
T07- BASE	983	35	207	21	1049	23	119	10	125	86	36
T08- BASE	1825	51	413	39	2183	42	252	28	251	148	247
T09- BASE	2602	71	635	58	3617	61	392	28	377	200	295
T10- BASE	3933	111	972	96	6087	101	613	28	560	326	374
T11- BASE	1865	52	417	40	2192	43	255	28	252	150	246
T12- BASE	1885	52	426	42	2066	48	263	28	291	156	275
T13- BASE	2209	61	492	45	2343	50	386	214	276	158	462
T14- BASE	3141	94	813	60	4387	68	457	28	388	205	295
T15- BASE	787	23	189	2	788	10	82	54	11	4	347
AspectJ											
T01	6174	298	896	85	6879	0	0	0	0	242	0
T02	6106	300	853	85	6830	0	0	0	0	238	0
T03	5525	291	778	78	6824	0	0	0	0	219	0
T04	6608	331	1024	102	7043	0	0	0	0	307	0
T05	6672	327	990	102	7146	0	0	0	0	311	0
T06	6392	324	961	97	7024	0	0	0	0	300	0
T07	6428	326	973	98	7038	0	0	0	0	302	0
T08	7234	343	1204	118	7159	0	0	0	0	366	0
T09	8012	363	1426	137	7302	0	0	0	0	418	0
T10	9345	403	1752	175	7872	0	0	0	0	544	0
T11	7330	344	1208	119	7166	0	0	0	0	368	0
T12	7290	344	1217	121	7163	0	0	0	0	374	0
T13	7528	353	1287	124	7189	0	0	0	0	376	0
T14	8551	386	1602	139	7992	0	0	0	0	422	0
T15	6144	314	983	78	7629	0	0	0	0	219	0
T01- BASE	748	7	135	10	57	10	70	2	86	28	0
T02- BASE	680	9	92	10	8	10	53	2	74	24	0
T03- BASE	99	0	17	3	2	3	4	2	12	5	0
T04- BASE	1182	40	263	27	221	27	142	4	181	93	5
T05- BASE	1246	36	229	27	324	27	240	234	155	97	120
T06- BASE	966	33	200	22	202	22	110	4	137	86	5
T07- BASE	1002	35	212	23	216	23	115	4	137	88	4
T08- BASE	1808	52	443	43	337	43	255	100	297	152	64
T09- BASE	2586	72	665	62	480	62	385	100	435	204	64
T10- BASE	3919	112	991	100	1050	100	593	102	637	330	64
T11- BASE	1904	53	447	44	344	44	257	100	302	154	64
T12- BASE	1864	53	456	46	341	46	266	100	343	160	65
T13- BASE	2102	62	526	49	367	49	297	98	354	162	64
T14- BASE	3125	95	841	64	1170	64	450	100	446	208	64
T15- BASE	718	23	222	3	807	3	98	36	36	5	43

Tabela 43 – Difference between AOP and CC considering RiSE Store SPL.

CC vs AOP	LOC	NOA	WOC	VS	LCOO	CDC	CDO	CDLOC	CBC	DIT	Scattering
T01	-4.7	-0.34	-4.69	-2.35	4.77	0	77.14	7600	-45.35	-1.24	-
T02	-4.09	-0.33	-4.92	-2.35	4.51	0	101.89	7600	-50	-1.26	-
T03	-0.27	-0.34	-1.16	-1.28	1.42	0	-25	200	-75	-0.91	0
T04	5.19	-0.3	-0.68	-2.94	13.56	-3.7	55.63	3850	-23.76	-33.55	9440
T05	4.12	-0.31	-0.91	-2.94	10.01	-3.7	-15.83	-33.33	-10.32	-33.12	283.33
T06	-0.33	-0.31	-0.52	-2.06	11.83	0	3.64	150	-8.76	-0.67	620
T07	-0.3	0	-0.51	-2.04	11.84	0	3.48	150	-8.76	-0.66	800
T08	0.24	-0.29	-2.49	-3.39	25.79	-2.33	-1.18	-72	-15.49	-1.09	285.94
T09	0.2	-0.28	-2.1	-2.92	42.96	-1.61	1.82	-72	-13.33	-0.96	360.94
T10	0.15	-0.25	-1.08	-2.29	63.99	1	3.37	-72.55	-12.09	-0.74	484.38
T11	-0.53	-0.29	-2.48	-3.36	25.79	-2.27	-0.78	-72	-16.56	-1.09	284.38
T12	0.29	-0.29	-2.47	-3.31	24.08	4.35	-1.13	-72	-15.16	-1.07	323.08
T13	1.42	-0.28	-2.64	-3.23	27.49	2.04	29.97	118.37	-22.03	-1.06	621.88
T14	0.19	-0.26	-1.75	-2.88	40.25	6.25	1.56	-72	-13	-0.71	360.94
T15	1.12	0	-3.36	-1.28	-0.25	233.33	-16.33	50	-69.44	-0.46	706.98

Tabela 44 – Metrics for Law Office SPL vs. Security Techniques

Law Office SPL	LOC	NOA	WOC	VS	LCOM	CDC	CDO	CDLOC	CBC	DIT	Scattering
Base	16665	743	1329	118	9254	0	0	0	0	369	0
Conditional Compilation											
T01	17201	750	1428	126	9376	0	0	0	0	395	0
T02	17149	752	1385	126	9327	0	0	0	0	391	0
T03	16746	743	1335	120	9256	0	0	0	0	372	0
T04	18245	781	1582	142	9539	0	0	0	0	459	0
T05	18239	777	1548	142	9527	0	0	0	0	463	0
T06	17570	774	1523	138	9521	0	0	0	0	453	0
T07	17612	780	1535	139	9537	0	0	0	0	455	0
T08	18436	794	1746	157	9776	0	0	0	0	517	0
T09	19214	814	1968	176	9905	0	0	0	0	569	0
T10	20603	856	2305	214	11118	0	0	0	0	695	0
T11	18482	795	1750	158	9785	0	0	0	0	519	0
T12	18496	795	1759	160	9780	0	0	0	0	525	0
T13	18827	804	1825	163	9806	0	0	0	0	527	0
T14	19762	837	2145	178	10675	0	0	0	0	573	0
T15	17882	766	1518	120	10042	0	0	0	0	373	0
T01- BASE	536	7	99	8	122	10	122	148	46	26	208
T02- BASE	484	9	56	8	73	10	105	148	36	22	191
T03- BASE	81	0	6	2	2	3	2	6	3	3	21
T04- BASE	1580	38	253	24	285	26	233	184	131	90	602
T05- BASE	1574	34	219	24	273	26	212	176	132	94	583
T06- BASE	905	31	194	20	267	22	113	10	119	84	62
T07- BASE	947	37	206	21	283	23	118	10	119	86	62
T08- BASE	1771	51	417	39	522	42	279	76	245	148	220
T09- BASE	2549	71	639	58	651	61	418	76	373	200	268
T10- BASE	3938	113	976	96	1864	101	638	80	558	326	405
T11- BASE	1817	52	421	40	531	43	281	76	246	150	221
T12- BASE	1831	52	430	42	526	48	289	76	287	156	248
T13- BASE	2162	61	496	45	552	50	403	242	268	158	441
T14- BASE	3097	94	816	60	1421	67	484	76	384	204	268
T15- BASE	1217	23	189	2	788	3	140	180	4	4	777
AspectJ											
T01	17641	750	1494	128	9311	0	0	0	0	397	0
T02	17560	752	1451	128	9262	0	0	0	0	393	0
T03	16762	743	1347	121	9321	0	0	0	0	374	0
T04	17769	781	1593	145	9439	0	0	0	0	462	0
T05	17835	777	1557	145	9426	0	0	0	0	466	0
T06	17595	774	1530	140	9420	0	0	0	0	455	0
T07	17637	780	1541	141	9434	0	0	0	0	457	0
T08	18470	794	1762	161	9670	0	0	0	0	521	0
T09	19248	813	1671	180	9813	0	0	0	0	461	0
T10	20645	856	2333	218	10522	0	0	0	0	699	0
T11	18569	795	1766	162	9677	0	0	0	0	523	0
T12	18524	795	1775	164	9674	0	0	0	0	529	0
T13	18722	803	1845	167	9700	0	0	0	0	531	0
T14	19796	837	2161	182	10583	0	0	0	0	577	0
T15	17619	766	1758	121	10042	0	0	0	0	374	0
T01- BASE	976	7	165	10	57	10	90	2	89	28	2
T02- BASE	895	9	122	10	8	10	73	2	77	24	0
T03- BASE	97	0	18	3	67	3	5	2	10	5	2
T04- BASE	1104	38	264	27	185	27	141	4	158	93	3
T05- BASE	1170	34	228	27	172	27	213	184	153	97	78
T06- BASE	930	31	201	22	166	22	109	4	136	86	3
T07- BASE	972	37	212	23	180	23	114	4	136	88	3
T08- BASE	1805	51	433	43	416	43	270	66	293	152	37
T09- BASE	2583	70	342	62	559	62	224	66	433	92	37
T10- BASE	3980	113	1004	100	1268	100	612	68	637	330	43
T11- BASE	1904	52	437	44	423	44	272	66	298	154	37
T12- BASE	1859	52	446	46	420	46	262	66	341	160	37
T13- BASE	2057	60	516	49	446	49	312	66	327	162	37
T14- BASE	3131	94	832	64	1329	64	467	68	444	208	37
T15- BASE	954	23	429	3	788	3	173	240	21	5	172

Tabela 45 – Difference between AOP and CC considering Law Office SPL.

CC vs AOP	LOC	NOA	WOC	VS	LCOO	CDC	CDO	CDLOC	CBC	DIT	Scattering
T01	-2.49	0	-4.42	-1.56	0.7	0	35.56	7300	-48.31	-0.5	10300
T02	-2.34	0	-4.55	-1.56	0.7	0	43.84	7300	-53.25	-0.51	-
T03	-0.1	0	-0.89	-0.83	-0.7	0	-60	200	-70	-0.53	950
T04	2.68	0	-0.69	-2.07	1.06	-3.7	65.25	4500	-17.09	-0.65	19966.67
T05	2.27	0	-0.58	-2.07	1.07	-3.7	-0.47	-4.35	-13.73	-0.64	647.44
T06	-0.14	0	-0.46	-1.43	1.07	0	3.67	150	-12.5	-0.44	1966.67
T07	-0.14	0	-0.39	-1.42	1.09	0	3.51	150	-12.5	-0.44	1966.67
T08	-0.18	0	-0.91	-2.48	1.1	-2.33	3.33	15.15	-16.38	-0.77	494.59
T09	-0.18	0.12	17.77	-2.22	0.94	-1.61	86.61	15.15	-13.86	23.43	624.32
T10	-0.2	0	-1.2	-1.83	5.66	1	4.25	17.65	-12.4	-0.57	841.86
T11	-0.47	0	-0.91	-2.47	1.12	-2.27	3.31	15.15	-17.45	-0.76	497.3
T12	-0.15	0	-0.9	-2.44	1.1	4.35	10.31	15.15	-15.84	-0.76	570.27
T13	0.56	0.12	-1.08	-2.4	1.09	2.04	29.17	266.67	-18.04	-0.75	1091.89
T14	-0.17	0	-0.74	-2.2	0.87	4.69	3.64	11.76	-13.51	-0.69	624.32
T15	1.49	0	-13.65	-0.83	0	0	-19.08	-25	-80.95	-0.27	351.74

Tabela 46 – Cliff's (δ), magnitude of effect power for size metrics.

Techniques	LOC		NOA		WOC		VS	
	δ de Cliff	Magnitude	δ de Cliff	Magnitude	δ de Cliff	Magnitude	δ de Cliff	Magnitude
T01	1	Large	0.3333	medium	0.5556	large	1	large
T02	1	large	0.3333	medium	1	large	1	large
T03	1	large	0.3333	medium	1	large	1	large
T04	-1	large	0.2222	small	0.5556	large	1	large
T05	-1	large	0.2222	small	0.5556	large	1	large
T06	0.3333	medium	0.2222	small	0.5556	large	1	large
T07	0.3333	medium	-0.3333	medium	0.5556	large	1	large
T08	-0.1111	negligible	0.3333	medium	0.5556	large	1	large
T09	-0.1111	negligible	0.1111	negligible	-0.1111	negligible	1	large
T10	-0.1111	negligible	0.2222	small	0.5556	large	0.7778	large
T11	0.5556	large	0.3333	medium	0.5556	large	1	large
T12	-0.1111	negligible	0.3333	medium	0.5556	large	1	large
T13	-0.5556	large	0.1111	negligible	0.5556	large	1	large
T14	0.3333	medium	0.3333	medium	0.5556	large	1	large
T15	-0.3333	medium	0	negligible	1	large	1	large

Tabela 47 – Cohen's d for effect power.

Techniques	LOC	NOA	WOC	VS
	d of Cohen	d of Cohen	d of Cohen	d of Cohen
T01	-3.3571	-0.8165	-1.5071	-Inf"
T02	-3.5517	-0.8165	-2.1346	-Inf"
T03	-5.0644	-0.8165	-2.8584	-Inf"
T04	4.3044	-0.6019	-0.3317	-5.1962
T05	2.4141	-0.6019	-0.4387	-5.1962
T06	-0.6795	-0.6019	-0.3861	-3.4641
T07	-0.627	0.5164	-0.383	-3.4641
T08	0.0361	-0.5661	-0.2475	-3.4641
T09	0.0306	-0.4182	0.4615	-3.4641
T10	0.0185	-0.4683	-0.1229	-1.7321
T11	-0.4986	-0.5661	-0.2831	-3.4641
T12	0.0257	-0.5661	-0.2719	-3.4641
T13	1.0015	-0.4182	-0.2673	-3.4641
T14	-0.8764	-0.5661	-0.1501	-3.4641
T15	0.4152	NaN	-2.1095	-Inf"

Tabela 48 – Cliff's (δ), magnitude of effect power for separation of concerns metrics.

Techniques	CDC		CDO		CDLOC		Scattering	
	δ de Cliff	Magnitude	δ de Cliff	Magnitude	δ de Cliff	Magnitude	δ de Cliff	Magnitude
T01	0	negligible	-1	large	-1	large	-1	large
T02	0	negligible	-1	large	-1	large	-1	large
T03	0	negligible	0.7778	large	-1	large	-1	large
T04	0.3333	medium	-1	large	-1	large	-1	large
T05	0.3333	medium	0.7778	large	0.3333	medium	-1	large
T06	-0.1111	negligible	-0.3333	medium	-0.5556	large	-1	large
T07	-0.1111	negligible	-0.3333	medium	-0.5556	large	-1	large
T08	0.3333	medium	-0.1111	negligible	0.1111	negligible	-1	large
T09	0.3333	medium	-0.7778	large	0.1111	negligible	-1	large
T10	-0.5556	large	-0.5556	large	0.1111	negligible	-1	large
T11	0.3333	medium	0.1111	negligible	0.1111	negligible	-1	large
T12	-0.7778	large	-0.1111	negligible	0.1111	negligible	-1	large
T13	-0.5556	large	-1	large	-1	large	-1	large
T14	-1	large	-0.3333	medium	0.1111	negligible	-1	large
T15	-0.6667	large	0.5556	large	0.1111	negligible	-1	large

Tabela 49 – Cohen's d for effect power of separation of concerns metrics.

Techniques	CDC	CDO	CDLOC	Scattering
	d of Cohen	d of Cohen	d of Cohen	d of Cohen
T01	NaN	2.3109	8.1245	6.6968
T02	NaN	2.7094	8.1245	7.133
T03	NaN	-1.633	Inf	8.2369
T04	0.2685	5.9604	4.2265	8.0022
T05	0.2685	-1.5172	-0.4116	5.1168
T06	0.6712	0	1.1371	2.0539
T07	0.6712	-0.0358	1.1371	1.9504
T08	0.2243	0.2073	-0.1893	2.5349
T09	0.2243	1.0238	-0.1893	3.0709
T10	0.6632	0.2167	-0.1996	3.7013
T11	0.2243	0.0989	-0.1893	2.5368
T12	1.2337	0.3565	-0.1893	2.8217
T13	0.8972	4.9516	1.949	3.5891
T14	1.7865	0.1525	-0.2594	3.0787
T15	1.449	-1.0006	-0.1797	2.4532

Tabela 50 – Cliff's (δ), magnitude of effect power for cohesion metrics.

Techniques	LCOO	
	δ de Cliff	Magnitude
T01	-0.7778	large
T02	-0.7778	large
T03	-0.3333	medium
T04	-0.7778	large
T05	-0.5556	large
T06	-0.7778	large
T07	-0.7778	large
T08	-0.7778	large
T09	-0.7778	large
T10	-1	large
T11	-0.7778	large
T12	-0.7778	large
T13	-0.7778	large
T14	-0.7778	large
T15	0.6667	large

Tabela 51 – Cohen's d for effect power of cohesion metrics.

Techniques	LCOO
	d of Cohen
T01	1.086
T02	1.0559
T03	0.2302
T04	1.6297
T05	1.386
T06	1.6247
T07	1.6012
T08	1.5184
T09	1.5683
T10	1.7989
T11	1.584
T12	1.5073
T13	1.528
T14	1.5784
T15	-1.0426

Tabela 52 – Cliff's (δ), magnitude of effect power for coupling metrics.

Techniques	CBC		DIT	
	δ de Cliff	Magnitude	δ de Cliff	Magnitude
T01	1	large	0.5556	large
T02	1	large	0.5556	large
T03	1	large	0.7778	large
T04	1	large	0.7778	large
T05	1	large	0.7778	large
T06	1	large	0.5556	large
T07	0.5556	large	0.5556	large
T08	1	large	0.5556	large
T09	1	large	-0.1111	negligible
T10	1	large	0.5556	large
T11	1	large	0.5556	large
T12	1	large	0.5556	large
T13	1	large	0.5556	large
T14	0.5556	large	0.5556	large
T15	1	large	0.5556	large

Tabela 53 – Cohen's d for effect power of coupling metrics.

Techniques	CDC	DIT
	d of Cohen	d of Cohen
T01	-3.4101	-0.3166
T02	-3.266	-0.3752
T03	-9.3897	-1.8257
T04	-2.7734	-0.9145
T05	-2.2575	-0.8981
T06	-1.9135	-0.0845
T07	-1.2243	-0.0825
T08	-4.9008	-0.0962
T09	-5.8837	0.5665
T10	-6.1018	-0.0436
T11	-5.1601	-0.0949
T12	-5.15	-0.0912
T13	-4.4865	-0.09
T14	-0.4194	-0.0633
T15	-2.258	-0.866

Referências

ABRAN, A. et al. *Guide to the Software Engineering Body of Knowledge - SWEBOK*. 2004 version. ed. Piscataway, NJ, USA: IEEE Press, 2004. 1–202 p.

ALMEIDA, E. S. D. *RiDE - The RiSE Process for Domain Engineering*. 2007. Tese (Ph.D thesis) — Universidade Federal de Pernambuco, Recife, Pernambuco, Brazil.

ALMEIDA, E. S. de; OQUENDO, F. Software components, architectures and reusemodeling, customization and evaluation. *Journal of Universal Computer Science*, v. 19, n. 2, p. 183–185, 2013.

ALVES, V. et al. Refactoring product lines. In: *Proceedings of the 5th International Conference on Generative Programming and Component Engineering*. New York, NY, USA: ACM, 2006. (GPCE '06), p. 201–210. ISBN 1-59593-237-2. Disponível em: <http://doi.acm.org/10.1145/1173706.1173737>.

ALVES, V. et al. From conditional compilation to aspects: a case study in software product lines migration. In: *Proceedings of the First Workshop on Aspect-Oriented Product Line Engineering*. New York, NY, USA: ACM, 2006.

AMORIM, S. da S.; ALMEIDA, E. S. D.; MCGREGOR, J. D. Extensibility in ecosystem architectures: an initial study. In: *The 1st International Workshop on Software Ecosystem Architectures, WEA 2013, Saint Petersburg, Russian Federation, August 19, 2013*. [S.l.: s.n.], 2013. p. 11–15.

AMORIM, S. da S.; ALMEIDA, E. S. de; MCGREGOR, J. D. Scalability of ecosystem architectures. In: *2014 IEEE/IFIP Conference on Software Architecture, WICSA 2014, Sydney, Australia, April 7-11, 2014*. [S.l.: s.n.], 2014. p. 49–52.

AMORIM, S. da S. et al. Flexibility in ecosystem architectures. In: *Proceedings of the ECSA 2014 Workshops & Tool Demos Track, European Conference on Software Architecture, 2014, Vienna, Austria*. [S.l.: s.n.], 2014. p. 14:1–14:6.

ANDRADE, H. S. de; ALMEIDA, E. S. de; CRNKOVIC, I. Architectural bad smells in software product lines: an exploratory study. In: *Proceedings of the WICSA Companion Volume, Sydney, NSW, Australia, April 7-11, 2014*. [S.l.: s.n.], 2014. p. 12:1–12:6.

AOYAMA, M.; YOSHINO, A. Aore (aspect-oriented requirements engineering) methodology for automotive software product lines. In: *Proceedings of the 2008 15th Asia-Pacific Software Engineering Conference*. Washington, DC, USA: IEEE Computer Society, 2008. (APSEC '08), p. 203–210. ISBN 978-0-7695-3446-6. Disponível em: <http://dx.doi.org/10.1109/APSEC.2008.59>.

APEL, S.; BEYER, D. Feature cohesion in software product lines: An exploratory study. In: *Proceedings of the 33rd International Conference on Software Engineering*. New York, NY, USA: ACM, 2011. (ICSE '11), p. 421–430. ISBN 978-1-4503-0445-0. Disponível em: <http://doi.acm.org/10.1145/1985793.1985851>.

APEL, S. et al. Exploring feature interactions in the wild: The new feature-interaction challenge. In: *Proceedings of the 5th International Workshop on Feature-Oriented Software Development*. New York, NY, USA: ACM, 2013. (FOSD '13), p. 1–8. ISBN 978-1-4503-2168-6. Disponível em: <http://doi.acm.org/10.1145/2528265.2528267>.

ARVANITOU, E. M. et al. A mapping study on design-time quality attributes and metrics. *Journal of Systems and Software*, v. 127, p. 52 – 77, 2017. ISSN 0164-1212. Disponível em: <//www.sciencedirect.com/science/article/pii/S016412121730016X>.

AYED, S. et al. Security aspects: A framework for enforcement of security policies using aop. In: *2013 International Conference on Signal-Image Technology Internet-Based Systems*. [S.l.: s.n.], 2013. p. 301–308.

BAGHERI, H.; SULLIVAN, K. J. A formal approach for incorporating architectural tactics into the software architecture. In: *SEKE*. [S.l.: s.n.], 2011. p. 770–775.

BARBACCI, M. et al. *Quality Attributes.* [S.l.], 1995.

BASILI, V.; CALDIERA, G.; ROMBACH, H. The Goal Question Metric Approach. *Encyclopedia of Software Engineering*, v. 1, p. 528–532, 1994.

BASILI, V. R.; SELBY, R.; HUTCHENS, D. Experimentation in Software Engineering. *IEEE Transactions on Software Engineering*, v. 12, n. 7, p. 733–743, July 1986.

BASILI, V. R.; SHULL, F.; LANUBILE, F. Building knowledge through families of experiments. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 25, n. 4, p. 456–473, jul. 1999. ISSN 0098-5589. Disponível em: <http://dx.doi.org/10.1109/32.799939>.

BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software Architecture in Practice*. 2. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003. ISBN 0321154959.

BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software Architecture in Practice*. 3rd. ed. [S.l.]: Addison-Wesley Professional, 2012. ISBN 0321815734, 9780321815736.

BASTOS, J. F. et al. Adopting software product lines: A systematic mapping study. In: *Evaluation Assessment in Software Engineering (EASE 2011), 15th Annual Conference on*. [S.l.: s.n.], 2011. p. 11–20.

BENAVIDES, D.; TRINIDAD, P.; RUIZ-CORTÉS, A. Automated reasoning on feature models. In: *Proceedings of the 17th International Conference on Advanced Information Systems Engineering*. Berlin, Heidelberg: Springer-Verlag, 2005. (CAiSE'05), p. 491–503. ISBN 3-540-26095-1, 978-3-540-26095-0. Disponível em: <http://dx.doi.org/10.1007/11431855_34>.

BERGER, T. et al. What is a feature?: a qualitative study of features in industrial software product lines. In: *Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20-24, 2015*. [S.l.: s.n.], 2015. p. 16–25.

BOSCH, J.; CAPILLA, R. Variability implementation. In: CAPILLA, R.; BOSCH, J.; KANG, K.-C. (Ed.). *Systems and Software Variability Management*. Springer Berlin Heidelberg, 2013. p. 75–86. ISBN 978-3-642-36582-9. Disponível em: <http://dx.doi.org/10.1007/978-3-642-36583-6_5>.

BRANCO, F. *Investigação Experimental: Potência estatística dos testes de aleatorização na comparação de dois grupos independentes*. 2010. Tese (Doutorado) — Universidade Aberta, Lisboa - Portugal.

BURGER, S.; HUMMEL, O.; HEINISCH, M. Airbus cabin software. *Software, IEEE*, v. 30, n. 1, p. 21–25, Jan 2013. ISSN 0740-7459.

CANDELA, I. et al. Using cohesion and coupling for software remodularization: Is it enough? *ACM Trans. Softw. Eng. Methodol.*, ACM, New York, NY, USA, v. 25, n. 3, p. 24:1–24:28, jun. 2016. ISSN 1049-331X. Disponível em: <http://doi.acm.org/10.1145/2928268>.

CARVALHO, M. L. L. *On the Implementation of Dynamic Software Product Lines: An Exploratory Study*. 2016. Dissertação (Mestrado) — Programa Multi-Institucional de Pós-Graduação em Ciência da Computação, Salvador, Bahia, Brazil.

CARVALHO, M. L. L. et al. On the implementation of dynamic software product lines: A preliminary study. In: *2016 X Brazilian Symposium on Software Components, Architectures and Reuse, SBCARS 2016, Maringá, Brazil, September 19-20, 2016*. [s.n.], 2016. p. 21–30. Disponível em: <http://dx.doi.org/10.1109/SBCARS.2016.13>.

CERVANTES, H. et al. Architectural approaches to security: Four case studies. *Computer*, v. 49, n. 11, p. 60–67, Nov 2016. ISSN 0018-9162.

CERVANTES, H.; VELASCO-ELIZONDO, P.; KAZMAN, R. A principled way to use frameworks in architecture design. *IEEE Software*, v. 30, n. 2, p. 46–53, March 2013. ISSN 0740-7459.

CHEN, L.; BABAR, M. A. A systematic review of evaluation of variability management approaches in software product lines. *Inf. Softw. Technol.*, Butterworth-Heinemann, Newton, MA, USA, v. 53, n. 4, p. 344–362, abr. 2011. ISSN 0950-5849. Disponível em: <http://dx.doi.org/10.1016/j.infsof.2010.12.006>.

CHIDAMBER, S. R.; KEMERER, C. F. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 20, n. 6, p. 476–493, jun. 1994. ISSN 0098-5589. Disponível em: <http://dx.doi.org/10.1109/32.295895>.

CLEMENTS, L. N. P. *Software Product Lines: Practices and Patterns*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001. ISBN 0-201-70332-7.

CLEMENTS, P.; NORTHROP, L. *Software Product Lines: Practices and Patterns*. Boston, MA, USA: Addison-Wesley, 2001.

CLIFF, N. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological Bulletin*, v. 114, p. 494 – 509, 11 1993.

CONOVER, W. *Practical nonparametric statistics*. 3. ed. ed. New York, NY [u.a.]: Wiley, 1999. (Wiley series in probability and statistics). ISBN 0471160687.

COTTON, J. W. *Elementary statistical theory for behavior scientists*. [S.l.]: Addison-Wesley, 1967.

CZARNECKI, K.; EISENECKER, U. W. *Generative Programming: Methods, Tools, and Applications*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000. ISBN 0-201-30977-7.

D., G.; M., F.-M. E. adn P.; C., G. Comparison of security patterns. *IJCSNS: International Journal of Computer Science and Network Security*, v. 6, n. 2B, p. 139–146, 2006. ISSN 1738-7906.

DANIEL, M.; EDUARDO, F.-M.; MARIO, P. Towards security requirements management for software product lines: A security domain requirements engineering process. *Comput. Stand. Interfaces*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 30, n. 6, p. 361–371, ago. 2008.

DANIEL, M.; EDUARDO, F.-M.; MARIO, P. Security requirements engineering framework for software product lines. *Inf. Softw. Technol.*, Butterworth-Heinemann, Newton, MA, USA, v. 52, n. 10, p. 1094–1117, out. 2010. ISSN 0950-5849.

DANTAS, F.; GARCIA, A. Software reuse versus stability: Evaluating advanced programming techniques. In: *Software Engineering (SBES), 2010 Brazilian Symposium on*. [S.l.: s.n.], 2010. p. 40–49.

DASHOFY, E. M.; HOEK, A. van der; TAYLOR, R. N. An infrastructure for the rapid development of xml-based architecture description languages. In: *Proceedings of the 24th International Conference on Software Engineering*. New York, NY, USA: ACM, 2002. (ICSE '02), p. 266–276. ISBN 1-58113-472-X. Disponível em: <http://doi.acm.org/10.1145/581339.581374>.

DENNING, D. E.; DENNING, P. J. Data security. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 11, n. 3, p. 227–249, set. 1979. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/356778.356782>.

DEPT, I. S. *IEEE Standard for a Software Quality Metrics Methodology*. [S.l.], 1998. i+ p.

DHUNGANA, D.; GRüNBACHER, P.; RABISER, R. The dopler meta-tool for decision-oriented variability modeling: A multiple case study. *Automated Software Engg.*, Kluwer Academic Publishers, Hingham, MA, USA, v. 18, n. 1, p. 77–114, mar. 2011. ISSN 0928-8910. Disponível em: <http://dx.doi.org/10.1007/s10515-010-0076-6>.

DIKEL, D. et al. Applying software product-line architecture. *Computer*, v. 30, n. 8, p. 49–55, Aug 1997. ISSN 0018-9162.

D'AMORIM, F.; BORBA, P. Modularity analysis of use case implementations. *Journal of Systems and Software*, v. 85, n. 4, p. 1012 – 1027, 2012. ISSN 0164-1212. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0164121211002950>.

EADDY, M. et al. Do crosscutting concerns cause defects? *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 34, n. 4, p. 497–515, jul. 2008. ISSN 0098-5589.

EDGINGTON, E. *Statistical Inference: The Distribution-Free Approach*. McGraw-Hill, 1969. (McGraw-Hill Series in Psychology). Disponível em: <https://books.google.com.br/books?id=nXWQAAAAIAAJ>.

EDGINGTON, E. S. Statistical inference and nonrandom samples. *Psychological bulletin*, American Psychological Association, v. 66, n. 6, p. 485, 1966.

EDGINGTON, E. S. Approximate randomization tests. *Journal of Psychology*, v. 72, p. 143–149, 1969.

EDGINGTON, E. S. Approximative randomization tests. *Journal of Psychology*, v. 72, p. 143–149, 1969.

EDGINGTON, E. S. *Randomization tests*. New York: Marcel Dekker, 1995.

ELLIS, P. *The Essential Guide to Effect Sizes: Statistical Power, Meta-Analysis, and the Interpretation of Research Results*. Cambridge University Press, 2010. ISBN 9780521142465. Disponível em: <https://books.google.com.br/books?id=5obZnfK5pbsC>.

FAEGRI, T.; HALLSTEINSEN, S. A Software Product Line Reference Architecture for Security. In: *Software Product Lines*. [S.l.: s.n.], 2006. p. 275–326.

FERNANDEZ-BUGLIONI, E. *Security Patterns in Practice: Designing Secure Architectures Using Software Patterns*. 1st. ed. [S.l.]: Wiley Publishing, 2013. ISBN 1119998948, 9781119998945.

FERNANDEZ, E. B.; ASTUDILLO, H.; PEDREZA-GARCIA gilberto. Revisiting architectural tactics for security. In: *European Conference on Software Architecture (ECSA 2015)*. [S.l.: s.n.], 2015. p. 85–91.

FERREIRA, G. C. S. et al. On the use of feature-oriented programming for evolving software product lines - A comparative study. *Sci. Comput. Program.*, v. 93, p. 65–85, 2014.

FIGUEIREDO, E. et al. Evolving software product lines with aspects: An empirical study on design stability. In: *Proceedings of the 30th International Conference on Software Engineering*. New York, NY, USA: ACM, 2008. (ICSE '08), p. 261–270. ISBN 978-1-60558-079-1. Disponível em: <http://doi.acm.org/10.1145/1368088.1368124>.

FIGUEIREDO, E. et al. Applying and evaluating concern-sensitive design heuristics. In: *Software Engineering, 2009. SBES '09. XXIII Brazilian Symposium on*. [S.l.: s.n.], 2009. p. 83–93.

FILMAN, R. et al. *Aspect-oriented Software Development*. First. [S.l.]: Addison-Wesley Professional, 2004. ISBN 0321219767.

FISHER, R. A. The coefficient of racial likeness and the future of craniometry. *Journal of the Anthropological Institute*, v. 66, p. 57–63, 1936.

FISHER, R. A. *The design of experiments (9e édition)*. New York: Hafner, 1971.

GACEK, C.; ANASTASOPOULES, M. Implementing product line variabilities. In: *Proceedings of the 2001 Symposium on Software Reusability: Putting Software Reuse in Context*. New York, NY, USA: ACM, 2001. (SSR '01), p. 109–117. ISBN 1-58113-358-8.

GAIA, F. N. et al. A quantitative and qualitative assessment of aspectual feature modules for evolving software product lines. *Sci. Comput. Program.*, v. 96, p. 230–253, 2014.

GALSTER, M. Architecting for variability in quality attributes of software systems. In: *Proceedings of the 2015 European Conference on Software Architecture Workshops*. New York, NY, USA: ACM, 2015. (ECSAW '15), p. 23:1–23:4. ISBN 978-1-4503-3393-1. Disponível em: <http://doi.acm.org/10.1145/2797433.2797456>.

GALSTER, M. et al. Variability in software systems—a systematic literature review. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 40, n. 3, p. 282–306, mar. 2014. ISSN 0098-5589. Disponível em: <http://dx.doi.org/10.1109/TSE.2013.56>.

GAO, D.; REITER, M. K.; SONG, D. Behavioral distance for intrusion detection. In: *Proceedings of the 8th International Conference on Recent Advances in Intrusion Detection*. Berlin, Heidelberg: Springer-Verlag, 2006. (RAID'05), p. 63–81. ISBN 3-540-31778-3, 978-3-540-31778-4.

GARCIA, A. et al. Modularizing design patterns with aspects: A quantitative study. In: *Proceedings of the 4th International Conference on Aspect-oriented Software Development*. New York, NY, USA: ACM, 2005. (AOSD '05), p. 3–14. ISBN 1-59593-042-6.

GEORG, G.; FRANCE, R.; RAY, I. An aspect-based approach to modeling security concerns. In: *In Proceedings of the Workshop on Critical Systems Development with UML*. [S.l.: s.n.], 2002. p. 107–120.

GOMAA, H. *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 2004. ISBN 0201775956.

GOMAA, H.; SHIN, M. E. Multiple-view meta-modeling of software product lines. In: *Engineering of Complex Computer Systems, 2002. Proceedings. Eighth IEEE International Conference on*. [S.l.: s.n.], 2002. p. 238–246.

GREENWOOD, P. et al. Ecoop 2007 – object-oriented programming: 21st european conference, berlin, germany, july 30 - august 3, 2007. proceedings. In: ____. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. cap. On the Impact of Aspectual Decompositions on Design Stability: An Empirical Study, p. 176–200.

GURP, J. van; BOSCH, J.; SVAHNBERG, M. On the notion of variability in software product lines. In: *Software Architecture, 2001. Proceedings. Working IEEE/IFIP Conference on*. [S.l.: s.n.], 2001. p. 45–54.

H., A. Kuiper's p-value as a measuring tool and decision procedure for the goodness-of-fit test. *Journal of Applied Statistics*, v. 15, n. 3, p. 131 – 135, 1988.

HADAR, I. et al. Comparing the comprehensibility of requirements models expressed in use case and tropos: Results from a family of experiments. *Inf. Softw. Technol.*, Butterworth-Heinemann, Newton, MA, USA, v. 55, n. 10, p. 1823–1843, out. 2013. ISSN 0950-5849. Disponível em: <http://dx.doi.org/10.1016/j.infsof.2013.05.003>.

HALLSTEINSEN, S. O. et al. Dealing with architectural variation in product populations. In: *Software Product Lines - Research Issues in Engineering and Management*. [s.n.], 2006. p. 245–273. Disponível em: <http://dx.doi.org/10.1007/978-3-540-33253-4_7>.

HAMED, H.; AL-SHAER, E. Taxonomy of conflicts in network security policies. *Communications Magazine, IEEE*, v. 44, n. 3, p. 134–141, March 2006.

HARRISON, N. B.; AVGERIOU, P. How do architecture patterns and tactics interact? a model and annotation. *J. Syst. Softw.*, Elsevier Science Inc., New York, NY, USA, v. 83, n. 10, p. 1735–1758, out. 2010. ISSN 0164-1212. Disponível em: <http://dx.doi.org/10.1016/j.jss.2010.04.067>.

HORCAS, J.-M.; PINTO, M.; FUENTES, L. An automatic process for weaving functional quality attributes using a software product line approach. *Journal of Systems and Software*, v. 112, p. 78 – 95, 2016. ISSN 0164-1212. Disponível em: <//www.sciencedirect.com/science/article/pii/S016412121500240X>.

HUI, Z. et al. A taxonomy of software security defects for sst. In: *Intelligent Computing and Integrated Systems (ICISS), 2010 International Conference on*. [S.l.: s.n.], 2010. p. 99–103.

IEE, E. IEEE Std 1061-1998. *IEEE Standard for a Software Quality Metrics Methodology*, 1998.

IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990*, p. 1–84, Dec 1990.

IIT, J. S.; SINGER, J. Using the american psychological association (apa) style guidelines to report experimental results janice singer. In: *In: Proceedings of the Fifth IEEE Workshop on Empirical Studies of Software Maintenance (WESS99*. [S.l.: s.n.], 1999. p. 71–75.

ISO. International standard - ISO/IEC 14764 IEEE Std 14764-2006. *ISO/IEC 14764:2006 (E) IEEE Std 14764-2006 Revision of IEEE Std 1219-1998)*, p. 1–46, 2006.

(ISO), I. O. for S. *ISO/IEC 25000:2005, Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE)*. [S.l.], 20011.

ISO/IEC. *ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models*. [S.l.], 2010.

JEDLITSCHKA, A.; PFAHL, D. Reporting guidelines for controlled experiments in software engineering. In: *Empirical Software Engineering, 2005. 2005 International Symposium on*. [S.l.: s.n.], 2005. p. 10 pp.–.

JURISTO, N.; MORENO, A. M. *Basics of Software Engineering Experimentation*. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2010. ISBN 1441950117, 9781441950116.

KALMUS, H. The design and analysis of experiments. by oscar kempthorne. new york. *Annals of Eugenics*, Blackwell Publishing Ltd, v. 17, n. 1, p. 96–97, 1952. ISSN 2050-1439. Disponível em: <http://dx.doi.org/10.1111/j.1469-1809.1952.tb02500.x>.

KANG, K. C. et al. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. [S.l.], 1990.

KASTNER, C.; APEL, S.; BATORY, D. A case study implementing features using aspectj. In: *Proceedings of the 11th International Software Product Line Conference*. Washington, DC, USA: IEEE Computer Society, 2007. (SPLC '07), p. 223–232. ISBN 0-7695-2888-0. Disponível em: <http://dx.doi.org/10.1109/SPLC.2007.5>.

KEMPTHORNE, O. The randomization theory of experimental inference. *Journal of the American Statistical Association*, [American Statistical Association, Taylor Francis, Ltd.], v. 50, n. 271, p. 946–967, 1955. ISSN 01621459. Disponível em: <http://www.jstor.org/stable/2281178>.

KEPPEL, T. D. W. G. *Design and analysis: A researcher handbook*. [S.l.]: Englewood Cliffs, NJ: Prentice-Hal l., 1973.

KICZALES, G. et al. Getting started with aspectj. *Commun. ACM*, ACM, New York, NY, USA, v. 44, n. 10, p. 59–65, out. 2001. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/383845.383858>.

KICZALES, G. et al. Aspect-oriented programming. In: AKşIT, M.; MATSUOKA, S. (Ed.). *ECOOP'97 — Object-Oriented Programming*. Springer Berlin Heidelberg, 1997, (Lecture Notes in Computer Science, v. 1241). p. 220–242. ISBN 978-3-540-63089-0. Disponível em: <http://dx.doi.org/10.1007/BFb0053381>.

KING, B.; MINIUM, E. *Statistical Reasoning in Psychology and Education*. Wiley, 2003. ISBN 9780471211877. Disponível em: <https://books.google.com.br/books?id=2V99AAAAMAAJ>.

KIRK, R. E. *Introductory statistics*. [S.l.]: Belmont, CA: Brooks/Cole, 1968.

KITCHENHAM, B. et al. Preliminary guidelines for empirical research in software engineering. *Software Engineering, IEEE Transactions on*, v. 28, n. 8, p. 721–734, Aug 2002. ISSN 0098-5589.

KROMREY, J. D.; HOGARTY, K. Y. Analysis options for testing group differences on ordered categorical variables: An empirical investigation of type i error control and statistical power. *Multiple Linear Regression Viewpoints*, v. 25, p. 70 – 82, 1998.

KRUEGER, C. W. Easing the transition to software mass customization. In: *Revised Papers from the 4th International Workshop on Software Product-Family Engineering*. London, UK, UK: Springer-Verlag, 2002. (PFE '01), p. 282–293. ISBN 3-540-43659-6. Disponível em: <http://dl.acm.org/citation.cfm?id=648114.748909>.

KRUEGER, C. W. Easing the transition to software mass customization. In: *Revised Papers from the 4th International Workshop on Software Product-Family Engineering*. London, UK, UK: Springer-Verlag, 2002. (PFE '01), p. 282–293. ISBN 3-540-43659-6. Disponível em: <http://dl.acm.org/citation.cfm?id=648114.748909>.

KULESZA, U. et al. Quantifying the effects of aspect-oriented programming: A maintenance study. In: *2006 22nd IEEE International Conference on Software Maintenance*. [S.l.: s.n.], 2006. p. 223–233. ISSN 1063-6773.

KULESZA, U. et al. Quantifying the effects of aspect-oriented programming: A maintenance study. In: *Proceedings of the 22Nd IEEE International Conference on Software Maintenance*. Washington, DC, USA: IEEE Computer Society,

2006. (ICSM '06), p. 223–233. ISBN 0-7695-2354-4. Disponível em: <http://dx.doi.org/10.1109/ICSM.2006.48>.

L., W. B. The significance of the differences between two means when the population variations are unequal. *Biometrika*, v. 29, p. 350–362, 1938.

LEE, J.; HWANG, S. A review on variability mechanisms for product lines. *Int. J. Adv. Media Commun.*, Inderscience Publishers, v. 5, n. 2/3, p. 172–181, abr. 2014. ISSN 1462-4613.

LEUNG, H. K. N.; WHITE, L. Insights into regression testing. In: *ICSM '89: Proceedings of the International Conference on Software Maintenance*. [S.l.: s.n.], 1989. p. 60–69.

LIENTZ, B. P.; SWANSON, E. B. *Software Maintenance Management*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1980.

LINCKE, R.; LUNDBERG, J.; LöWE, W. Comparing software metrics tools. In: *Proceedings of the 2008 International Symposium on Software Testing and Analysis*. New York, NY, USA: ACM, 2008. (ISSTA '08), p. 131–142. ISBN 978-1-60558-050-0. Disponível em: <http://doi.acm.org/10.1145/1390630.1390648>.

LINDEN, F. J. v. d.; SCHMID, K.; ROMMES, E. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.

LISBOA, L. B. et al. Toolday: a tool for domain analysis. *STTT*, v. 13, n. 4, p. 337–353, 2011. Disponível em: <http://dx.doi.org/10.1007/s10009-010-0174-6>.

LOHMANN, D. et al. On the configuration of nonfunctional properties in operating system product lines. In: *Proceedings of the 4th AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software.* [S.l.: s.n.], 2005. (AOSD).

MACBETH, G.; RAZUMIEJCZYK, E.; LEDESMA, R. D. Cliff's delta calculator: A non-parametric effect size program for two groups of observations. *Universitas Psychologica*, scieloco, v. 10, p. 545 – 555, 05 2011. ISSN 1657-9267.

MARI, M.; EILA, N. The impact of maintainability on component-based software systems. In: *Euromicro Conference, 2003. Proceedings. 29th*. [S.l.: s.n.], 2003. p. 25–32. ISSN 1089-6503.

MCGREGOR, J.; SODHANI, P.; MADHAVAPEDDI, S. Testing Variability in a Software Product Line. In: *SPLIT '04: Proceedings of the International Workshop on Software Product Line Testing*. Boston, Massachusetts, USA: [s.n.], 2004. p. 45.

MCGREGOR, J. D. Ten years of the arcade game maker pedagogical product line. In: *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools - Volume 2*. New York, NY, USA: ACM, 2014. (SPLC '14), p. 24–25. ISBN 978-1-4503-2739-8. Disponível em: <http://doi.acm.org/10.1145/2647908.2655962>.

MCILROY, D. Mass-produced software components. In: *ICSE '68: Proceedings of the 1st International Conference on Software Engineering*. Garmisch Pattenkirchen, Germany: [s.n.], 1968. p. 88–98.

MELLADO, D.; FERNÁNDEZ-MEDINA, E.; PIATTINI, M. Security requirements management in software product line engineering. In: FILIPE, J.; OBAIDAT, M. S. (Ed.). *e-Business and Telecommunications: International Conference, ICETE 2008, Porto, Portugal, July 26-29, 2008, Revised Selected Papers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 250–263.

MIRAKHORLI, M.; MäDER, P.; CLELAND-HUANG, J. Variability points and design pattern usage in architectural tactics. In: *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. New York, NY, USA: ACM, 2012. (FSE '12), p. 52:1–52:11.

MYLLARNIEMI, V. *Quality Attribute Variability in Software Product Lines*. 2015. Tese (Doutorado) — Aalto University, Finland.

MYLLäRNIEMI, V.; RAATIKAINEN, M.; MäNNISTö, T. A systematically conducted literature review: Quality attribute variability in software product lines. In: *Proceedings of the 16th International Software Product Line Conference - Volume 1*. New York, NY, USA: ACM, 2012. (SPLC '12), p. 41–45. ISBN 978-1-4503-1094-9. Disponível em: <http://doi.acm.org/10.1145/2362536.2362546>.

MYLLäRNIEMI, V.; RAATIKAINEN, M.; MäNNISTö, T. Representing and configuring security variability in software product lines. In: *Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures*. New York, NY, USA: ACM, 2015. (QoSA '15), p. 1–10.

MYLLäRNIEMI, V.; RAATIKAINEN, M.; MäNNISTö, T. Inter-organisational approach in rapid software product family development — a case study. In: MORISIO, M. (Ed.). *Reuse of Off-the-Shelf Components*. [S.l.]: Springer Berlin Heidelberg, 2006. (Lecture Notes in Computer Science, v. 4039), p. 73–86.

MYLOPOULOS, J.; CHUNG, L.; NIXON, B. Representing and using nonfunctional requirements: A process-oriented approach. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 18, n. 6, p. 488–497, jun. 1992. ISSN 0098-5589. Disponível em: <http://dl.acm.org/citation.cfm?id=129962.129966>.

NAKAGAWA, E. Y.; ANTONINO, P. O.; BECKER, M. Reference architecture and product line architecture: A subtle but critical difference. In: *Software Architecture - 5th European Conference, ECSA 2011, Essen, Germany, September 13-16, 2011. Proceedings*. [s.n.], 2011. p. 207–211. Disponível em: <http://dx.doi.org/10.1007/978-3-642-23798-0_22>.

NGUYEN, Q. L. Non-functional requirements analysis modeling for software product lines. In: *Proceedings of the 2009 ICSE Workshop on Modeling in Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2009. (MISE '09), p. 56–61. ISBN 978-1-4244-3722-1. Disponível em: <http://dx.doi.org/10.1109/MISE.2009.5069898>.

NORTHROP, L. M. Sei's software product line tenets. *IEEE Software*, Los Alamitos, CA, USA, v. 19, n. 4, p. 32–40, 2002.

NUNES, C. et al. Assessment of the design modularity and stability of multi-agent system product lines. *J. UCS*, v. 15, n. 11, p. 2254–2283, 2009.

NUñEZ-VARELA, A. S. et al. Source code metrics: A systematic mapping study. *Journal of Systems and Software*, v. 128, p. 164 – 197, 2017. ISSN 0164-1212. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0164121217300663>.

PITMAN, E. Significance Tests Which May be Applied to Samples from Any Population. *Journal of the Royal Statistical Society*, v. 4, n. 1, 1937.

PITMAN, E. J. G. Significance tests which may be applied to samples from any populations. ii. the correlation coefficient test. *Supplement to the Journal of the Royal Statistical Society*, [Wiley, Royal Statistical Society], v. 4, n. 2, p. 225–232, 1937. ISSN 14666162. Disponível em: <http://www.jstor.org/stable/2983647>.

PITMAN, E. J. G. Significance tests which may be applied to samples from any population. iii. the analysis of variance test. *Biometrika*, v. 29, p. 322–335, 1938.

POHL, K.; BOCKLE, G.; LINDEN, F. J. v. d. *Software Product Line Engineering: Foundations, Principles and Techniques*. Secaucus, NJ, USA: Springer-Verlag, 2005.

PRESCHERN, C. Catalog of security tactics linked to common criteria requirements. In: *19th Conference on Pattern Languages of Programs*. [S.l.: s.n.], 2012.

PRESSMAN, R. S. *Software Engineering: A Practitioner's Approach*. eigth. [S.l.]: McGrap-Hill, 2014.

RAY, I. et al. An aspect-based approach to modeling access control concerns. *Information and Software Technology*, v. 46, n. 9, p. 575–587, 2004. Disponível em: <http://dblp.uni-trier.de/db/journals/infsof/infsof46.html#RayFLG04>.

REGNELL, B.; SVENSSON, R.; OLSSON, T. Supporting roadmapping of quality requirements. *Software, IEEE*, v. 25, n. 2, p. 42–47, March 2008. ISSN 0740-7459.

RIBEIRO, V. V.; TRAVASSOS, G. H. Testing non-functional requirements: Lacking of technologies or researching opportunities? In: *XV Braziliam Symposium on Software Quality*. [S.l.: s.n.], 2016. p. 110–119.

ROBINSON, A. Randomization, bootstrap and monte carlo methods in biology. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, Blackwell Publishing Ltd, v. 170, n. 3, p. 856–856, 2007. ISSN 1467-985X. Disponível em: <http://dx.doi.org/10.1111/j.1467-985X.2007.00485_5.x>.

ROZANSKI, N.; WOODS, E. *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives*. [S.l.]: Addison-Wesley Professional, 2005. ISBN 0321112296.

RUNESON, P.; HöST, M. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, Springer US, v. 14, n. 2, p. 131–164, 2009. ISSN 1382-3256. Disponível em: <http://dx.doi.org/10.1007/s10664-008-9102-8>.

RYOO, J.; KAZMAN, R.; ANAND, P. Architectural analysis for security. *IEEE Security and Privacy*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 13, n. 6, p. 52–59, nov. 2015. ISSN 1540-7993. Disponível em: <http://dx.doi.org/10.1109/MSP.2015.126>.

RYOO, J.; LAPLANTE, P.; KAZMAN, R. Revising a security tactics hierarchy through decomposition, reclassification, and derivation. In: *Software Security and Reliability Companion (SERE-C), 2012 IEEE Sixth International Conference on*. [S.l.: s.n.], 2012. p. 85–91.

SANDHU, R.; SAMARATI, P. Access control: principle and practice. *Communications Magazine, IEEE*, v. 32, n. 9, p. 40–48, Sept 1994. ISSN 0163-6804.

SANT'ANNA, C. et al. On the reuse and maintenance of aspect-oriented software: An assessment framework. In: *Proceedings XVII Brazilian Symposium on Software Engineering*. [s.n.], 2003. Disponível em: <http://twiki.im.ufba.br/pub/Aside/NossasPublicacoes/sbes2003-135.PDF>.

SANT'ANNA, C. N. *Manutenibilidade e Reusabilidade de Software Orientado a Aspectos: Um Framework de Avaliação*. 2004. Tese (Doutorado) — Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Rio de Janeiro, Brazil.

SARAIVA, J. de A. G. et al. Classifying metrics for assessing object-oriented software maintainability: A family of metrics' catalogs. *Journal of Systems and Software*, v. 103, p. 85–101, 2015.

SCHMID, K.; JOHN, I. A customizable approach to full lifecycle variability management. *Sci. Comput. Program.*, Elsevier North-Holland, Inc., Amsterdam, The Netherlands, The Netherlands, v. 53, n. 3, p. 259–284, dez. 2004. ISSN 0167-6423. Disponível em: <http://dx.doi.org/10.1016/j.scico.2003.04.002>.

SCHMIDT, D.; BUSCHMANN, F. Patterns, frameworks, and middleware: their synergistic relationships. In: *Software Engineering, 2003. Proceedings. 25th International Conference on*. [S.l.: s.n.], 2003. p. 694–704. ISSN 0270-5257.

SCHULZE, S. et al. Does the discipline of preprocessor annotations matter?: A controlled experiment. In: *Proceedings of the 12th International Conference on Generative Programming: Concepts & Experiences*. New York, NY, USA: ACM, 2013. (GPCE '13), p. 65–74. ISBN 978-1-4503-2373-4. Disponível em: <http://doi.acm.org/10.1145/2517208.2517215>.

SCHUMACHER, M. et al. *Security Patterns: Integrating Security and Systems Engineering*. [S.l.]: John Wiley & Sons, 2005. ISBN 0470858842.

SHULL, F. J. et al. The role of replications in empirical software engineering. *Empirical Softw. Engg.*, Kluwer Academic Publishers, Hingham, MA, USA, v. 13, n. 2, p. 211–218, abr. 2008. ISSN 1382-3256. Disponível em: <http://dx.doi.org/10.1007/s10664-008-9060-1>.

SIEGMUND, N. et al. Measuring non-functional properties in software product line for product derivation. In: *Software Engineering Conference, 2008. APSEC '08. 15th Asia-Pacific*. [S.l.: s.n.], 2008. p. 187–194. ISSN 1530-1362.

SILVEIRA, P. et al. Rise events - a testbed for software product lines experimentation. In: *First International Workshop on Variability and Complexity in Software Design*. [S.l.: s.n.], 2016. p. 40–49.

SINCERO, J.; SCHRODER-PREIKSCHAT, W.; SPINCZYK, O. Approaching non-functional properties of software product lines: Learning from products. In: *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*. [S.l.: s.n.], 2010. p. 147–155. ISSN 1530-1362.

SOARES, L. R. et al. Analysis of non-functional properties in software product lines: A systematic review. In: *Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on*. [S.l.: s.n.], 2014. p. 328–335.

SOFTWARE Engineering - Product Quality, ISO/IEC 9126-1. [S.l.], 2001.

SOMMERVILLE, I. *Software Engineering*. 9. ed. Harlow, England: [s.n.], 2010.

SPENCE, J. T. et al. *Elementary statistics*. [S.l.]: New York: Appleton-Century-Crofts, 1976.

SPENCE, J. T. et al. *Randomization tests (4th ed.)*. [S.l.]: Boca Raton, FL: Chapman and Hall/CRC., 2007.

SPINELLIS, D. Tool writing: A forgotten art? *IEEE Software*, v. 22, n. 4, p. 9–11, July/August 2005. ISSN 0740-7459. Disponível em: <http://www.dmst.aueb.gr/dds/pubs/jrnl/2005-IEEESW-TotT/html/v22n4.html>.

STEEL, C.; NAGAPPAN, R.; LAI, R. *Core security patterns: Best practices and strategies for J2EE, Web services, and identity management*. Prentice-Hall, 2006. (Prentice Hall Core Series). Disponível em: <http://www.coresecuritypatterns.com/>.

SUN, S.; PAN, W.; WANG, L. L. A comprehensive review of effect size reporting and interpreting practices in academic journals in education and psychology. *Journal of Educational Psychology*, v. 102, p. 989–1004, 11 2010.

SVAHNBERG, M.; GURP, J. van; BOSCH, J. A taxonomy of variability realization techniques. *Software Practice and Experience*, New York, NY, USA, v. 35, n. 8, p. 705–754, 2005.

TAYLOR, R. N.; MEDVIDOVIC, N.; DASHOFY, E. M. *Software Architecture: Foundations, Theory, and Practice*. [S.l.]: Wiley Publishing, 2009. ISBN 0470167742, 9780470167748.

TRACZ, W. Software reuse myths. *ACM SIGSOFT Software Engineering Notes*, New York, NY, USA, v. 13, n. 1, p. 17–21, 1988.

VALE, G. A. D.; FIGUEIREDO, E. M. L. A method to derive metric thresholds for software product lines. In: *29th Brazilian Symposium on Software Engineering*. [S.l.: s.n.], 2015. p. 110–119.

VERDON, D. Security policies and the software developer. *Security Privacy, IEEE*, v. 4, n. 4, p. 42–49, July 2006. ISSN 1540-7993.

VIOLA, D. N. *Detecção e modelagem de padrão espacial em dados binários e de contagem.* 2007. Tese (Doutorado) — USP: Escola Superior de Agricultura “Luiz de Queiroz”, Piracicaba.

WAGNER, S. *Software Product Quality Control*. Springer, 2013. ISBN 978-3-642-38570-4. Disponível em: <http://dx.doi.org/10.1007/978-3-642-38571-1>.

WAHL, N. J. An overview of regression testing. *ACM SIGSOFT Software Engineering Notes*, New York, NY, USA, v. 24, n. 1, p. 69–73, 1999. ISSN 0163-5948.

WANG, Y. et al. Pla-based runtime dynamism in support of privacy-enhanced web personalization. In: *Software Product Line Conference, 2006 10th International*. [S.l.: s.n.], 2006. p. 10 pp.–162.

WESTFALL, P.; YOUNG, S. *Resampling-Based Multiple Testing: Examples and Methods for P-Value Adjustment*. Wiley, 1993. (A Wiley-Interscience publication). ISBN 9780471557616. Disponível em: <https://books.google.com.br/books?id=nuQXORVGI1QC>.

WOHLIN, C. et al. *Experimentation in Software Engineering: An Introduction*. Norwell, MA, USA: Kluwer Academic Publishers, 2012. ISBN 0-7923-8682-5.

WOODS, E.; ROZANSKI, N. Using architectural perspectives. In: *Software Architecture, 2005. WICSA 2005. 5th Working IEEE/IFIP Conference on*. [S.l.: s.n.], 2005. p. 25–35.

ZHANG, H.; JARZABEK, S.; YANG, B. Quality prediction and assessment for product lines. In: *Proceedings of the 15th International Conference on Advanced Information Systems Engineering*. Berlin, Heidelberg: Springer-Verlag, 2003. (CAiSE'03), p. 681–695. ISBN 3-540-40442-2. Disponível em: <http://dl.acm.org/citation.cfm?id=1758398.1758458>.