“RiPLE-RE: A Requirements Engineering Process for Software Product Lines”

Danuza Ferreira Santana Neiva

M.Sc. DISSERTATION

Universidade Federal de Pernambuco
Centro de Informática

Recife – PE
August, 2009
"RiPLE-RE: A Requirements Engineering Process for Software Product Lines"

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

A M.Sc. Dissertation presented to the Federal University of Pernambuco in partial fulfillment of the requirements for the degree of M.Sc. in Computer Science.

Advisor: Silvio Romero de Lemos Meira
Co-advisor: Eduardo Santana de Almeida

Recife – PE
August, 2009
This work is dedicated to God, my lovely parents and my siblings.
Acknowledgements

There are a lot of people that directly or indirectly participated on the results of this work. Without all them, I could not go the entire Master path. My excuses for the ones I forgot, it does not mean you were not important.

Initially, I would like to thank all who helped me to start the Master, in special to Christina von Flach, Ecivaldo Matos and Claudete Alves for recognizing my abilities and recommending my work.

Next, I would like to thank the Reuse in Software Engineering (RiSE) group to give me the support and the space to discuss the aspects that were useful to my work. In special, my co-advisor Eduardo Almeida, for his patience in giving me the directions for my Master. Thiago Burgos, Liana Lisboa, Ednaldo Dilorenzo, Yguaratã Cavalcanti, Vinicius Garcia, Paulo Silveira and Ivan Machado, for their revisions, thoughts and comments. My advisor, Silvio Meira, for accepting me as his student.

I would like to thank all the students who participated in the evaluation of this work, thanks for their dedication and patience.

There are people that I have special thanks, friends who helped me feel a little bit like home during this almost two years living in Recife. Adriana Ribeiro, a syster who was together me since my transition from Salvador until Recife, in moments more difficult. Ana, for her care and support. Micheline and Aliny, for their reflections, thoughts and companionship. Wagner and Mário, for their help in many situations. Wilnara and Tereza Barros, for their advices and lessons.

I would like to thank to the friends who helped me in the last moments for finishing of this work. Felipe Dias e Rogério, thanks for their reviews and companionship.

My gratitude to my family is immeasurable. My lovely mom, Rita, a great example for my life, a person to whom I dedicate all my conquests. My father, Antonio, for their support for me to get here. My siblings (or children), Danilla, Tony and Lorena, for their care, respect and love. Maria, for being a mother, person that helped me in adverse situations. Without all of you this could have never been possible. There were many moments in which I could not be present due to the distance, however, you always understood me.

Finally, I would like to thank God for giving me the wisdom and force to perform this work. In all the moments, You never abandoned me!
“De tudo ficaram três coisas:
a certeza de que estava sempre começando,
a certeza de que era preciso continuar
e a certeza de que seria interrompido antes de terminar.
Fazer da interrupção um caminho novo,
fazer da queda, um passo de dança,
do medo, uma escada,
do sonho, uma ponte,
da procura, um encontro.”
—FERNANDO SABINO (1923-2004)
Resumo

Linhas de Produto de Software é uma importante estratégia de reuso para minimizar custos e tempo de entrega das aplicações, e maximizar a qualidade e produtividade do desenvolvimento de software. Entretanto, isso envolve o gerenciamento dos pontos comuns e variáveis entre diferentes aplicações, que aumenta sua complexidade quando comparado com desenvolvimento de software tradicional. Assim, desenvolver uma Linha de Produto requer tempo e planejamento para apresentar resultados positivos, ao contrário, o investimento pode ser perdido devido a falhas no projeto.

Nesse contexto, um processo de Engenharia de Requisitos é importante para reduzir os riscos envolvidos em uma Linha de Produto, fornecendo gerenciamento e desenvolvimento de requisitos corretos. Por outro lado, existe um desafio chave em Engenharia de Requisitos para Linhas de Produto, que envolve uma solução adequada para gerenciar variabilidades, integrando-as e relacionando decisões em diferentes artefatos para facilitar a derivação de produtos. Assim, o desenvolvimento de Linhas de Produto deve ser apoiado por um processo de Engenharia de Requisitos adequado para o seu contexto.

Atualmente, existem muitas abordagens de Engenharia de Requisitos para Linhas de Produto, entretanto, elas apresentam alguns problemas, tais como a ausência de um processo completo e sistemático, com detalhes suficientes para o ciclo de vida da Engenharia de Requisitos. Assim, este trabalho define um processo sistemático de Engenharia de Requisitos, descrevendo atividades, tarefas, entradas, saídas, papéis e guidelines para o contexto de Linhas de Produto, em uma forma usável, efetiva e eficiente. Por fim, um estudo experimental é apresentado para identificar a viabilidade do processo proposto.

**Palavras-chave:** Linhas de Produto de Software, Engenharia de Requisitos, Reuso de Software
Abstract

Software Product Lines (SPL) is an important reuse strategy to minimize costs and time-to-market, and maximize quality and productivity of the software development. However, it involves the management of variabilities and commonalities among several applications, which increases its complexity compared to traditional software development. Thus, developing a SPL requires time and systematic planning to present positive results, otherwise, the investment can be lost due to failures in the project.

In this context, a Requirements Engineering (RE) process is important to reduce the risks involved in a SPL, providing correct requirements development and management. On the other hand, there is a key challenge in RE for SPL, which involves an adequate solution to manage variabilities, integrating them and linking decisions in different assets for making easy the products derivation. Thus, the SPL development should be supported by an appropriate RE process for its context.

Currently, there are several RE approaches for SPL, however, they present gaps, such as not presenting the complete and systematic process, with sufficient details to cover the whole RE lifecycle. Thus, this work defines a systematic requirements engineering process, describing activities, tasks, inputs, outputs, roles and guidelines for the SPL context, in an usable, effective and efficient way. Finally, an initial experimental study is presented to identify the viability of this proposed process.

**Keywords:** Software Product Lines, Requirements Engineering, Software Reuse
# Table of Contents

## 1 Introduction

1.1 Motivation ................................................. 1
1.2 Goal of the Dissertation .............................. 2
1.3 Overview of the Proposed Solution ...................... 2
  1.3.1 Context ........................................ 3
  1.3.2 Proposal Outline ................................ 5
1.4 Out of Scope ............................................. 5
1.5 Statement of the Contribution ............................ 5
1.6 Organization of the Dissertation ....................... 6

## 2 SPL and RE

2.1 Software Product Lines .................................... 7
  2.1.1 SPL Strategies .................................... 8
  2.1.2 SPL Essential Activities ........................... 9
2.2 Requirements Engineering ................................. 10
2.3 Requirements Engineering for SPL ...................... 11
  2.3.1 Risks and Challenges ............................... 15
2.4 Chapter Summary ....................................... 16

## 3 A Systematic Review

3.1 Introduction ............................................. 17
3.2 Planning .................................................. 18
  3.2.1 Protocol Definition ................................ 18
  3.2.2 Research Question ................................. 19
3.3 Conducting ............................................... 19
  3.3.1 Search Strategy .................................... 20
  3.3.2 Data Sources ....................................... 20
  3.3.3 Studies Selection .................................. 21
  3.3.4 Quality Assessment ................................ 22
  3.3.5 Data Collection .................................... 23
  3.3.6 Data Analysis and Synthesis ....................... 23
3.4 Reporting ................................................. 26
  3.4.1 Requirements Engineering Activities ............... 26
  3.4.2 Requirements Engineering Models and Techniques 27
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4.3 Variability</td>
<td>28</td>
</tr>
<tr>
<td>3.4.4 Review Summary</td>
<td>29</td>
</tr>
<tr>
<td>3.5 Threats to Validity</td>
<td>31</td>
</tr>
<tr>
<td>3.6 Related Work</td>
<td>32</td>
</tr>
<tr>
<td>3.7 Chapter Summary</td>
<td>33</td>
</tr>
<tr>
<td>4 RiPLE-RE Process</td>
<td>34</td>
</tr>
<tr>
<td>4.1 Introduction</td>
<td>34</td>
</tr>
<tr>
<td>4.2 Elicit</td>
<td>38</td>
</tr>
<tr>
<td>4.3 Model Feature</td>
<td>40</td>
</tr>
<tr>
<td>4.3.1 Identify Features</td>
<td>41</td>
</tr>
<tr>
<td>4.3.2 Represent Features</td>
<td>43</td>
</tr>
<tr>
<td>4.4 Describe Requirements</td>
<td>46</td>
</tr>
<tr>
<td>4.4.1 Classify Requirements</td>
<td>48</td>
</tr>
<tr>
<td>4.4.2 Detail Requirements</td>
<td>48</td>
</tr>
<tr>
<td>4.5 Define Use Cases</td>
<td>54</td>
</tr>
<tr>
<td>4.5.1 Identify Use Cases</td>
<td>55</td>
</tr>
<tr>
<td>4.5.2 Detail Use Cases</td>
<td>57</td>
</tr>
<tr>
<td>4.6 Verify</td>
<td>63</td>
</tr>
<tr>
<td>4.7 Evolution Management</td>
<td>66</td>
</tr>
<tr>
<td>4.7.1 Traceability</td>
<td>68</td>
</tr>
<tr>
<td>4.7.2 Traceability Tool</td>
<td>71</td>
</tr>
<tr>
<td>4.8 Chapter Summary</td>
<td>71</td>
</tr>
<tr>
<td>5 The RiPLE-RE Evaluation</td>
<td>74</td>
</tr>
<tr>
<td>5.1 Introduction</td>
<td>74</td>
</tr>
<tr>
<td>5.2 The Experimental Study</td>
<td>75</td>
</tr>
<tr>
<td>5.2.1 The Definition</td>
<td>75</td>
</tr>
<tr>
<td>5.2.1.1 Goal</td>
<td>76</td>
</tr>
<tr>
<td>5.2.1.2 Questions</td>
<td>76</td>
</tr>
<tr>
<td>5.2.1.3 Metrics</td>
<td>76</td>
</tr>
<tr>
<td>5.2.2 The Planning</td>
<td>78</td>
</tr>
<tr>
<td>5.2.2.1 Context Selection</td>
<td>79</td>
</tr>
<tr>
<td>5.2.2.2 Hipotesis Formulation</td>
<td>79</td>
</tr>
<tr>
<td>5.2.2.3 Variables Selection</td>
<td>80</td>
</tr>
<tr>
<td>5.2.2.4 Selection of Subjects</td>
<td>80</td>
</tr>
</tbody>
</table>
# List of Figures

1.1 RiSE Labs Influences ................................................. 3
1.2 RiSE Labs Projects .................................................. 4

2.1 Essential Activities for Software Product Lines (Clements and Northrop, 2001) .................................................. 10
2.2 Requirements Engineering cyclic nature (adapted from (Sommerville, 2005)) .................................................. 12

4.1 Overview of the RiPLE-RE ........................................... 36
4.2 Elicitation fluxogram ................................................. 40
4.3 Overview of the Activity Model Scope ............................. 42
4.4 Variation Point in Feature Model ................................... 44
4.5 Representation example of feature model ......................... 45
4.6 Feature model legend ............................................... 46
4.7 Overview of the Activity Define Requirements .................. 47
4.8 XML Template ....................................................... 53
4.9 Example of a functional requirement with implication among VPs. 54
4.10 Example of a functional requirement with implication between a variant and VP ............................................... 55
4.11 Non-Functional Requirement with implication among requirements. 55
4.12 Overview of the Activity Define Use Cases ..................... 56
4.13 Use Case with small variations .................................... 60
4.14 Base Use Case with complex variation represented by extension point. 61
4.15 Extension use case (it extends base use case) .................. 61
4.16 Abstract use case .................................................. 62
4.17 Concrete Use Case with Include relationship ................... 63
4.18 Traceability metamodel ............................................. 69

5.1 Difficulties Histogram .............................................. 86
5.2 Frequency of the subjects’ satisfaction ............................ 87
5.3 Error Distribution Histogram ...................................... 88

B.1 Domain Requirements Template ................................. 107
B.2 Domain Use Cases Template ................................. 108
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Review Summary</td>
<td>30</td>
</tr>
<tr>
<td>4.1</td>
<td>Summary of the RiPLE-RE Process</td>
<td>38</td>
</tr>
<tr>
<td>4.2</td>
<td>List of possible information sources (based on (Kang et al., 1990; Chastek et al., 2001))</td>
<td>39</td>
</tr>
<tr>
<td>4.3</td>
<td>Features attributes</td>
<td>46</td>
</tr>
<tr>
<td>4.4</td>
<td>Non-Functional Requirements Types</td>
<td>49</td>
</tr>
<tr>
<td>4.5</td>
<td>Requirements attributes</td>
<td>50</td>
</tr>
<tr>
<td>4.6</td>
<td>Requirement Variation Point attributes</td>
<td>52</td>
</tr>
<tr>
<td>4.7</td>
<td>Use Cases attributes</td>
<td>58</td>
</tr>
<tr>
<td>4.8</td>
<td>Comparison with other approaches</td>
<td>73</td>
</tr>
<tr>
<td>5.1</td>
<td>Subject’s Profile in the Experimental Study</td>
<td>85</td>
</tr>
<tr>
<td>5.2</td>
<td>Results for Defect Index.</td>
<td>89</td>
</tr>
<tr>
<td>5.3</td>
<td>Effort in the use of the process.</td>
<td>89</td>
</tr>
<tr>
<td>5.4</td>
<td>Hypotheses Analysis.</td>
<td>91</td>
</tr>
<tr>
<td>C.1</td>
<td>Quality Criteria</td>
<td>110</td>
</tr>
<tr>
<td>C.2</td>
<td>Quality Criteria Results</td>
<td>111</td>
</tr>
</tbody>
</table>
Software reuse is an important aspect to minimize costs and time-to-market, and maximize quality and productivity (Northrop, 2002). However, these benefits are not assured by the application of ad-hoc reuse, which is an opportunistic reuse, not systematic, and generally restricted to source code.

One of the approaches towards the systematic and predictable way to achieve reuse is the software product lines approach (Pohl et al., 2005). In this context, product lines requirements are essential for identifying and analyzing opportunities for large-grained reuse, composing a common set of core assets. Thus, the focus of this dissertation is in providing a requirements engineering process for software product lines, in order to maximize the benefits of systematic reuse.

This chapter contextualizes the focus of this dissertation and starts by presenting its motivation in Section 1.1 and the goal of this work in Section 1.2. A brief overview of the proposed solution is presented in Section 1.3, while Section 1.4 describes some related aspects that are not directly addressed by this work. Section 1.5 presents the main contributions and, finally, Section 1.6 outlines the structure of the remainder of this dissertation.

1.1 Motivation

During the last decade, several efforts were conducted to achieve effective ways of dealing with the software industry competitive needs. The competitiveness increases according to industry ability to quickly react to market changes and users requirements (Atkinson et al., 2002). On the other hand, some studies show the quality improvement and time to market reduction in software development when a Software Product Lines (SPL) approach is adopted (Mili et al., 2001; Atkinson et al., 2002; Pohl et al., 2005). Thus it has showed to
be an effective solution to deal with such competitive needs, in specific domains, enabling the reuse of requirements, design, code, and other assets.

On the other hand, developing a SPL requires time and systematic planning to present positive results, otherwise, the investment can be lost due to failures in the project (Chastek et al., 2001; Clements and Northrop, 2001). In this context, a Requirements Engineering (RE) process is important to reduce the risks involved in a SPL, providing correct requirements development and management. Without it, the complexity can hardly be mastered (Fricker and Stoiber, 2008).

Single-system requirements have been repeatedly recognized during the last decades to be a real problem (van Lamsweerde, 2000). Their activities normally involve many stakeholders with different needs, which can result in conflicts. A study by The Standish Group (Group, 1995) showed that incomplete and inconsistent requirements are some factors for deficiency and cancellation of software projects. One analysis of the potential return on investment from better requirements suggests that requirement errors can consume between 70 and 85 percent of all project rework costs (Leffingwell, 1997).

In the SPL context, the RE activities are more complex, because they involve more products and stakeholders, and require attention to variabilities and commonalities (Clements and Northrop, 2001; Birk et al., 2003b; Thurimella and Bruegge, 2007). Thus, the SPL development should be supported by an appropriate RE process for its context, which assures usability, effectiveness and efficiency in its activities. Nevertheless, the available RE approaches for SPL present some gaps. In general, they do not present a complete and systematic process, with sufficient details to cover the whole RE lifecycle, as will be discussed in Chapter 3.

1.2 Goal of the Dissertation

Motivated by the scenario presented in the previous Section, the goal of the work described in this dissertation can be stated as:

This work defines a requirements engineering process by providing activities, tasks, inputs, outputs, roles and guidelines for correct requirements development and management for the software product lines context in an usable, effective and efficient way.
1.3 Overview of the Proposed Solution

In order to accomplish the goal of this dissertation, RiPLE-RE (RiSE Product Line Engineering Process - Requirements Engineering) is proposed.

This Section presents the context where it is regarded and outlines the proposed solution.

1.3.1 Context

This dissertation is part of the RiSE Labs\(^1\) (Almeida et al., 2004), formerly called RiSE Project, whose goal is to develop a robust framework for software reuse in order to enable the adoption of a reuse program. RiSE Labs is influenced by a series of areas, such as software measurement, architecture, quality, environments and tools, and so on, in order to achieve its goal. The influence areas are depicted in Figure 1.1.

![Figure 1.1 RiSE Labs Influences](image)

Based on these areas, the RiSE Labs is divided in several different projects related to software reuse, as shown in Figure 1.2:

- **RiSE Framework**: It involves reuse processes (Almeida et al., 2005), component certification (Alvaro et al., 2006) and reuse adoption and adaptation processes (Garcia et al., 2008).

\(^1\)http://www.rise.com.br/research.
• **RiSE Tools**: Research focused on software reuse tools, such as the Admire Environment (Mascena et al., 2006), the Basic Asset Retrieval Tool (B.A.R.T) (Santos et al., 2006), which was enhanced with folksonomy mechanisms (Vanderlei et al., 2007), semantic layer (Durao, 2008), facets (Mendes, 2008) and data mining (Martins et al., 2008), the Legacy InFormation retrieval Tool (LIFT) (Brito et al., 2008), the Reuse Repository System (CORE) (Melo et al., 2008), and the Tool for Domain Analysis (ToolIDay) (Lisboa, 2008).

• **RiPLE**: Stands for RiSE Product Lines Engineering Process and aims at developing a methodology for Software Product Lines, composed of scoping, requirements engineering, design, implementation, test, and evolution management.

• **SOPL**: Development of a methodology for Software Product Lines based on services, with some idea of RiPLE.

• **MATRIX**: Investigates the area of measurement in reuse and its impact in quality and productivity.

• **BTT**: Research focused on tools for detection of duplicated change requests (Cavalcanti, 2009).

• **Exploratory Research**: Investigates new research directions in software engineering and its impact on reuse.

• **CX-Ray**: Focused on understanding with empirical data the C.E.S.A.R \(^2\), its processes and practices in software development, including reuse.

This dissertation is part of the RiPLE project and its goal is to support the requirements engineering in the software product lines context.

### 1.3.2 Proposal Outline

The goal of this dissertation is to develop and manage domain requirements in software product lines, by defining a systematic process composed by three main activities: *Model Scope, Define Requirements* and *Define Use Cases*, integrated in a macroflow that guides all requirements engineering activities. According to the SPL lifecycle, its main area is the core assets development, which involves to create reusable assets for reuse.

---

\(^2\)C.E.S.A.R - Recife Center for Advanced Studies and Systems - http://www.cesar.org.br
It is important to highlight that this proposed process does not exclude existing requirements engineering techniques and methods, but comes to complement traditional requirements engineering with three activities focused on the software product lines context.

1.4 Out of Scope

As the proposed process is part of a broader context (RiPLE), a set of related aspects will be left out of its scope. Thus, the following issues are not directly addressed by this work:

- **Product Development.** An important issue in a SPL process is to create individual products by reusing the core asset, i.e. performing the products development with reuse. However, this aspect can be as complex as core assets development, involving the definition of activities, inputs, outputs, guidelines and roles. Thus, it is out of the scope of this work, but it has direct relationship with this process.

- **Evolution Management.** The SPL evolution control is ensured by appropriate practices of changes management, including changes in requirements. These practices are part of the Evolution Management process, thus they will not be considered in this process. They will be part of another process in RiPLE (Oliveira, 2009). However, this work provides guidelines to changes in requirements.

- **Evolution Metrics.** Measurement activities are essential in any engineering process. Both measurement activities inside the process and metrics to be used outside the process (to formally evaluate) could be incorporated to the process.
1.5 Statement of the Contribution

As a result of the work presented in this dissertation, a list of main contributions may be enumerated:

- A systematic review on requirements engineering for software product lines.

- The definition of a requirements engineering process for software product lines.

- The development of a tool to support the software product lines traceability.

- The definition, planning, analysis of an experimental study in order to evaluate the proposed process.

1.6 Organization of the Dissertation

The remainder of this dissertation is organized as follows:

- Chapter 2 discusses the software product lines basic concepts and activities, as well as requirements engineering. The relation between software product lines and requirements is also discussed.

- Chapter 3 presents a systematic review on requirements engineering approaches in the context of software product lines with the objective of characterizing them to better understand the state-of-the-art in this area.

- Chapter 4 describes the proposed requirements engineering process in the SPL context, presenting the roles associated, activities, guidelines, inputs and outputs and the key concepts of the process.

- Chapter 5 presents the definition, planning, operation, analysis and interpretation and packaging of the experimental study which evaluates the use of the proposed process.

- Chapter 6 concludes this dissertation by summarizing the findings and proposing future enhancements to the solution.

- Appendix A describes the questionnaires applied in the experimental study.

- Appendix B presents the templates proposed by this work to support RiPLE-RE.
Appendix C presents the quality criteria of the systematic review. It also presents the evaluation results of these criteria.
Software reuse is already a consolidate idea to decreasing development costs and improving quality (Northrop, 2002). However, past reuse agendas, which focused on reusing relatively small pieces of code or opportunistically cloning code designed for one system for use in another, have not been profitable (Northrop, 2002). The experience from reuse projects in the 1990s shows that without proper planning, the costs for reuse may be higher than for developing the assets from scratch (Pohl et al., 2005).

On the other hand, the Software Product Lines (SPL) approach has emerged as a software reuse trend, in which the reuse is planned, enabled, and enforced (Northrop, 2002). It uses a strategy that plans the use of assets in multiple products rather than ad-hoc approaches that reuse assets only if they happen to be suitable (McGregor et al., 2002). Thus, SPL has proven to be the methodology for developing a diversity of software products and software-intensive systems at lower costs, in shorter time, and with higher quality (Pohl et al., 2005).

In this Chapter will be introduced the principles of software product lines, emphasizing the requirements engineering within its context.

## 2.1 Software Product Lines

A product line is a set of products that together address a particular market segment or fulfill a particular mission (Clements and Northrop, 2001). It is a emergent concept in software. However, it is not new in other areas, such as manufacturing, aeroespacial, automotive and electronic components industries. Some companies out of the “software world” develop product lines, such as Boeing, Ford, Dell, and McDonald’s (Northrop,
2.1. SOFTWARE PRODUCT LINES

2002). Each of these companies exploits commonality in different ways. Boeing, for example, developed the 757 and 767 transports in tandem, and the parts list for these very two different aircraft overlap by about 60% (Clements and Northrop, 2001).

In the software context, product lines succeed because the commonalities shared by the software products can be exploited to achieve economies of production (Clements and Northrop, 2001). Historically, software engineers treated each new project as a single development effort that discarded the use of previous experiences and assets produced when building other software systems (Mili et al., 2001). Instead, software product lines view many systems within the context of similar systems built in the past, exploiting their commonalities and variabilities. Thus, substantial production economies can be achieved when the systems in a SPL are developed from a common set of assets in a prescribed way, in contrast to being developed separately, from scratch, or in an arbitrary fashion.

Software Product Line is defined as a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way (Clements and Northrop, 2001). In the literature, different terms are adopted to transmit essentially the same meaning. They might refer to a product line as a product family, to the core asset set as a platform, or to the products of the software product line as customizations instead of products. Core asset development might be referred as domain engineering, and product development as application engineering. In this work, the terms adopted are product line, core asset development and product development.

The identification of commonalities and variabilities is the key prerequisite for software product lines. Commonality reveals common characteristic for the products of the SPL. Variability is a tangible difference among the products of the SPL that can be revealed and distributed in any phase of development cycle of the product line. Thus, the variabilities are modeled to enable the development of customized products by reusing predefined and adjustable assets. The moment of variability resolution in realization is often called the binding time of the variability (Pohl et al., 2005).

Next, it is presented the strategies that can be adopted in a SPL approach.

### 2.1.1 SPL Strategies

There are three different adoption strategies for SPL: proactive, reactive and extractive (Krueger, 2002). In the proactive approach, the organization analyses, designs, and implements a SPL to support the full scope of products needed on the foreseeable horizon.
In the reactive approach, the organisation incrementally grows an existing SPL when the demand arises for new products or new requirements on existing products. In the extractive approach, the organization capitalizes on existing custom software systems by extracting the common and varying source code into a single production line.

In a proactive strategy, the products come to market extremely quickly with a minimum of code writing. However, it requires a significant up-front investment to produce the core assets base and a extensive domain knowledge. It is appropriate when the requirements for the set of products needed, extending to the far horizon, are well defined and stable. The proactive approach requires considerably more effort up front, but this drops sharply once the SPL is complete.

The reactive strategy has a lower cost of entry to SPL because the core asset base is not built up front. It is appropriate when the requirements for new products in the production line are somewhat unpredictable. However, for the SPL to be successful, the core asset base must be robust, extensible, and appropriate to future SPL needs. If the up front cost, time, and effort is prohibitive or if the risk of guessing wrong is high, the reactive approach is preferred over proactive.

With the extractive approach, the high level of software reuse enables an organization to very quickly adopt software mass customization. It is appropriate when an existing collection of customized systems can be reused.

In all strategies, the applicability of the SPL is advised to the domains where there is a great demand for specific products, but with a set of common characteristics and well-defined variabilities. A domain is a specialized body of knowledge, an area of expertise, or a collection of related functionality (Clements and Northrop, 2001).

The following section presents the essential activities of a SPL approach.

### 2.1.2 SPL Essential Activities

SPL involves three essential activities: **Core Asset Development**, **Product Development**, and **Management** (Clements and Northrop, 2001). Core Asset Development is responsible to the creation of the common assets of the SPL and the evolution of the assets in response to product feedback, new market needs, and so on. Core assets form the basis for the product line (Clements and Northrop, 2001). They often include assets in software development that are costly to develop from scratch, such as architecture, components, domain models, requirements, and tests. Product development creates individual products by reusing the core assets, gives feedback to core asset development, and evolves the products. Management includes technical and organizational management, where
technical management is responsible for the requirements control and the coordination between core asset and product development.

Figure 2.1 shows these three activities, which are intrinsically related to each other in a way that changing one of them implicates in analyzing the impact in the others. Each rotating circle represents one of the essential activities, which are linked together and in perpetual motion, showing that all three are essential, are inextricably linked, can occur in any order, and are highly iterative.

Software engineering practice areas, such as requirements engineering and design, provide appropriate techniques for the essential activities. Next sections will present more detail about the requirements engineering, which is the context of this work.

2.2 Requirements Engineering

Requirements are descriptions of what the system must implement, how it must behave, the constraints that the system or its development must satisfy, the properties and quality attributes it must possess (Kotonya and Sommerville, 1998). Their definition reside primarily in the problem space whereas other software assets reside primarily in the solution space (Cheng and Atlee, 2007). Thus, requirements problem space helps to de-
limit the solution space, providing directions to define the system boundaries, prioritizing requirements, negotiating resolutions to conflicts, and so on.

Requirements categorization is defined in a classic way as functional and non-functional. Functional requirements describe what the system must do and non-functional requirements place constraints on how these functional requirements are implemented (Sommerville, 2005).

Requirements Engineering (RE) is the process by which the requirements are determined, providing appropriate mechanisms for the activities of its lifecycle. The “engineering” term implicates that systematic and repetitive techniques should be used for ensuring complete, consistent and relevant requirements (Kotonya and Sommerville, 1998). On the other hand, a process is defined as an organized set of activities that transforms inputs to outputs (Kotonya and Sommerville, 1998). Thus, the complete description of a RE process should include what activities are carried out, the structuring or schedule of these activities, who is responsible for each activity and the tools used to support the RE activities.

The RE lifecycle encompasses requirements elicitation, analysis, negotiation, specification, verification, and management, where (Clements and Northrop, 2001; Sommerville, 2005):

- **Elicitation** identifies sources of requirements information and discovers the users’ needs and constraints for the system.
- **Analysis** understands the requirements, their overlaps, and their conflicts.
- **Negotiation** reaches agreement to satisfy all stakeholders, solving conflicts that are identified.
- **Specification** documents the user’s needs and constraints clearly and precisely.
- **Verification** checks if the requirements are complete, correct, consistent, and clear.
- **Management** controls the requirements changes that will inevitably arise.

The Requirements Engineering process has a cyclic nature, in which its standard disciplines are performed in an iterative mode, as can be seen in Figure 2.2.

The following section describes the peculiar aspects about Requirements Engineering for SPL.
2.3 Requirements Engineering for SPL

SPL requirements constitute an important core asset. They are collected in reusable models, in which commonalities and variabilities are documented explicitly. Thus, these requirements can be instantiated and adapted to derive the requirements for an individual product (Cheng and Atlee, 2007). During product derivation, for each variant asset, it is decided whether the asset is (or is not) supported by the product to be built. When a domain requirement is instantiated, it can become a concrete product requirement. Thus, new products in the SPL will be much simpler to specify, because the requirements are reused and tailored (Clements and Northrop, 2001).

Deciding which products to build depends on business goals, market trends, technological feasibility, and so on. On the other hand, there are many sources of information to be considered and many tradeoffs to be made. The product line requirements must be general enough to support reasoning about the scope of the SPL, predicting future changes in requirements, and anticipated SPL growth.

In practice, establishing the requirements for a SPL is an iterative and incremental effort, covering multiple requirements sources with many feedback loops and validation activities (Chastek et al., 2001). Thus, RE in SPL has an additional cost. Many SPL requirements are complex, interlinked, and divided into common and product-specific requirements (Birk et al., 2003a). Regarding to single systems, RE for SPL has some differences, such as (Clements and Northrop, 2001; Pohl et al., 2005; Thurimella and
Bruegge, 2007):

- **Elicitation** captures anticipated variations over the foreseeable lifecycle of the SPL. RE must anticipate prospective changes in requirements, such as laws, standards, technology changes, and market needs for future applications. Its information sources are probably larger than for single-system requirements elicitation.

- **Analysis** identifies variations and commonalities, and discovers opportunity for reuse.

- **Negotiation** solves conflicts not only from a logical viewpoint, but also taking into consideration economical and market issues. The SPL requirements may require sophisticated analysis and intense negotiation to agree on both common requirements and variation points that are acceptable for all the systems.

- **Specification** documents a product-line-wide set of requirements. Notations are used to represent the product line variabilities and enable the product instantiation;

- **Verification** checks if the SPL requirements can be instantiated for the products, ensuring the reusability of the requirements.

- **Management** must provide a formal mechanism for proposing changes, evaluating how the proposed changes will impact the SPL, specifically its core asset base. Evolution can affect the reuse and customization, therefore appropriate mechanisms must be used to manage the variabilities.

The RE process inevitably varies from one organization to another, but the inputs and outputs are similar in most cases (Kotonya and Sommerville, 1998). Some factors contribute to the variability of a RE process for SPL, such as organizational context, market orientation, product type, domain maturity and stability, information source, business constraints, and so on (Bayer et al., 1999; Birk et al., 2003b).

In SPL, RE also has influence of several stakeholders that participate of the product line. Stakeholders are people with a vested interest in the SPL. Identifying stakeholders that directly influence the RE is essential to define the requirements negotiation participants. They are responsible by resolving conflicts and providing information.

Each stakeholder plays a role with respect to the product line. Many of the stakeholders that help to define the requirements also use them. These users have different expectations of the outputs of product line analysis. Some may simply want to confirm
that their interests have been represented (e.g. marketers, domain expert and analyst domain). Others (e.g., architects and developers) may want to describe proposed functional and non-functional capabilities, and their commonality and variability across the product line, so that decisions about architectural solutions and asset construction can be made (Chastek et al., 2001). The requirements analyst has a central communication role, being responsible for interacting with all these stakeholders.

In general, the RE decisions can be justified by the product line context, i.e., how concrete characteristics of this context guide the process. Examples of decisions are which strategies are used, how the process is performed (e.g. lifecycle), and how stakeholders are involved. The different contexts that can influence the RE decisions are classified according to categories, as follows:

- **Starting situation:** A SPL can start from the scratch, it can be introduced while some products are already under development, or its core assets can be re-engineered from legacy systems (Bayer et al., 1999). Whether the SPL is introduced from existing systems or products in development, the new requirements can be influenced by constraints of these assets. The starting situation also influences the elicitation strategy to be adopted.

- **Market orientation:** A SPL can be development for a specific market segment without a concrete customer in mind, or for individual customer projects (Birk et al., 2003b). This situation influences the identification of the stakeholders involved in RE, and also the elicitation strategy to be adopted. For example, whether the SPL is oriented for a market segment, potential information source can be marketers and market analysis. But whether the SPL is oriented for a specific customer, he is a potential stakeholder for elicitation.

- **Domain maturity:** The domain exists already for quite some time and they are well understood, or the domain is new. Thus, it directly influences the problem understanding, so it can be essential for decisions in relation to the detail level of the requirements specification.

- **Domain stability:** Domain is not expected to change in the near feature, or the domain changes with frequency. This situation must be analyzed to negotiate requirements and identify best strategies for changes management. Whether the domain is not stable, it requires a SPL evolution management more systematic.
• **Business constraints:** Business constraints (e.g. time to market and resources) can influence the RE process. They can affect the process lifecycle, the prioritizing and selecting requirements for releases of the assets by exploring requirements refinement choices and evaluating requirements pertinence (Fricker and Stoiber, 2008). Projects with short time can adopt a RE process more agile and iterative, and the requirements can be specified incrementally according to its releases.

• **Organizational context:** The organizational structure and behavior can influence the RE process, such as team size, organizational structure and available stakeholders. In the context of small organizations is recommendable an approach more agile, in which requirements are specified incrementally and are detailed only use cases more complex.

• **Geographical context:** The geographical separation can influence the RE process, since different means for communication and coordination are used, depending on the distance among collaborators (Fricker and Stoiber, 2008). It influences, for example, the language used for documentation and the manner of storing requirements (repository). In case of distributed project, the requirement management must to allow access to all requirements from any location.

• **Information source:** Different requirements sources can exist in the SPL context. The information sources identification is essential for the requirements elicitation. Thus, the elicitation strategies must be adapted according to the type of information source, such as existing assets and systems, stakeholders, textbooks, and so on.

### 2.3.1 Risks and Challenges

A key RE challenge for SPL development includes strategic and effective techniques for analyzing domains, identifying opportunities for SPL, and identifying the commonalities and variabilities of a product line (Cheng and Atlee, 2007). Another challenge related to RE is that the applicability of more formal techniques and tools is limited, partly because such techniques are not yet designed to cope with SPL development’s inherent complexities (Birk et al., 2003a).

Regarding to the risks associated with RE for SPL, the major risk is failure to capture the right requirements over the life of the product line (Clements and Northrop, 2001). Documenting the wrong or inappropriate requirements, failing to keep the requirements up-to-date, or failing to document the requirements at all, puts the architect and the
component developers at a grave disadvantage. They will be unable to produce systems
that satisfy the customers and fulfill the market expectations. Moreover, inappropriate
requirements can result from the following (Clements and Northrop, 2001):

- **Failure to tell product-line-wide requirements from product-specific requirements.**
The core asset builders need to know the requirements they must build, while the
product-specific software builders must know what is expected of them.

- **Insufficient generality.** Insufficient generality in the requirements leads to a design
that is too brittle to deal with the change actually experienced over the lifecycle of
the SPL.

- **Excessive generality.** Overly general requirements lead to excessive effort in
producing both core assets (to provide that generality) and specific products (which
must turn that generality into a specific instantiation).

- **Wrong variation points.** Incorrect determination of the variation points results
in inflexible products and the inability to respond rapidly to customer needs and
market shifts.

- **Failure to account for qualities other than behavior.** Product line requirements
(and software requirements in general) should capture requirements for quality
attributes such as performance, reliability, and security.

## 2.4 Chapter Summary

This chapter presented the concepts about software product lines and their aspects related
to requirements engineering, which is the context of this dissertation. One of the key
properties in software product lines is to manage commonalities and variabilities. Thus,
in RE, it is the main aspect that differs SPL and single systems.

Next Chapter presents a systematic review in requirements engineering for software
product lines, discussing current approaches, their weaknesses and strengths, in order to
define a base to the proposal of this work.
A Systematic Review on Requirements Engineering for Software Product Lines

Compared to single systems development, the requirements engineering for SPL becomes more complex and challenging, since it involves more risks and requires attention for commonalities and variabilities. In order to address part of these challenges, some approaches have been proposed by academy and industry. This systematic review is an attempt to identify the available approaches, and verify how they deal with RE for SPL. This systematic review contributes to understand the state-of-the-art in the area, providing insights to help choosing among the existing approaches, and pointing out directions for future work.

This Chapter is organized as follows. Section 3.1 introduces the systematic literature reviews and presents the purpose of this study. Section 3.2 presents the planning phase, which defines the need and the protocol used for conducting the review. Section 3.3 discusses the conduction procedures, along with the search strategy used and the approaches selected to be considered in this review. Section 3.4 presents the reporting phase, which discusses the results obtained in the review. Section 3.5 presents the threats to validity, Section 3.6 describes the related work, and, finally, Section 3.7 summarizes this Chapter.

3.1 Introduction

Systematic Review (SR) refers to a specific methodology of research, developed in order to gather and evaluate the available evidence related to a research question or topic. There are several important reasons for undertaking a systematic literature review, it can be used to summarize the existing evidences concerning a technology, identify gaps in research areas, suggest issues for further investigation and examine empirical
evidence to support or contradicts hypotheses Kitchenham and Charters (2007). The well defined methodology makes it less likely that the results are biased, and provides a greater scientific value than conventional literature reviews Kitchenham and Charters (2007).

Some other features differentiate systematic reviews from conventional literature reviews, such as the definition of a review protocol that specifies the research question being addressed, the search strategy documentation in order to other readers assess their rigor, completeness and repeatability. Systematic reviews also requires inclusion and exclusion criteria as an evaluation form of each primary study.

The review presented in this chapter is systematically performed, following Kitchenham’s guideline Kitchenham and Charters (2007), which aids in assuring consistent results from individual studies (called primary studies). According to Kitchenham and Charters (2007), the SR is divided in three phases, which are linked with the paper structure as follows.

- **Planning**: its goal is developing a protocol that specifies the plan that the systematic review will follow to identify, assess, and collect evidence;

- **Conducting**: responsible for executing the protocol planned in the previous phase; and

- **Reporting**: its purpose is relating the review steps to the community and it is fulfilled with this review report.

Based on this definition, each one of the main phases is further described in the following sections.

### 3.2 Planning

This section presents the stages associated with the initial phase of this SR, which includes the protocol definition and the research questions specification and structure.

#### 3.2.1 Protocol Definition

The SR protocol aims at delineating the research objectives and clearly explaining the review execution process, through defining research questions and planning on how the sources and studies selection will be accomplished. The research questions will guide the design of the review process.
Regarding to the roles in this work, there are one research (M.Sc. candidate) and four reviewer (three M.Sc. candidates and one PH.D.). The research built the protocol, conducted the search for primary studies and further analysis. The reviewers oversaw the activity, inspecting the protocol, the search and analysis tasks since they have expertise in this domain and could give important contribution.

### 3.2.2 Research Question

In this context, the purpose of this study is to review the Software Product Lines approaches to understand, characterize and summarize evidence about Requirements Engineering, identifying their activities and important practical and research issues in the area.

Thus, this review is intended to answer the following research questions:

**Q1. Which requirements engineering activities are adopted by the SPL approaches?** This question aims identifying the RE activities in the SPL approaches according to disciplines (elicitation, specification, analysis, verification and management) which guide a RE process (Kotonya and Sommerville, 1998; Clements and Northrop, 2001).

**Q2. Which models and techniques are used by the SPL approaches in the RE?** This question aims identifying which models and techniques, the SPL approaches use for requirements engineering. Thus, the goal of this question is to discover the solutions trends and differences adopted by the approaches.

**Q3. How do the approaches deal with variability?** The purpose of this question is to characterize how the SPL approaches deal with variability in RE. Depending on how an approach deals with variability, it can be easy or not for an analyst to configure and instantiate a requirement for an application.

The research questions are discussed from three viewpoints:

**Population.** It defines an application area or people affected by the intervention. Thus, the Software Product Lines approaches define the population of this study.

**Intervention.** This review focused on analyzing Requirements Engineering practices.

**Outcomes.** This review outcomes the need for a complete and well defined process to conduct requirements engineering activities during the development of software product lines.
3.3  Conducting

The steps referred to how the review was conducted are detailed in this section. It involves the search strategy, data sources, studies selection and data analysis and synthesis.

3.3.1  Search Strategy

The primary studies were searched by keywords. We used various combinations of search items to achieve a more adequate set of keywords, they are: software product line, software product family, product line, application family, production line, product population, product family, domain analysis, domain engineering, requirements engineering, domain, requirement, variability, variation, domain requirement, domain requirements software reuse and reuse, as well as their syntactic variations (e.g. plural form).

Search strings are derived from research questions and are constructed using boolean ANDs and ORs to combine keywords. Thus, this review defines the following search strings:

• (requirements engineering OR requirement OR requirements OR domain analysis) AND (software product line OR software product lines OR software product family OR product family OR software product families OR application family OR application families OR product line OR product lines OR production line OR product population OR product family OR product families OR domain engineering OR software reuse OR reuse)

• (requirement engineering OR requirement OR requirements OR domain) AND (variability OR variabilities OR variation)

• (software AND (domain requirement OR domain requirements))

3.3.2  Data Sources

The search for primary studies was performed in web search engines and digital libraries of the most famous publishers and organizations in software engineering. In these searches, we tried to find journals, conferences proceedings, books, theses and technical reports, and also analyzed the referenced studies. The used methodology is explained next:

1. Since usually the most important work are published on journals (from IEEE, ACM and Springer), they were the first place searched. The research was done through
search terms matching at their web site and/or access portals. The searched journals were: IEEE Software; ACM Transactions on Software Engineering and Methodology; IEEE Computer; IEEE ACM Sigsoft SW Engineering Notes; Communications of the ACM; Journal of Systems and Software; and Requirements Engineering Journal.

2. After that, the papers were searched in important conferences proceedings (IEEE and ACM). The research methodology was the same as the one used for the journals. The searched conferences were: International Conference on Software Engineering (ICSE); Annual ACM Symposium on Applied Computing (SAC); International Conference on Aspect-Oriented Software Development (AOSD); International Conference on Advanced Information Systems Engineering (CAiSE); International Conference on Enterprise Information Systems (ICEIS); Euromicro Conference on Software Engineering and Advanced Applications (SEAA); International Conference on Model Driven Engineering Languages and Systems (MODELS); International Conference on Composition-Based Software Systems (ICCBSS); Fundamental Approaches to Software Engineering (FASE); International Conference on Software Reuse (ICSR); Software Product Line Conference (SPLC); European Software Engineering Conference (ESEC); International Computer Software and Applications Conference (COMPSAC); International Conference on Software Engineering and Knowledge Engineering (SEKE); Empirical Software Engineering and Measurement (ESEM); and International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA).

3. As next step, papers referenced by the authors of the found papers were also analyzed.

4. Moreover, we analyzed the grey literature, such as technical reports from different research groups.

5. After the analysis of the papers, it was possible to identify some researchers working with this topic. Therefore, we could contact them directly to ask further questions.

3.3.3 Studies Selection

As soon as the potentially relevant primary studies have been obtained, according to the match of the search items and research questions, they needed to be assessed for their
actual relevance. To achieve that, the criteria for inclusion and exclusion of the objects in this review were defined. For each selected primary study was applied the reading of the following elements: titles, abstracts, keywords, conclusion and references. As studies had been read, if an element satisfy any exclusion criteria or the set of elements do not suit to any inclusion criteria, the selected study was rejected. The criteria are next detailed.

**Inclusion Criteria** establish the inclusion reasons for each study found during the search. Studies which present at least one of the following topics will be included:

- Requirements Engineering for Software Product Lines. In this topic can be included studies related with one or more RE parts (e.g. elicitation, analysis, traceability, management, negotiation, verification, specification).

- Domain Analysis. It is directly related with requirements engineering for SPL.

- Software Product Lines Variability. Variability is the key concept for SPL modeling, so it is important analyzing studies related with variability on requirements.

**Exclusion Criteria** describe the following reasons to discard studies:

- Approaches with insufficient information in RE. Studies that do not have usable information about their RE activities.

- Study is not an approach. It treats only concept or issues related with requirements engineering for software product lines.

- Duplicate studies. When a study has been published in more than one publication, the most complete version will be used.

### 3.3.4 Quality Assessment

In addition to general inclusion/exclusion criteria, it is considered critical to assess the “quality” of primary studies (Kitchenham and Charters, 2007). In this context, quality criteria application improves the studies evaluation, serving as a means of weighting the importance of individual studies and guiding their understanding, and also bringing forward even more confidence about the analysis.

Selected studies by inclusion/exclusion criteria are evaluated in next according to 7 quality criteria. Taken together, these 7 criteria provided a measure of the extent to which we could be confident that a particular study could present a valuable contribution to the review. We did not use the result to serve a threshold to inclusion decision, but rather to identify the ones which can be better considered more relevant to our study.
The quality assessment criteria for these studies is presented in Appendix C.

### 3.3.5 Data Collection

The data extraction forms must be designed to collect all the information needed to address the research questions and the study quality criteria.

For each study, we extracted the following data:

- The source (conference, journal, and so on.). The additional publications to understand the study were grouped.
- The year when the study was published (year of the first publication).
- Scope (requirements engineering, elicitation, specification, management, analysis, negotiation and verification).
- Research question answers and quality criteria analysis.
- Summary (a brief analysis, overview of its weaknesses and strengths).
- Quality criteria score.
- Reviewer’s name.
- Date of the review.

### 3.3.6 Data Analysis and Synthesis

The primary studies were analyzed and synthesized. The data was tabulated in studies chronological order, so we analyzed possible associations with study age and outcome. The tabulation was grouped according to research questions and it was reviewed to answer the research questions and identify any interesting trends or limitations in current research. Quality criteria were used as part of the data analysis.

The first stage resulted the set of 111 studies raised from the web search, through the use of search strategy presented in Section 3.3.1 applied to the data sources, as showed in Section 3.3.2. Studies with titles indicating clearly that they were outside the scope of this SR were discarded. Nevertheless, titles are not always clear indicators of what a study is about. The likelihood of finding not so relevant studies was considered, therefore in a next stage inclusion and exclusion criteria were applied to them, which resulted in 61 studies. It basically comprised a brief analysis of abstract and conclusion. This strategy
is claimed to be useful for giving the researcher more detailed information on the subject. In the next stage, the whole text must be visited in order to have a critical viewpoint on the topic addressed by the study. At the end of this stage, we discarded a total of 45 studies. Thus, remained 16 selected studies, which represent the primary studies to be analyzed and evaluated in this SR.

Each of the 16 approaches (primary studies) was assessed independently by the author, according to research questions and quality criteria. A brief description about the selected approaches is presented in chronologic order of publication, as follows:

**FODA.** Feature-oriented Domain Analysis (FODA) (Kang et al., 1990) is a method for domain analysis that focus on identifying variability and commonality in feature model. It was published by the Software Engineering Institute (SEI) in 90s. Since, this proposal for domain modeling has being adopted in several approaches for SPL (e.g. (Kang et al., 1998; Griss et al., 1998; Chastek et al., 2001; Pohl et al., 2005)), in which are defined variants, improvement or integration with other SPL methods.

**FAST.** Family-oriented Abstraction, Specification, and Translation (FAST) (Gupta et al., 1997; Weiss, 1997) is a SPL process that uses Domain Specific Language (DSL) for domain requirements specification, defining abstractions for representing them, and translating of the products specifications into deliverable software.

**FeatuRSEB.** This approach (Griss et al., 1998) integrates FODA (Kang et al., 1990) and Reuse-Driven Software Engineering Business (RSEB), in which the feature model is derived from the domain use case model.

**FORM.** Feature-Oriented Reuse Method (FORM) (Kang et al., 1998) is an extension from FODA. Its main characteristic is the four-layer decomposition, which describes different viewpoints on SPL development, defining the architectural framework according to the feature model layers.

**VODRD** (Mannion et al., 1998). Viewpoint-Oriented Domain Requirements Definition (VODRD) is an iterative method for analysis of domain requirements from stakeholders’ perspectives, in which a set of viewpoints are defined. Once all stakeholders agree on these viewpoints, the requirements specification can be started.

**Odyssey.** This study (Braga et al., 1999) defines a domain oriented software development framework, in which its main objective is to create components that can be reused in domain application.

**PuLSE-CDA.** The Customizable Domain Analysis (CDA) method (Bayer et al., 2000) is a subset of the PuLSE methodology (Bayer et al., 1999). This method was

---

1. [http://www.sei.cmu.edu](http://www.sei.cmu.edu)
developed to be adapted to the needs of each project, envisioning the project scope
evolution and the consequent change in the requirements.

**DH.** This study (Kuusela and Savolainen, 2000) describes a Definition Hierarchy
(DH) method. Its proposal is a goal-based approach, in which the domain requirements
are aligned to the SPL architectural drivers. Requirements are structured into a definition
tree where detailed requirements are children of a major design goal.

**Goal-oriented.** This method (Mylopoulos et al., 2001; Liaskos et al., 2006) presents
a goal-oriented approach, representing goals in a AND/OR decomposition tree. Its
variability frames enables describing a set of domain elements (e.g.: actors and conditions)
for representing the scenario involved in each goal.

**PLA.** Chastek et al. (Chastek et al., 2001) present the Product Line Analysis (PLA)
approach, which describes two modeling strategies: a feature-based strategy and an
use-case-based strategy.

**PRS.** Product-line Requirements Specification (PRS) (Faulk, 2001) is an approach
for formal domain requirements specification, based on class structuring mechanism.
The general strategy is to group together parts of the requirements that vary together and
encapsulate them in a class.

**DRAMA.** This approach (Park et al., 2004; Kim et al., 2006, 2008) proposes a
framework for Domain requirements Analysis and Modeling Architectures (DRAMA) in
SPL. It aims at identifying the domain requirements and providing the rationale for them
with goals and scenario modeling.

**PLUS.** Product Line UML-based Software engineering (PLUS) (Gomaa, 2005) is an
evolutionary and iterative process for SPL development, which uses the Unified Modeling
Language (UML) notation. Its lifecycle was strongly influenced by the Rational Unified
Process (RUP).

**PLUSS.** Product Line Use case modeling for Systems and Software engineering
(PLUSS) (Eriksson et al., 2005, 2006) is based on FeatureRSEB (Griss et al., 1998)
and combines use case modeling with feature modeling, defining links among them. It
also defines change cases, which are basically use cases that specify anticipated future
changes to a system. Once accepted for implementation in a product within the SPL,
these change cases are transformed to use cases.

**PR.** This approach (Moon and Chae, 2005) defines domain specification atoms. Each
atom is called Primitive Requirement (PR) and has a granularity less than use case. It is
defined as a transaction that has an effect on an external actor.

**SPLE.** Pohl et al. (Pohl et al., 2005) define a Software Product Line Engineering
(SPLE) framework, including RE activities. The key point of this framework is the Orthogonal Variability Model (OVM), which represents variability separated from the domain assets. It presents different views for representing the problem space.

More detail about these approaches is described in the following section, which reports the results of this review.

3.4 Reporting

In this section, we present the results of this review. Each sub-section presents the results of the research questions analysis. After the results, the review summary is presented.

3.4.1 Requirements Engineering Activities

According to Kotonya and Sommerville (Kotonya and Sommerville, 1998), a basic RE process should define activities for elicitation, analysis, negotiation, verification, specification and management of requirements. Thus, we analyzed which RE activities are defined in each approach, and how these activities are described based on the quality criteria:

- QC1. Are the roles described?
- QC2. Are the guidelines described?
- QC3. Are the inputs and outputs described?

Activities for requirements elicitation is common in majority of the approaches (except DH, Goal-oriented, PRS and PLUSS). However, the activities are partial and informally defined, with insufficient guidelines to gather the requirements. According to Clements and Northrop (Clements and Northrop, 2001), the major risk associated with RE is the failure to capture the right requirements over the life of the product line.

Regarding to requirements analysis and specification, all approaches define activities for them. Over the years, the SPL approaches focused their studies on the definition or adaptation of domain analysis methods, in which commonalities and variabilities are modeled.

From the requirement management perspective, few approaches define activities for it (Odyssey, PuLSE-CDA, PLA, PRS, PLUSS and Sple). The main gap is in the definition of mechanisms to manage the variabilities. The requirements for the SPL will evolve
over time (Bayer et al., 2000), so they should be managed in a systematic way, evaluating how the proposed changes will impact the SPL.

Requirements verification is found in VODRD, Odyssey, DH, Goal-oriented, PRS, DRAMA, PLUSS and PR. Requirements negotiation is explicitly defined only in SPL and PuLSE-CDA. However, it can be implicit in the practices for requirements elicitation and analysis (e.g. viewpoints-based).

3.4.2 Requirements Engineering Models and Techniques

Different models and techniques are presented by the RE approaches for SPL. They are presented as follows.

Feature Model. In this model, the requirements are organized by features, which represent end-user visible characteristic of a system (Kang et al., 1990). This representation shows an overview of the domain reuse that aids in the decision-make among features, being very used to generalize and parameterize other assets. On the other hand, this view has a high abstraction level, it does not describe the system behavior. Furthermore, its advantages depend of the domain size due to its hierarchical view, which is impracticable in a complex SPL with many features. The following approaches use this model: SPL, PLUSS, PLUS, PLA, Odyssey, FORM, FODA and FeatuRSEB.

Goal-Based. Goal-driven techniques and models are useful for eliciting and validating requirements, meeting the organization’s goals and needs, presenting the reasons for domain variabilities (Kim et al., 2003). However, they have an abstraction high-level and they do not describe the system behavior as in feature model. This strategy is adopted by DRAMA, Goal-oriented and DH approaches.

Use Case. Use case models are documented using scenarios and/or use case diagrams. They are user-oriented and addressed for specifying functional requirements (Griss et al., 1998; Gomaa, 2005). Its success in the software industry influenced the new use cases proposal driven to SPL, in which are adopted notations for representing variabilities. However, the use case model cannot represent neither non-functional requirements nor requirements without interaction. The following approaches adopt use cases in their RE strategy: SPL, PR, PLUSS, PLUS, PLA, PuLSE-CDA, Odyssey and FeatuRSEB.

Primitive Requirement. The Primitive Requirements (proposal of the PR approach) is a new direction for defining requirements as atomic elements. A PR is defined as a transaction that has an effect on an external actor, so it is identified as a unit of use case. A use case can be a relatively large end-to-end process description that typically includes many primitive requirements (Moon and Chae, 2005). This decomposition is useful for
linking variabilities. However, it has the same problem found in use cases, i.e. it cannot represent neither non-functional requirements nor requirements without interaction.

**Viewpoint.** Viewpoint-based techniques and models define different perspectives for the SPL requirements. Thus, they are efficient for requirements elicitation and negotiation, in which are identified different needs and conflicts. On the other hand, this strategy can be hard when there are many stakeholders involved. Too many viewpoints can create an unmanageable amount of information; too few can make comparing separate concerns difficult (Mannion et al., 1998). The difficulty in finding this balance is aggravated when the number and complexity of requirements increase. It is adopted by the VODRD method, which gathers stakeholder viewpoints to organize user requirements.

**Decision Model.** It describes variations as decision. The variabilities are connected to decisions that, when completely resolved, specify a member of the SPL (Bayer et al., 2000). PRS, FAST and PuLSE-CDA use this model as strategy for instantiating domain model.

**Orthogonal Variability Model.** The SPLE approach proposed a new model to connect variabilities in different domain assets, making easy the decisions for instantiating products of the SPL. It defines traceability links variants and variation points and the corresponding definitions of the variability in requirements assets (Pohl et al., 2005). However, this proposal is viable when there is a support tool to maintain the traceability between the assets and OVM.

**Change Case.** PLUSS defines change cases, which are basically use cases that specify anticipated future changes to a system. New requirements are first modeled as change cases, however, once accepted for implementation in a system within family, these changes are transformed to use cases (Eriksson et al., 2005).

**PRS class.** Product line Requirements Specification (PRS) is a formal model for documenting requirements for embedded-system product lines, used in the PRS approach. It defines modularization and encapsulation of requirements in class, which is useful for that parts of the specification describing variable requirements when can be changed without impacting other parts of the document (Faulk, 2001). However, its proposal is more complex than others due to formality of the model.

### 3.4.3 Variability

In this aspect, we analyzed how the variability representation enables product derivation and core assets configuration. The variability strategies found in the approaches are presented as follows.
In FODA and FORM, variability is modeled in feature model, in which AND/OR nodes are defined using notations for representing common and variable features. FeatureBSEB, PLUS, PLUSS, PLA and Odyssey define variabilities in feature model and use cases, defining the relation between them. However, in all these approaches, the variabilities among assets (e.g. domain models and components) are not directly linked, difficulting the decisions in the instantiation of the products.

For VODRD approach, commonalities and variabilities are persisted in domain dictionary and linked in viewpoints. However, it does not defines inclusion and exclusion dependency. In this approach, also there is not a variability model for integration of variabilities in different assets, linking decisions.

The PuLSE-CDA proposal is defining a decision model for each workproduct that captures variability. On top of the different decision models for the workproducts, a domain decision model is built that combines them. Thus, links are established that explicitly model interrelationships among information in the different workproducts. It introduces meta elements that indicate points of variation and enable the instantiation of the workproducts. The decision hierarchy reduces the number of decisions that must be made and supports the intellectual control of constraints and dependencies.

FAST and PRS also work with decision model. However, it does not define a strategy for combining decisions in different workproducts, as is defined in PuLSE-CDA.

In the DH approach, a hierarchical tree represents the decisions of the products in each edge. However, there are some gaps in this proposal. It represents external variability, but the internal variability is not represented. When the number of product is large its representation is little readable. It does not present also solution for traceability among decisions.

The Goal-oriented approach represents variability in stakeholder goals, through the notion of OR-decomposition of goals. In the DRAMA approach, the variability is represented in level of scenarios and the goal provides the rationale for the variations. Both approaches have a common problem, it does not define an integration model for decisions in domain. Moreover, DRAMA does not represent exclusion dependency in the decisions.

The PR approach defines commonality ratio for each primitive requirement, which indicates how much the requirement is used in the domain. Variability is defined in level of primitive requirement and use cases. Relationship between them is identified and captured in a matrix. However, it does not represent exclusion dependency and traceability among different decisions in whole SPL.
SPLE defines the Orthogonal Variability Model (OVM), which represents variability separated from the domain assets. Thus, the variabilities complexity is reduced by linking dependencies among variable assets, employing the constraints associations in the OVM.

### 3.4.4 Review Summary

In this section, the results of the review are summarized according to the data extracted from the questions. Table 3.1 presents a summary of these results.

Initially, we investigated which RE activities are adopted by the found SPL approaches. In general, only PuLSE-CDA and SPLE cover the whole RE. However, its activities are defined in an insufficient detail level: PuLSE-CDA does not define guidelines and roles in its activities; and SPLE defines partially the guidelines and it does not describe roles for the RE activities. However, these aspects are not sufficient to evaluate the approaches completeness. It is essencial to analyze the models and techniques proposed, and the strategies adopted by the approaches for dealing with variabilities, which is the main characteristic for SPL.

Regarding to the models and techniques, each ones has a specific purpose and it needs of others to a complete support to RE in SPL. A model alone is insufficient to guide a team in the development of a SPL. For example, some models are defined in high abstraction level, such as Feature Model and Goal-Based. Others present a detailed specification, such as Use Cases. The appropriate integration of different strategies, according to SPL context, is essential to attend to domain constraints and stakeholders’ needs. Thus, the approaches must provide multiple viewpoints for the stakeholders, representing functional and non-functional requirements, describing and linking commonalities and variabilities. Some approaches integrate different models, such as FeatuRSEB, Odyssey, PLA, PLUS, PLUSS and SPLE. SPLE is highlighted due to its multiple views for representing the SPL requirements, providing also a strategy for integrating the variabilities of the different assets.

Variability, as an inherent concept from SPL, is naturally addressed by many studies. All approaches are structured to define SPL commonalities and variabilities. However, several approaches (DRAMA, DH, FAST, Goal-oriented, Odyssey, PLA, PLUS, PLUS, PRS and VODRD) do not define strategies for linking decisions among different assets. Only PuLSE-CDA and SPLE present a proposal to solve it.

From this analysis, we conclude that there are some gaps in RE for SPL. In general, there is not a complete and systematic RE process for SPL. Therefore, due to the complex and evolutionary nature of SPL development, it is essential to have a systematic RE
### Table 3.1 Review Summary

<table>
<thead>
<tr>
<th>Approach</th>
<th>Activities</th>
<th>Disciplines</th>
<th>Does it define Roles?</th>
<th>Does it define Guidelines?</th>
<th>Does it define Inputs and Outputs?</th>
<th>Models and Techniques</th>
<th>Does it link decisions among different assets?</th>
</tr>
</thead>
<tbody>
<tr>
<td>FODA</td>
<td>Model Domain, Analyze Commonality</td>
<td>Specification, Analysis, Elicitation</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Feature Model</td>
<td>No</td>
</tr>
<tr>
<td>FAST</td>
<td>Define Decision Model, Analyze Commonality</td>
<td>Specification, Analysis, Elicitation, Verification</td>
<td>Yes</td>
<td>Yes</td>
<td>Partially</td>
<td>Decision Model</td>
<td>No</td>
</tr>
<tr>
<td>FeaturiSEB</td>
<td>Model Feature, Model Use Case</td>
<td>Specification, Analysis, Elicitation, Verification</td>
<td>No</td>
<td>Partially</td>
<td>Partially</td>
<td>Feature Model, Use Case</td>
<td>No</td>
</tr>
<tr>
<td>FORM</td>
<td>Identify Feature, Classify Feature, Validate Model</td>
<td>Specification, Analysis, Elicitation, Verification</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Feature Model</td>
<td>No</td>
</tr>
<tr>
<td>VODRD</td>
<td>Characterize the domain, Document the viewpoints, Analyze the viewpoints</td>
<td>Specification, Analysis, Elicitation</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Viewpoint</td>
<td>No</td>
</tr>
<tr>
<td>Odyssey</td>
<td>Refine scope definition, Elicit raw domain knowledge, Model domain knowledge</td>
<td>All</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Use Case, Decision Model</td>
<td>Yes</td>
</tr>
<tr>
<td>DH</td>
<td>Represent definition hierarchy, Model Goal</td>
<td>Specification, Analysis</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Goal-based</td>
<td>No</td>
</tr>
<tr>
<td>Goal-oriented</td>
<td>Model Goal</td>
<td>Specification, Analysis</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Goal-based</td>
<td>No</td>
</tr>
<tr>
<td>PLA</td>
<td>Model Feature, Model Use Case, Model Object, Verify Models</td>
<td>Specification, Analysis, Elicitation, Verification</td>
<td>Yes</td>
<td>Yes</td>
<td>Partially</td>
<td>Feature Model, Use Case</td>
<td>No</td>
</tr>
<tr>
<td>PRS</td>
<td>Organize the Domain Definition, Create a Decision Model, Encapsulate Variations in CoRE, Define variations, Provide Traceability</td>
<td>Specification, Management, Analysis</td>
<td>No</td>
<td>Partially</td>
<td>Partially</td>
<td>Decision Model, PRS class</td>
<td>No</td>
</tr>
<tr>
<td>DRAMA</td>
<td>Analyze Domain Requirement, Analyze Commonalities and Variabilities, Represent Domain Requirement</td>
<td>Elicitation, Analysis, Specification</td>
<td>No</td>
<td>Partially</td>
<td>Partially</td>
<td>Goal-based</td>
<td>No</td>
</tr>
<tr>
<td>PLUS</td>
<td>Model Requirements, Model Product Line Analysis, Test Product Line</td>
<td>Specification, Analysis, Elicitation, Verification</td>
<td>No</td>
<td>Yes</td>
<td>Partially</td>
<td>Feature Model, Use Case</td>
<td>No</td>
</tr>
<tr>
<td>PLUSS</td>
<td>Model Feature, Model Use Case, Manage Change</td>
<td>Specification, Analysis, Elicitation, Verification</td>
<td>No</td>
<td>Partially</td>
<td>Partially</td>
<td>Feature Model, Use Case, Change Case</td>
<td>No</td>
</tr>
<tr>
<td>PR</td>
<td>Identify domain requirements, Refine domain requirements, Model domain use case</td>
<td>Elicitation, Analysis, Specification</td>
<td>No</td>
<td>Yes</td>
<td>Partially</td>
<td>Use Case, Primitive Requirement</td>
<td>No</td>
</tr>
<tr>
<td>SPLiE</td>
<td>Analyze variability, Analyze Priority-based, Model Requirements, Define Traceability</td>
<td>All</td>
<td>No</td>
<td>Partially</td>
<td>Yes</td>
<td>Feature Model, Use Case, Orthogonal Variability Model</td>
<td>Yes</td>
</tr>
</tbody>
</table>
process.
The results of the quality criteria evaluation, they can be found in Appendix C.

3.5 Threats to Validity

The main threats to validity identified in this review are described next:

- **Selection bias.** The studies selection from web search engines and key digital libraries do not ensure that all RE approaches for SPL were reviewed. Possibly, relevant approaches were excluded from this review. Thus, for reducing this threat, we searched for referenced papers, and also analyzed theses and technical reports found through an ad-hoc search.

- **Research Questions.** It is likely that some questions defined in the protocol may not be so relevant. They are not the unique questions to address in this area. To minimize this threat, several meetings were held in order to discuss the questions.

- **Data Analysis Bias.** Sometimes the researchers experience may influence the results, and sometimes a reduced set of papers or a unique paper addressing specific issues, may also influence. Aiming to avoid it, we had a set of discussion meetings with the project members and experts in the area.

3.6 Related Work

There are other studies analyzing the context of RE for SPL. However, none of these studies are done through a systematic review with the same goal presented in this chapter.

Kullor and Eberlein (Kuloor and Eberlein, 2002) compare some SPL approaches, analyzing their RE essential activities. They do not describe which criteria were considered for identify the gaps in the approaches, so their analysis is questionable. For example, it describes that the FODA process does not have any process for requirements specification. But the feature modeling is a type of requirements specification. It assesses the applicability of several techniques to SPL development. However, it does not define techniques of SPL, such as domain analysis and feature modeling.

Birk et al. (Birk et al., 2003b) present the report of the GI Work Group, composed by Robert Bosch GmbH, Hewlett-Packard (HP), Fraunhofer IESE, University of Aachen and the company sd&m AG. It summarizes the results of a systematic survey of existing
CHAPTER 3. A SYSTEMATIC REVIEW

3.7. CHAPTER SUMMARY

SPL problems and solutions that were present in the organizations. Some fundamental ideas that were evaluated during the studies also are presented. It describes different SPL situations that were experienced by group members. Thus, it presents the practice state, being a good study source for SPL development. It differs of our review by its experimental characteristics and by it does not treat all RE disciplines; it focuses on presenting problems found in the organizations and proposing solutions for them.

Kovacevic et al. (Kovacevic et al., 2007) present a survey about current status of practice in the RE area for Model-Driven Development (MDD) and SPL. It identifies possible alternatives that can explore the synergies of MDD and Aspect-Oriented Software Development (AOSD). The approaches are compared according to common and specific criteria, such as scalability, traceability, modeling, scoping and validation. Furthermore, the SPL approaches are grouped according to some RE approaches types, as follow: feature-oriented, use case-based, use case and feature-based, viewpoint-based, goal-based, and aspect oriented. From a requirements elicitation perspective, this survey does not present criteria for comparison.

### 3.7 Chapter Summary

In this Chapter, it was presented a systematic review in which the main goal was to identify the available SPL approaches to verify how they deal with Requirements Engineering. This review can be useful for researchers since it collects information about the studies strengths and weaknesses.

The effort and quality of the review presents an important result that can be used as background information for requirements analysts, researchers and companies interested in Software Product Lines and Requirements Engineering.

This study identified also that each approach analyzed had interesting insights about at least one, or more requirements engineering disciplines, and each of these approaches can be used for the creation of a more complete and formal process definition.

The results of this review was the basis for the definition of the proposal process, as described in the next Chapter.
RiPLE-RE Process

The quality of a software product heavily depends on the people, organization, and procedures used to create and deliver it (Fuggetta, 2000). These procedures when described as a process are important because they allow knowledge to be reused (Kotonya and Sommerville, 1998).

There are several definitions on software process (Fuggetta, 2000; Pressman, 2005; Sommerville, 2006). According to (Sommerville, 2006), a software process is a set of activities that leads to the production of a software product. In the context of requirements engineering, a process is a structured set of activities which are followed to derive, validate and maintain a systems requirement document (Kotonya and Sommerville, 1998). (Ezran et al., 2002) consider that processes are nested: they may be broken down into sub-processes until reaching atomic tasks (sometimes called activities, performed uninterruptedly by one person in one place).

According to (Kotonya and Sommerville, 1998), a complete RE process description should include what activities are carried out, the structuring or schedule of these activities, who is responsible for each activity and the tools used to support requirements engineering.

In this context, this chapter presents the proposed requirements engineering process for software product lines, its overview, foundations, activities and elements.

4.1 Introduction

This work aims specifying a requirements engineering process for software product lines, in which its goal is to be usable, effective and efficient. According to the SPL lifecycle, its scope is the core assets development, i.e., development for reuse. The product development with reuse is out of the scope of this work, but it has direct relationship with
CHAPTER 4. RIPLE-RE PROCESS

4.1. INTRODUCTION

this process. During product derivation, for each variant asset, it is decided whether the asset is (or is not) supported by the product to be built. Thus, when a domain requirement is instantiated it is became in a concrete product requirement.

The proposed requirements engineering process is called RIPLE-RE. It is part of the RIPLE (The RiSE Process for Product Line Engineering) project, which is being developed within Reuse in Software Engineering (RiSE) Labs \(^1\) to cover the whole SPL lifecycle \(^2\).

According to Chastek et al. (Chastek \(et\ al.\), 2001), establishing the requirements for a SPL is an iterative and incremental effort covering multiple requirements sources with many feedback loops and validation activities. Thus, the RIPLE-RE defines a process that can be performed in an incremental and iterative mode, enabling the refinement and improvement of requirements in different steps of the SPL lifecycle. The incremental and iterative characteristics can be applied when the domain is not well known, the domain expert availability is difficult and there are time constraints.

In terms of SPL adoption strategies, the process considers the reactive strategy as well as the proactive strategy by both analysing existing product requirements and envisioning new ones. The evolution management process (RIPLE-EM) is responsible to manage the changes according to each SPL adoption strategy.

Considering that a SPL requires a systematic planning (Chastek \(et\ al.\), 2001; Clements and Northrop, 2001) and that a systematic process is essential to ensure complete, consistent and relevant requirements (Kotonya and Sommerville, 1998), this work is defined with activities, tasks, inputs, outputs, roles and guidelines. Its modeling was based on Software & Systems Process Engineering Meta-Model 2.0 (SPM 2.0), standard developed by Object-Management Group (OMG) (OMG, 2008).

RIPLE-RE consists of three activities: Model Scope, Define Requirements and Define Use Cases. An overview of the process is shown in Figure 4.1.

This Figure shows the main output of the RIPLE-RE, the Domain Requirements Specification (DRS). It is composed of Feature Model, Domain Requirements and Domain Use Cases. These work products are specified with notations that represent SPL variabilities and enable the product instantiation. Then, when instantiated, the DRS becomes a concrete product requirements specification. Each work product represents a view with complementary abstraction level, as follows:

- **Feature Model.** Specification with an abstraction in high level that purposes

\(^1\)http://www.rise.com.br/research.

\(^2\)A discussion involving the reasons to propose a new product line process can be seen in (Almeida \(et\ al.\), 2005; Almeida, 2007).
to model the SPL scope. It represents the variabilities and commonalities in an
general view, as perceived by users, making easy the products configuration and
understanding by the stakeholders. Its view is often static (Chastek et al., 2001),
without capabilities internal details. In general, it provides the “which” of the
domain: which functionality can be selected when engineering new systems in the
domain (Griss et al., 1998).

• **Domain Requirements**. View addressed to specify SPL requirements in functional
and non-functional levels. It presents a description of what systems in the domain
do and how the systems are supposed to be.

• **Domain Use Cases**. View more detailed than other models, with details of interactions
between actors and systems. Thus, it is addressed for the users and development team. However, use cases are not well suited to capture easily non-functional requirements or without interaction. In this work, the functional requirements are
specified in an abstraction level that encompass the use cases. Thus, an use case always is derived of a functional requirement.

In Figure 4.1, it is also possible to observe the direct relationship of the RiPLE-RE with other processes which compose RiPLE: Scoping, Evolution Management (EM) and Design. These processes can be performed in different modes, but they must be consistent with the inputs and outputs of the RiPLE-RE.

The Scoping process defines the Product Map - matrix where rows and columns are used to represent features and products, allowing the user to identify, among all possible features, which ones should be part of the SPL. This asset is essential to start the RiPLE-RE activities responsible for refining the scope.

The communication between RE and EM is performed by Change Requests (CR) and tasks. If the scope definition is insufficient to the DRS, RE requests change to EM. EM analyzes the change and when approved it sends a task to Scoping. In addition, EM sends tasks to RE when the domain requirements need be modified. It occurs, for example, when product characteristics or scope change over time; and inadequate or insufficient specified requirements have to be improved.

The relationship between RiPLE-RE and Design is supported by the DRS, which provides the basis for the architecture. When requirements are specified inadequately or insufficiently, it can be difficult to design an acceptable domain architecture. In this situation, the Design process must notify the requirements problem, sending a CR to EM.

On the composition of the RiPLE-RE, its activities define the sequence of tasks and each task is related with roles and work products (inputs and outputs). The Requirement Analyst is the main role to perform this process. Table 4.1 presents a summary of the process, with Activities, Tasks, Inputs, Outputs and Roles.

As it can be seen in Table 4.1, all activities have the common tasks Elicit and Verify, which are performed in an iterative way to attend the purpose of each activity. The other tasks (Model Feature, Describe Requirements and Describe Use Cases) are addressed for specifying the domain requirements according to the view of each work product.

Some tasks can be performed at the same time when appropriate. For example, information may be directly elicited and modeling at the same time. Moreover, during modeling, the task Elicit can be solicited when information is not yet complete or considered inconsistent.

The following sections present tasks and guidelines of the RiPLE-RE. The next sections order does not represent the order in which the tasks are performed, because there are common tasks for all activities (Task Elicit and Task Verify). The Section 4.2
CHAPTER 4. RIPLE-RE PROCESS

4.2 Elicit

Table 4.1 Summary of the RiPLE-RE Process

<table>
<thead>
<tr>
<th>Activities</th>
<th>Tasks</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Roles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elicit</td>
<td>Existing Assets, Product Map,</td>
<td>Raw DRS, Glossary</td>
<td>Requirements Analyst, Domain</td>
<td>Domain expert, Domain analyst</td>
</tr>
<tr>
<td></td>
<td>Glossary</td>
<td></td>
<td></td>
<td>Decision-markers, Domain analyst</td>
</tr>
<tr>
<td>Model Scope</td>
<td>Product Map, Raw DRS</td>
<td>Feature Model, Trace Links</td>
<td>Requirements Analyst</td>
<td></td>
</tr>
<tr>
<td>Verify</td>
<td>Feature Model</td>
<td>Verification Report, Change</td>
<td>Domain Expert, Domain Analyst,</td>
<td>Requirement Analyst, SQA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Request</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Elicit</td>
<td>Existing Assets, Glossary,</td>
<td>Glossary, Raw DRS</td>
<td>Domain expert, End User,</td>
<td>Requirements Analyst</td>
</tr>
<tr>
<td></td>
<td>Feature Model</td>
<td></td>
<td>Requirements Analyst</td>
<td></td>
</tr>
<tr>
<td>Define Requirements</td>
<td>Feature Model, Raw DRS</td>
<td>Domain Requirements, Trace</td>
<td>Requiremets Analyst</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Links</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Verify</td>
<td>Domain Requirements</td>
<td>Verification Report, Change</td>
<td>Domain expert, End users,</td>
<td>Requirement Analyst, SQA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Request</td>
<td>Requirement Analyst</td>
<td></td>
</tr>
<tr>
<td>Elicit</td>
<td>Existing Assets, Feature Model,</td>
<td>Glossary, Raw DRS</td>
<td>Domain Expert, Requirements</td>
<td>End User</td>
</tr>
<tr>
<td></td>
<td>Domain Requirements, Glossary</td>
<td></td>
<td>Analyst</td>
<td></td>
</tr>
<tr>
<td>Define Use Cases</td>
<td>Domain Requirements, Raw DRS</td>
<td>Domain Use Cases, Trace</td>
<td>Requirements Analyst</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Links</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Verify</td>
<td>Domain Use Cases</td>
<td>Verification Report, Change</td>
<td>Domain Expert, Requirement</td>
<td>End Users, SQA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Request</td>
<td>Analyst, End User</td>
<td></td>
</tr>
</tbody>
</table>

presents the Task Elicit. In the Sections 4.3, 4.4 and 4.5 are defined the tasks addressed to specifying the SPL requirements, respectively: Model Feature, Define Requirements and Define Use Cases. Section 4.6 describes the Task Verify. Finally, the Section 4.7 describes guidelines to evolution management that support all activities and the RiPLE-EM process.

4.2 Elicit

Requirements elicitation for a SPL must capture anticipated variations over the foreseeable lifecycle of the SPL (Clements and Northrop, 2001). In general, its information sources are probably larger than for single-system requirements elicitation. Thus, the Elicit task is responsible for identifying domain requirements and variations in a SPL. It also reaches agreement to satisfy all stakeholders, solving conflicts that are identified. Its main inputs are the Product Map (scope of the SPL), the glossary and existing assets.

In the process initiation, this task must identify the requirements overview, defining the boundary that divides the SPL and the real world that surrounds the solution, identifying the main variations among its products. These variations must reflect value for organizations and present differential among products.

Elicit is an iterative task in which new information can be added in each iteration according to its activity and work products. For example, in the Define Use Cases activity (see Section 4.5) can be necessary to identify some information not yet elicited, such
as actors, scenarios, pre-conditions ans their variations. It can be elicited using the existing systems and assets, interviewing potential users and domain experts. In the Define Requirements activity (see Section 4.4) can be necessary to identify and negotiate priorities with different stakeholders. It can need of different stakeholders’ viewpoints, so it can result in conflicts. In this case, it is recommendable performing elicitation in a brainstorming session, in which will be performed requirements elicitation and negotiation.

The elicitation strategy must be adopted according to the information source type, so its identification is essential for this task. Examples of information sources and their information provided are shown in Table 4.2.

**Table 4.2** List of possible information sources (based on (Kang *et al.*, 1990; Chastek *et al.*, 2001))

<table>
<thead>
<tr>
<th>Sources</th>
<th>Information Provided</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain experts</td>
<td>Domain problems, solutions, future needs and terminology</td>
</tr>
<tr>
<td>Customers</td>
<td>Product features, expected qualities, requirements priorities</td>
</tr>
<tr>
<td>End Users</td>
<td>Product features, requirements, scenarios, actors</td>
</tr>
<tr>
<td>Executives / Owner</td>
<td>Requirements priorities, business goals, marketing strategies, time and resources constraints</td>
</tr>
<tr>
<td>Managers</td>
<td>Resource constraints</td>
</tr>
<tr>
<td>Domain Analyst</td>
<td>Domain features</td>
</tr>
<tr>
<td>Architects/Developers</td>
<td>Technical feasibility of the requirements</td>
</tr>
<tr>
<td>Maintainers</td>
<td>Knowledge of past changes needed</td>
</tr>
<tr>
<td>Marketers</td>
<td>Features of existing and anticipated products, knowledge of competing products</td>
</tr>
<tr>
<td>Domain standards/ Regulatory organizations</td>
<td>Conformance requirements, future standards, safety requirements, and legal issues</td>
</tr>
<tr>
<td>Textbooks</td>
<td>Domain-specific techniques and theory</td>
</tr>
<tr>
<td>Existing assets</td>
<td>Domain requirements, scenarios, actors, features, constraints</td>
</tr>
<tr>
<td>External systems</td>
<td>Interface and technological constraints</td>
</tr>
</tbody>
</table>

The identification of the SPL requirements can be performed by interview or brainstorming sessions with stakeholders that provide the necessary information. However, the potential stakeholders might be busy (e.g. expert domain with little availability), so it is recommendable that elicitation starts by other information sources, such as outputs from previous activities (e.g. product map for Model Scope activity), existing assets (e.g. user manual, available systems) and textbooks. If information cannot be found in these sources, then it is necessary to contact the stakeholders. Moreover, it is necessary direct and intensive contact with stakeholders when there are not existing assets and systems. Figure 4.2 presents the systematic of the elicitation.

In this Figure the investigation in existing documentation or assets represents any information source which is not a stakeholder, for example, available DRS, textbooks, existing systems and their documentation.

When elicitation is conducted in a session brainstorming, it is necessary elicit a major
account of information. This session must be moderated by the requirement analyst and can be conducted in more than one day. The session must be in presence of the domain analysts, domain experts and of other stakeholders of particular concern in the SPL. This brainstorming provides an opportunity to gather stakeholders together to provide input about their needs and expectations with respect to key requirements that are of particular concern to them. All participants are expected to be fully engaged and present throughout the brainstorming. Participants are encouraged to comment and ask questions at any time during the brainstorming. However, it is important to recognize that the moderator may occasionally have to cut discussions in order to control the time or when it is clear that the discussion is not focused on the required brainstorming outcomes. Moderator also should help stakeholders create well-formed requirements.

After elicitation, similar requirements are consolidated when reasonable. The vocabulary used to describe requirements varies widely, so it is necessary to search domain common terms and update them in the glossary. If common terms are not defined, different perceptions of domain concepts could cause confusion among stakeholders and lead to time-consuming discussions.

Therefore, the Elicit task is concluded in each iteration with the updated glossary and raw DRS, which will be input for the following tasks.
4.3 Model Feature

The purpose of this task is to model the SPL scope defined, so it is the first task of interpreting and ordering the requirements. The purpose of this task is modeling the defined SPL scope, so it is the first task of interpreting and ordering the requirements. Its main inputs are the Product Map and Row DRS (from the elicitation).

This scope is modeled using features, which are externally visible characteristics of products in a domain (Lee et al., 2002) and represent an abstraction from requirements (Gurp et al., 2001). In particular, features are characteristics that are used to differentiate among members of a SPL and hence to determine and define its common and variable functionalities (Gomaa, 2005).

Feature modeling captures and represents commonalities and variabilities of the features in a model, so this information may help developers to identify which parts of their system need to support variability. This modeling is based on FODA (Lee et al., 2002), in which the features are organized into a AND/OR hierarchy diagram that captures structural or conceptual relationships among features. According to (Sinnema et al., 2004), this hierarchical organization of variability representation reduces the cognitive complexity that these large numbers impose on engineers that deal with variation points.

This task is part of the activity Model Scope. Figure 4.3 shows an overview of this activity. Scoping and Model Scope may have to be performed in parallel or iteratively until the scope becomes complete, at which time detailed feature modeling may start. When new features are identified in Model Scope, the Scoping needs to approve the inclusion of this new feature in SPL. Thus, a change request must be submitted to EM, and EM performs communication with Scoping. Whether a feature is updated in the scope, it also must be updated in the feature model.

In the following sections are presented guidelines for features modeling. The examples illustrated for explicit representation of variation points and variants will use a simplified example of a home automation domain based on (Pohl et al., 2005).

4.3.1 Identify Features

There are some questions about the abstraction level of features modeling. Some questions are often identified, such as “To what extent it is worth refine a feature which has many specific sub-features?”. Thus, this work presents the following guidelines to help to identify features and avoid these doubts.
• A feature model must represent features that reflect value and competitiveness for organizations launch product quicker. Thus, it is necessary to represent the features more relevant and that present a differential among products. Some variations are irrelevant to represent in a feature model, since they do not represent decisions with differential. For example, it does not represent variations in attributes level with low technical impact. Other situations of irrelevance must be observed by the analyst. A typical situation is when can be necessary represent several reports in a SPL. However, the reports amount can become the feature model a complex model with many non-relevant features. In this case, the solution can be create categories that group reports more relevant and that add a differential among products (e.g. quality reports, managerial reports and financier reports).

• Features can be refined into sub features. It must be performed to represent relevant variations in a more detailed level. The sub-features are identified during the feature
model representation, which will be seen in the next section.

- It does not identify all implementation details that do not distinguish products in a domain. A skillful developer tends to enumerate all the implementation details and identify them as features, even though there are no variations among them. It is necessary remember that a feature model is not a requirement model, which expresses the details of internal functions. The modeling focus should be on identifying properties, factors, assumptions that can differentiate one product from other in the same domain, not on finding all implementation details that are necessary to implement the products in a domain (Lee et al., 2002).

- Attached features that always exist together can be grouped in a single feature. For example, the User feature always exists in presence of the Login/Logout feature, so it can be yet grouped into Authentication feature. However, some constraints must be observed for the features grouping:
  
  - If there are dependency constraints (e.g. implication or exclusion) in a part of a grouped feature, it is necessary ungroup the feature part with distinct constraints.
  
  - If there are distinct variabilities types among grouped features (i.e. a part is optional and another is feature) it is necessary ungroup the features.
  
  - Related features that are required by all members of the SPL are packaged into mandatory features.
  
  - A grouped feature must define in its description the other grouped features.

Another way to identify new features is by the feature model representation, which is presented in the following section.

### 4.3.2 Represent Features

Features are modeled in terms of the following relationships:

- **Composed-of.** This relationship is used if there is a whole-part relationship between a feature and its sub-features (parent-role: whole, child-role: part);

- **Generalization/specialization.** In cases where features are generalization of sub-features, they are organized using the generalization/ specialization relationship (parent-role: general-entity, child-role: specialized-entity). Two or more features
with similar functionalities can be generalized into one feature by abstracting the differences into variability through feature generalization. It is possible to define feature in more general fashion which helps to identify a common functionality among products;

- *Implemented-by*. The implemented-by relationship is used when a feature (i.e. a feature of an implementation technique) is necessary to implement the other feature;

- *Dependency*. This relationship represents dependency constraints of implication (require) or exclusion. It is used to constrain the selection from variant features. That is, it is possible to specify which features should be selected along with a designated one and which features should not. In this relationship, a mandatory feature cannot depends on another, but a variant feature can depends on a mandatory feature.

The features model must not represent functional dependencies, like a function call hierarchy, but organize features to capture and represent commonalities and differences (Lee *et al.*, 2002). Feature model must represent relationships among features that impact in decision making in the products scope.

In the feature model, there are concrete and conceptual features. A concrete feature represents a real feature in the SPL scope. The conceptual feature is used only to help in the model organization. It is not represented in the SPL scope. For example, the feature at the root of the tree is called root feature and it is usually a conceptual feature that represents the whole SPL. Another example is in the generalization relationship, in which the general-entity is normally a conceptual feature. For composed-of relationship, the parent and child features are usually concrete.

A key feature in the feature model is the Variation Point (VP). VP represents a variability subject and it is composed by variability objects (options of the VP) (Pohl *et al.*, 2005). An example of the VP is shown in the Figure 4.4. This VP is part of a SPL example in the home automation domain used in (Pohl *et al.*, 2005). It represents that the “Door Lock” VP has the options “Fingerprint scanner” and “Keypad”.

The commonality and variability representation is defined by the feature types and are given as follows:

- *Mandatory*. Common feature among all products, i.e. it always should be selected by the products of the SPL.
• **Optional.** One feature may or may not be selected for a product.

• **Alternative.** This feature type is applicable only for features that are options of a VP. It signifies that at least one variant should be selected for the VP. The variations are mutually exclusive. It corresponds to cardinality (1:1), i.e. in the minimum and in the maximum one variant can be selected, when VP is a mandatory feature. Whether VP is an optional feature, the cardinality is (0:1).

• **OR.** This feature type is applicable only for features that are options of a VP. It signifies that one or more variants may be selected for the VP. It correspond to cardinality (1:n). For representing of the cardinality (0:n), i.e. zero or more variants may be selected, is necessary represent variation point as an optional feature.

All variability objects of a VP, i.e. its variants, must have same type (Alternative or OR). An example with all those types is presented in Figure 4.5.

The Figure 4.6 shows the legend of the presented feature model. Each feature should be specified with the attributes presented in the Table 4.3.
4.4 Describe Requirements

The purpose of the task Describe Requirements is to specify the SPL requirements in a functional and non-functional view, describing also the SPL priorities. This task is part
of the activity Define Requirements, which is presented in Figure 4.7.

This Figure shows the dependencies among the tasks that compose the activity Define Requirements. The requirements are specified from the output of the task Elicit, having the Feature Model and the Row DRS as inputs. The requirements identification (Task Elicit in Section 4.2) should be performed after the activity Model Scope because Feature Model is the initial basis for the requirements. Therefore, it is important to observe the relationship between features and requirements, which is a many-to-many association, thus one feature can encompass many requirements, and different requirements can encompass many features. It is also possible to group requirements that are reused together into a feature. A requirement may also have several VPs within it, where each VP can be mapped to a feature. Finally, it is also possible that some requirements are not mapped into features. It occurs normally with non-functional requirements (e.g. Maintainability).

After requirements identification, the specification is started according to task De-
scribe Requirements. These requirements must be managed in order to maintain the traceability among different assets and controlling the SPL evolution. For this traceability, the trace links are maintained according to guidelines defined in Section 4.7. After it, the verification must be performed according to Task Verify (Section 4.6).

The following sections present the steps to perform the task Describe Requirements.

### 4.4.1 Classify Requirements

The step Classify is responsible for list all elicited requirements and group them in functional and non-functional category. Functional requirements describe what the system should do and non-functional requirements place constraints on how these functional requirements are implemented (Sommerville, 2005). For example, a functional requirement might state that a system must provide some facility for authenticating the identification of a system user; a non-functional requirement might state that the authentication process should be completed in four second or less. However, it is not always as simple as this. A requirement could be interpreted as functional or non-functional according to the above definition. High-level non-functional requirements are often decomposed into functional system requirements (Sommerville, 2005).

The non-functional requirement can be quality attribute or constraints, such as are shown in Table 4.4.

After classification, the requirements are detailed. The steps Classify and Detail can be performed iteratively until the domain requirements become complete.

### 4.4.2 Detail Requirements

In this step, the requirements are described and their variabilities are represented. There are mandatory and variant requirements. A mandatory requirement is common for all products in the SPL. A variant requirement is not common for all products in the SPL, it there is in some products. The variability can also be found in the requirement part, i.e. a text fragment can be a variant.

For each requirement are defined the attributes defined in the Table 4.5

Non-functional requirement should be, wherever possible, quantified in its description to measure and assure objectively the precision of the results. The following metrics are presented as examples on (Sommerville, 2005):

- Usability: i) Time taken to learn 75% of user facilities and ii) Average number of errors made by users in a given time period.
Table 4.4 Non-Functional Requirements Types

<table>
<thead>
<tr>
<th>Quality Attributes</th>
<th>Usability</th>
<th>It is how easy it is for the user to accomplish tasks and what support the system provides for the user to accomplish this.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maintainability</td>
<td>This should specify attributes of software that relate to the ease of maintenance of the software itself. There may be some requirement for certain modularity, interfaces, complexity, etc. Requirements should not be placed here just because they are thought to be good design practices (IEEE Std. 830, 1998)</td>
<td></td>
</tr>
<tr>
<td>Security</td>
<td>It is the ability of the system to prevent or resist unauthorized access while providing access to legitimate users. An attack is an attempt to breach security. Examples these requirements are authentication and authorization.</td>
<td></td>
</tr>
<tr>
<td>Availability</td>
<td>It is concerned with system failure and duration of system failures, i.e. when the system does not provide the service for which it was intended.</td>
<td></td>
</tr>
<tr>
<td>Portability</td>
<td>This should specify attributes of software that relate to the ease of porting the software to other host machines and/or operating systems (IEEE Std. 830, 1998).</td>
<td></td>
</tr>
<tr>
<td>Performance</td>
<td>The degree to which the software accomplishes its designated functions within given constraints, such as speed, accuracy, or memory usage. This attribute characterizes the timeliness of the service delivered by the computer system.</td>
<td></td>
</tr>
<tr>
<td>Reliability</td>
<td>A set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time.</td>
<td></td>
</tr>
</tbody>
</table>

| Constraints     | Process | It is related with technology, persistency, tools, and patterns. |
|                | External | These constraints are related with legal and licensing issues, hardware, and operational environment. |
Table 4.5 Requirements attributes

<table>
<thead>
<tr>
<th>Id</th>
<th>Unique identifier, which can be standardized according to requirement type. Thus, whether requirement is functional, its identifier can be FRX. Whether requirement is non-functional, its identifier can be NFRX. In both cases, X is a unique number for identification.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>It can be Functional or Non-Functional.</td>
</tr>
<tr>
<td>Name</td>
<td>The name that better represent the requirement.</td>
</tr>
<tr>
<td>Variability Type</td>
<td>Mandatory or Variant.</td>
</tr>
<tr>
<td>Binding time</td>
<td>It is defined as in feature model, i.e. time at which the decision for a variant requirement is bound.</td>
</tr>
<tr>
<td>Priority</td>
<td>The priorities are relative to core asset implementation order. They can be High, Medium or Low.</td>
</tr>
<tr>
<td>Rationale</td>
<td>It defines reasons for requirement existence.</td>
</tr>
<tr>
<td>Description</td>
<td>Description about the requirements that can have variant text fragment that represent a VP.</td>
</tr>
<tr>
<td>Implication</td>
<td>It presents dependency relationship between requirement and another requirement object (requirement or requirement part). It occurs when a requirement requires another object.</td>
</tr>
<tr>
<td>Exclusion</td>
<td>It means that a requirement cannot exist in presence of another requirement object.</td>
</tr>
</tbody>
</table>

- Performance: i) Number of transactions to be processed per second and ii) Response time to user input.
- Reliability: i) Mean time to failure and ii) Rate of occurrence of failure.
- Availability: i) Probability of failure on demand.

The requirement variability type can be defined from feature model whether requirement is mapped to feature and this feature represents directly the requirement or it encompasses the requirement. When a feature encompasses requirements, their variabilities types are the same. In this case, whether feature variability type is Mandatory then requirement variability type is Mandatory. Whether feature variability type is Optional, Alternative or OR, then the requirement variability type is Variant.

When requirement is not mapped to feature then its variability type must be identified in elicitation (see Section 4.2). In this case, it must identify whether all products require or does not require the requirement.

The SPL requirements priorities must be defined by the decision-maker (e.g. customer, executive and SPL owner). However, it is necessary that the SPL development
team support this work for assure consistent priorities, analyzing dependencies among requirements and their reuse potential. Potential end-users and domain expert also can have influence in the priorities decisions. Therefore, stakeholders with different goals and needs can define conflicting priorities, so it is necessary negotiate conflict among priorities.

These priorities affect directly the core assets implementation order and the products releases plan. Thus, they should be carefully analyzed according to following types:

- **High.** This requirement is absolutely needed. The SPL will not enable the derivation of an useful product without this requirement.

- **Medium.** This requirement is important to the stakeholder and will provide a key driver to convince stakeholders of usefulness of the derived products. Including this requirement into the final system will provide verifiable benefits to the stakeholder.

- **Low.** This requirement clearly provides additional value to the stakeholder, but it can be omitted without affecting the main uses of the derived products.

The priorities can be influenced by several factors, such as:

- **Business strategies.** The decision-makers can define requirement priority according to its cost, time to market, value and competitively. These factors can characterize also the priorities among products. For example, the release of the first product can be based on the high amount of core assets, high business value, and time to market not so big.

- **Variability type.** For optimizing the SPL reuse potential, it is best that mandatory requirements are implemented before variant requirements. However, in some situations can have variant requirement with major priority than mandatory requirement. For example, whether a variant requirement presents a market differential for a product that need be launch quicker. Or, whether a mandatory requirement does not present differential for the products, its priority is low.

- **Design constraints.** Relationship of dependency among requirements can affect their priorities.

The variability found in requirement text fragment is represented by Variation Point (VP). A requirement can have 0 to n VPs (see example in Figure 4.9). For each VP should be identified the attributes presented in Table 4.6.
Table 4.6 Requirement Variation Point attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>All VPs have unique identifier to assure traceability among objects. The identifier can be standardized as ( Y.VPX ), where ( Y ) is the identifier of the requirement that VP is part, and ( X ) represents a sequential number of the VPs in the requirement (e.g. FR1.VP1).</td>
</tr>
<tr>
<td>Description</td>
<td>Characterization about VP. It can inform the basis to make choice among the variants. It is optional.</td>
</tr>
<tr>
<td>Cardinality</td>
<td>It indicates how many variants can be applied to the variation point and are given by pairs ([\text{min}, \text{max}]). This attribute is required when there are two or more variants. Some examples of cardinality are: ([1, 1]) - Exactly one variant should be selected for a variation point, variants are mutually exclusive; ([1, n]) - At least one variant should be selected for a variation point, the variants are not mutually exclusive; ([0, 1]) - One variant may or may not be selected for a variation point; and ([0, n]) - Zero or more variants may be selected for a variation point.</td>
</tr>
<tr>
<td>Variant</td>
<td>A variation point can have 1 to n variants. These variants are options of the VP. Each variant also have a unique identifier that must be standardized as ( Y.X.VZ ), where ( Y ) and ( X ) are respectively the identifiers of the requirement and VP that the variant is part, and ( Z ) represents a sequential number of the variant in the VP (e.g. FR1.VP1.V1).</td>
</tr>
<tr>
<td>Binding time</td>
<td>It decides when variants are bound to the member of the SPL. The scoping time is the default value, so whether it section is not defined, the default value is assumed.</td>
</tr>
<tr>
<td>Implication</td>
<td>It presents dependency relationship among variations. It occurs when a variation (variation point or variant) requires another variation. More than one implication should be separated by commas.</td>
</tr>
<tr>
<td>Exclusion</td>
<td>It means that a variation cannot exist in presence of another. More than one exclusion should be separated by commas.</td>
</tr>
</tbody>
</table>
The proposed template for requirements is structured in XML (eXtensible Markup Language), and XSLT (eXtensible Stylesheet Language Transformation) is used to enable XML transformation to other outputs (e.g. HTML document). Thus, XML enables the explicit documentation of the variability and XSLT provides the capability of processing XML documents.

All requirements are added in natural language text in terms of XML tags. The XML Requirement, presented in Figure 4.8, is the document used to specify the SPL requirements with XML tags. This document is transformed to derive the requirements in an interface usable. Thus, XSLT SPL Requirement is responsible by describe as XML Requirement will be transformed by Transform XML/XSLT to generate the SPL requirement document in HTML format (HTML SPL Requirement). In XSLT Product Requirement, the product decisions about the variabilities are stored. It is responsible by describe the transformation of the XML according to the decisions of a product, so it enables generate the product requirement document (HTML Product Requirement) by the Transform XML/XSLT, which selecting the variabilities.

![Figure 4.8 XML Template](image)

The notation used in example of the Figure 4.9, 4.10 and 4.11 employs tags to mark text fragments. It provides an identifier of the corresponding variant for each fragment. Consequently, it is possible to select the text fragments that belong to specific variants.

Dependencies can occur among different objects, but a mandatory object cannot depend on a variant object. In the context of requirements, these dependencies can be requirement-requirement, requirement-VP, VP-VP (e.g. Figure 4.9), VP-variant, variant-variant and requirement-variant.
CHAPTER 4. RIPLE-RE PROCESS

4.5 Define Use Cases

The purpose of this task is to represent the requirements and their variabilities in the Use Case view. Use Case (UC) is widely used in modern industrial development, so it seems natural trying to find an effective way to combine it with the SPL. However, it must be selected in situations where user-level information is essential for domain model (John and Muthig, 2002).

In this approach, functional requirements are defined in terms of use cases and actors. An actor is an user of the system, i.e., they are external entities that interact with the system and can be human users or other systems. An use case defines a sequence of interactions between one or more actors and the system.

The detail level and total specified use cases will depend of the SPL context. For example, in projects with constraints of time it might be more useful identify all use cases but detail only use cases that have variability, complexity or are unknown by team. This decision depends of the domain complexity and maturity, organizational constrains and team understanding. Whether domain is well understood, it can specify only use cases

From the stable requirements are defined the use cases, as can be seen in the following section.

4.5 Define Use Cases

The system must monitor the state of doors, whether they are open, closed, locked, or unlocked.

- <description>
  - <text>The system must monitor the state of doors, whether they are open, closed, locked, or unlocked.</text>
  - <vp id="FR1.VP1">
    - <variant id="FR1.VP1.V1">Doors can be unlocked electronically based on the following identification mechanism:</variant>
  </vp>
  - <vp id="FR1.VP2" binding-time="scoping-time" implication="FR1.VP1">
    - <description>Identification mechanism Type</description>
    - <cardinality min="1" max="3" />
      - <variant id="FR1.VP2.V1">fingerpint scanner</variant>
      - <variant id="FR1.VP2.V2">keypad</variant>
      - <variant id="FR1.VP2.V3">magnetic card</variant>
  </vp>
</description>
</requirement>

Figure 4.9 Example of a functional requirement with implication among VPs.
CHAPTER 4. RIPLE-RE PROCESS  

4.5. DEFINE USE CASES

Figure 4.10  Example of a functional requirement with implication between a variant and VP.

- <requirement id="FR2" type="Functional">  
  <name>Fire Detection</name>  
  <variability-type>Variant</variability-type>  
  <priority>High</priority>  
  <rationale>Assure that people are freedom from danger.</rationale>  
  <description>  
   The system must be able to detect fire. When fire is detected the system activates the alarm.</description>  
  <vp id="FR2.VP1" binding-time="scoping-time">  
   <description>Actions after fire detection</description>  
   <cardinality min="1" max="n" />  
   <variant id="FR2.VP1.V1">shuts all windows</variant>  
   <variant id="FR2.VP1.V2">closes gas valves</variant>  
   <variant id="FR2.VP1.V3">deactivates power outlets</variant>  
   <variant id="FR2.VP1.V4">switches on emergency lights</variant>  
   <variant id="FR2.VP1.V5">implication="FR1.VP1" unlocks doors</variant>  
   <variant id="FR2.VP1.V6">informs the fire station</variant>  
  </vp>  
</requirement>

Figure 4.11  Non-Functional Requirement with implication among requirements.

- <requirement id="NFR1" type="Non-Functional" implication="FR2">  
  <name>Fire detection Performance</name>  
  <variability-type>Variant</variability-type>  
  <priority>High</priority>  
  <rationale>Control system performance</rationale>  
  <description>  
   Fire detection cannot to delay more than two seconds.</description>  
</requirement>

more complex or few understood.

This task is part of the activity Define Use Cases, which is presented in Figure 4.12.  

This Figure shows the dependencies among the tasks that compose the activity Define Use Cases. The Use Cases are specified from the output of the task Elicit (Section 4.2), having the Domain Requirements and Row DRS as inputs. After use cases identification, the specification is started according to task Describe Use Cases. These use cases must be managed in order to maintain the traceability among different assets and controling the SPL evolution. For this traceability, the trace links are maintained according to guidelines defined in Section 4.7. After it, the verification must be performed according to Task Verify (Section 4.6).

The following sections present the task Describe Use Cases, which is defined in steps.
4.5.1 Identify Use Cases

The elicited functionalities can represent disordered information that need be organized in each view of the DRS, such as domain use cases. From requirements description (see Section 4.4) are identified all SPL use cases, so use cases always are mapped to requirements (use case is always part of or correspond to a whole requirement). For this identification, it is important observe that the relationship between requirements and use cases is a many-to-many association, such that one requirement could encompass many use cases, whereas an use case could be encompassed by many requirements. Often an use case is encompassed by one requirement. However, for example, an use case could be a secondary functionality that is encompassed by several requirements (when reused).

In the example of the Figure 4.10 presented in the previous section, the Fire detection requirement can be break down in some use cases. Each variant could be a new use case. This decision will depend of the functional complexity of each variant, which is discussed in the next section.

After identifying use cases, it is necessary document them according to their elements,
which are presented in next section. Information (e.g. actors, interaction between actor and system, preconditions) still does not elicited must be gathering according to guidelines of the Task Elicit (Section 4.2).

### 4.5.2 Detail Use Cases

When utilizing use cases for product line modeling, they should be extended with variability mechanism. In this context, several elements may potentially be a variant, such as use case as whole, actors, dependency, rationale, precondition, postcondition, and event flows. An actor is a variant, for example, if a certain user class is not supported by a product. A use case as whole is a variant if it is not supported by some products in the SPL.

The use case variability type is defined from the requirements variability type. This definition must be based on the following rules:

1. If the use case corresponds to whole requirement, thus its variability type is the same of the requirement.
2. Otherwise, if the use case is common part (non-variant) of a mandatory requirement, then it is mandatory.
3. Otherwise, if the use case is variant part of any requirement, then it is variant.
4. Otherwise, if the use case is common part only of variant use cases, then it is variant.

For each use case are defined the attributes defined in the Table 4.7

Variability found in the use case element is represented by Variation Point (VP). A UC can have 0 to n VPs. The attributes of its VPs are defined as in requirements. A VP must be represented in the location where changes can take place.

The domain use cases elements are analyzed according to needs of each product. Thus, its behavior should be observed to identify common interactions between actors and systems. This information can be found with existing systems and assets, domain experts, end-users and so. Thus, the following guideline must be applied in the Task Elicit (see Section 4.2):

1. Identify the following elements for all products: actors, interaction between actors and system, preconditions and postconditions, rationale.
### Table 4.7 Use Cases attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>All use cases have unique identifier that can be represented as UCX, where X is a sequential number.</td>
</tr>
<tr>
<td>Name</td>
<td>The name that better represent the use case.</td>
</tr>
<tr>
<td>Variability Type</td>
<td>Mandatory or Variant.</td>
</tr>
<tr>
<td>Binding time</td>
<td>Time at which the decision for a variant use case is bound.</td>
</tr>
<tr>
<td>Rationale</td>
<td>It defines the basis for existence of the use case. Its description is optional.</td>
</tr>
<tr>
<td>Actors</td>
<td>It defines the actors that participate in the use case. There is always a primary actor that initiates the use case. In addition, secondary actors may participate in the use case.</td>
</tr>
<tr>
<td>Dependency</td>
<td>This optional attribute describes whether the use case includes or extends another use case.</td>
</tr>
<tr>
<td>Preconditions</td>
<td>States that must be true before the Use Case starts.</td>
</tr>
<tr>
<td>Main Flow</td>
<td>It describes in narrative manner a typical run of the use case, i.e. the most usual sequence of interactions between the actors and the system, where nothing goes wrong. The description is in the form of the input from the actor, followed by the response of the system. The system is treated as a black box - that is, dealing with what the system does in response to the actor’s inputs, not the internals of how it does it.</td>
</tr>
<tr>
<td>Alternative Flow</td>
<td>It describes a different path from the main flow, but where nothing goes wrong. There may be several alternative paths, i.e. an use case can have one or more alternative flows, which must be identified by an identifier (e.g. AF1).</td>
</tr>
<tr>
<td>Exception Flow</td>
<td>It describes what happens when things go wrong at the system level, occuring deviation in the Main or Alternative Flow. There may be several Exception Flows, which must be identified by an unique identifier (e.g. EF1).</td>
</tr>
<tr>
<td>Post conditions</td>
<td>This attribute identifies the condition that is always true at the end of the use case if the main flow has been followed.</td>
</tr>
<tr>
<td>Implication</td>
<td>It presents dependency relationship between use case and another variant object (UC or UC element).</td>
</tr>
<tr>
<td>Exclusion</td>
<td>It means that an use case cannot exist in presence of another object (UC or VP).</td>
</tr>
</tbody>
</table>
2. Identify synonyms and homonyms elements. For example, actors with different names but the same role (e.g. client and customer). This information must be described in the glossary.

3. Identify common responsibilities among similar actors and become them in an abstract actor (e.g. configuration manager and webmaster actors become an abstract administrator actor) (Griss et al., 1998).

4. Identify common and variable elements among products. The variabilities found are represented according to VP notation presented.

The variability modeling in UCs is based on PLUS (Gomaa, 2005). VPs in UCs can be handled in one of the two ways: small and complex variations. For small variations, the VP is described in the UC itself, identifying the place in the UC where the change can occur (see example in Figure 4.13). When a UC becomes too complex because it has too many variant and exceptional sequences of interactions, dependencies among UCs can be defined by include and extend relationships.

All use cases are specified in natural language text in terms of XML tags. The Figure 4.13 presents this notation in an use case with small variations.

According to (Gomaa, 2005), using include and extend relationships to handle variations in a product line works better when the use case contains a block of functionality that can be described as a sequence of interactions between an actor and the system. In some situations, however, a small variation affects only one or two lines in the use case descriptions. Trying to address those situations in separate use cases is liable to fragment the use case model, leading to several small use cases. Next, it is presented more detail about extend and include relationships.

**Extend Relationship.** An extend relationship indicates that an instance of an use case may be increased by the behavior specified by another use case. For complex use cases an extend relationship can be created splitting variations into a separate use case. The purpose of this new use case is to extend the old use case, if the appropriate conditions hold. The use case that is extended is referred to as the base use case, and the use case that does the extending is referred to as the extension use case. Under certain conditions, a base use case can be extended by a description given in the extension use case. It is important to note that the base use case does not depend on the extension use case and can function independently. The extension use case, on the other hand, executes only if the condition in the base use case that causes it to execute is true. The extension use case cannot function without the presence of a base use case. For representing the
<uc id="UC1">
  <name>Front door unlock</name>
  <variability-type>Mandatory</variability-type>
  <rationale>Controlling home access by authentication.</rationale>
  <actors>Inhabitant</actors>
  <precondition>Inhabitant registered in the system</precondition>
  <main-flow>
    <f>Inhabitant approaches the front door.</f>
    <f>System requests authentication.</f>
    <f>[UC1.VP1] [AF1]</f>
    <f>System permits entry to the home. [EF1]</f>
  </main-flow>
  <alternative-flow id="AF1">
    <f>Inhabitant cancels operation.</f>
  </alternative-flow>
  <exception-flow id="EF1">
    <f>System does not permit entry to the home.</f>
    <f>Inhabitant is not registered.</f>
    <f>A error message is shown to user.</f>
  </exception-flow>
  <postcondition>The front door is unlocked and inhabitant can access the home.</postcondition>
  <vps>
    <vp id="UC1.VP1">
      <description>Access types</description>
      <cardinality min="1" max="1"/>
      <variant id="UC1.VP1.V1">Inhabitant enters the PIN.</variant>
      <variant id="UC1.VP1.V2">Inhabitant touches the fingerprint sensor.</variant>
    </vp>
  </vps>
</uc>

Figure 4.13 Use Case with small variations
precise place in the base use case where extensions can be added, an extension point is used with the tag extension-point. The Figure 4.14 and 4.15 present an example of extend relationship in a domain of sales. These examples are presented in another domain because it is of easy understanding.

Figure 4.14 Base Use Case with complex variation represented by extension point.

**Include Relationship.** The include relationship indicates that an instance of an use case will contain the behavior of another use case. Interactions sequences can reflect functionality that is common to more than one use case. A common interaction sequence can be extracted from several use cases and made into a new use case that can then be reused. This new use case, which is called inclusion use case, is usually abstract, i.e., it cannot be executed on its own. An abstract use case must be executed as part of a concrete, i.e. executable use case. An abstract use case might not have a specific actor. The actor is in fact the actor of the concrete use case that includes the abstract use case, so it is possible for an abstract use case to be used by different actors from different concrete use cases. An example of include relationship between abstract and concrete use case is presented in the Figures 4.16 and 4.17.

In UCs, the dependencies can occur between the objects in each of the following pairs: UC-UC, UC-VP, VP-VP VP-variant, UC-variant and variant-variant. However, a
CHAPTER 4. RIPLE-RE PROCESS  4.5. DEFINE USE CASES

Figure 4.15 Extension use case (it extends base use case).

Figure 4.16 Abstract use case.
4.6 Verify

Verification is concerned with checking a final draft of the DRS (feature model, domain requirements or domain use cases), which must be reviewed by the domain experts and other potential stakeholders. At least, one stakeholder who did not participate in the RE activities should verify the DRS to determine the generality and applicability of the DRS. Domain experts are essential to verify the DRS accuracy and completeness. Domain analyst is the team member that understands more about the SPL scope, so he can evaluate whether the requirements are according to it. Potential users can be significant reviewers to evaluate scenarios and functionalities. Software Quality Assurance (SQA) is the expert in standard, so he can assure the standardization of the RE work products. Requirement Analyst is essential to check the consistency and traceability of the requirements, but he cannot verify the DRS modeled by himself to avoid tendentious verification.

The following stakeholders (when they exist in the project) are recommended for each DRS view:

- Feature Model: Domain expert, Domain analyst, Requirement analyst, SQA.
- Domain Requirements: Domain expert, End users, Requirement analyst, SQA.
• Domain Use cases: Domain expert, End users, Requirement analyst, SQA.

The verification objective is to analyze incompleteness, inconsistency, ambiguity, traceability and standardization in the DRS. Inconsistencies in the DRS should be identified prior to the development of other reusable assets in the SPL. If an inconsistency is not detected in the DRS, other domain assets are developed based on an inconsistent DRS. If the inconsistency is detected late in product derivation, not only the requirements have to be adjusted (to resolve the inconsistency) but also all SPL assets affected by the inconsistency must be adjusted. All the products derived from the SPL benefit from the consistency checking of the DRS (Lauenroth and Pohl, 2008).

It is essential to instantiate the DRS for at least one product from the SPL and checking if the instantiated specification correctly represents the product. Instantiated DRS is useful to verify the consistency, completeness and ambiguities.

After each verification, it must be generated a report with the problems found. This report must be generated by the reviewer, describing all reviewed items and found problems. Each problem must be identified with the item, problem type (incompleteness, inconsistency, ambiguity, traceability and standardization), problem severity (critical, medium, trivial) and problem cause (lack of experience, lacks in input documentation, lacks in documentation standards, omissions in analysis and fault in input documentation). Critical problems are faults that may interrupt the operation of the core assets (e.g. components). Medium problems are faults that will cause the core assets to operate in a degraded mode. Trivial problems may be defined as those that will not affect the core assets operation.

This report is essential to evaluate the errors distribution, identifying the causes of the defects and analyzing their severity. Therefore, it maintains the verification history that aids the organization to obtain expertise in RE for SPL projects, identifying improvement for the weaknesses found.

The following checklist is made available to DRS verification.

**Completeness.** Is the DRS complete, it is, does the checker know any of missing requirements, features or use cases or is there any information missing from their individual descriptions? It is also related to the following criteria:

• Is the DRS comprehensible, it is, can readers of the document understand what the features, requirements or use cases mean? Is the granularity level right? Insufficient generality in the requirements leads to a design that is too brittle to deal with the change actually experienced over the lifetime of the product line (Clements, 2001).
Abstract DRS comes with a high level of adaptation effort in product derivation. Thus, DRS should have a granularity level useful to use.

- Is the derived product requirements specification useful? It is important to check if the DRS capture all domain variabilities accurately, and if derived DRS represents the problem that the corresponding SPL member solves.

**Consistency.** Is the DRS consistent, it is, does the descriptions of different features, requirements and use cases include contradictions? The specification may contain inconsistencies among different work products of the DRS (additional complexity of variability modeling), for example, inconsistency between features and requirements, or between requirements and use cases. Thus, the consistency analysis restores the relationships among work products when one of these work products is updated. The following check items must be used:

- **Contraction.** The requirements $A$ and $\neg A$ are therefore part of the DRS of this SPL. The consistency check of this DRS would identify the contradiction between $A$ and $\neg A$ and would indicate that the DRS is inconsistent (Lauenroth and Pohl, 2008). A derived product cannot have $A$ and $\neg A$.

- **Constraints.** In each model, evaluating whether constraints have conflicts. For example, a mandatory object cannot require a variant object.

- **Priorities.** Are there conflicting priorities among requirements?

**Ambiguity.** Is the DRS ambiguous, it is, are there different possible interpretations of the features, requirements and use cases? It must compare ambiguous DRS points with different stakeholders to investigate possible interpretations.

**Traceability.** Is the DRS traceable, it is, are features, requirements, requirement variable part, use cases and requirement variable part unambiguously identified, do they include links to related features, requirements, requirement variable part, use cases and requirement variable part? The following check items must be used:

- Check if all use cases are related to functional requirements.

- Check if all composition relationships in the domain use cases and requirements are traceable. For example, whether a variation point is part of an use case then is necessary trace link among them.
• Check if all external associations (among objects of different work products of the DRS) are traceable. For example, which use cases are directly related to a requirement, and which requirements are directly related to a feature.

• Check if associated features are traceable.

• Check if all associated requirements are traceable.

• Check if all associated use cases are traceable.

**Standardization.** Is the DRS conform to defined standards? Some additional check items are defined for this validation, as follows:

• Check if all work products are defined according to template.

• Check if all features, requirements and use cases mandatory attributes are defined.

• Check if all variabilities are correctly represented.
  
  – All VPs have at least one variant
  
  – Implication or exclusion rules are defined only from variant.
  
  – Binding time is defined only for variant.
  
  – All VPs with more that one variant have cardinality defined.

• Check if all non-functional requirements (quality attributes) are quantified.

After verification, a set of work products may be added, deleted or refined, so change requests can be necessary according to the baseline. Therefore, the possible outputs of this task are the verification report and change requests.

Next task presents guidelines for evolution management related to RE.

### 4.7 Evolution Management

The SPL evolution control is ensured by appropriate practices of change management, including changes in requirements. These practices are part of the Evolution Management process (EM), thus it will not be considered in this process. However, this work presents the evolution context in RE and proposes some guidelines for it.

Evolution can occur by several reasons, such as new customer needs, regulation changes, technical progress, developers’ increased understanding of the product or domain, business pressure, opportunities and/or change in business goals. During the
development of a product (member of the SPL), requirements often arise that are not in
the scope of the SPL. Hence, the scope of the product line must be either expanded or
product-specific requirements must be created.

These changes can impact different areas, such as business (e.g. price), system
(e.g. functionality, quality attribute, technological constraints) and project (e.g. release,
schedule). Changes also can have conflict among them, so their decisions can need of
negotiations among different stakeholders.

In DRS, some options of changes are:

- New feature, requirement or use case.
- New variation point
- New variant in variation point
- Turn a variant or variation point into an obsolete (remove).
- Turn a feature, requirement or use case into an obsolete.
- Mandatory feature changes into optional, and vice versa.
- Mandatory requirement changes into variant, and vice versa.
- Mandatory use case changes into variant, and vice versa.
- New cardinality, how many variants can be applied to the variation point.
- Turn a high priority into a low or medium.
- Turn a medium priority into a high or low.
- Turn a low priority into a medium or high.
- New measurable value of the quality attribute.
- Detail use case or requirement.

For analyzing a change in requirements, the requirements analyst must check the
change impact in the SPL, which can be verified by traceability links that identify
relationships among different assets. The following items must be checked:
• Conflicts among new requirements (including features and use cases) and core assets. The core assets must be taken into account, analyzing the impact of new requirement in the SPL. For example, requirements which do not conform to the architecture can only be implemented with high effort and modification of the reuse infrastructure or even cannot be implemented at all.

• Conflict in quality attributes. New constraints, functionalities or non-functional requirement can affect existing quality attributes.

• Priority conflicts. Changes in requirement can affect existing priorities.

• Changes can impact the dependency relationship (implication and exclusion) among objects of the DRS.

• New variations can affect existing variation points (e.g. cardinality, variability type).

• New requirements (including features and use cases) or variants can replace others.

After analyzing changes, it can be necessary to negotiate with different stakeholders for solving conflicts, and the accepted decisions must be submitted to EM. In addition, change requests must have change description, rationale, and cause type. The cause types can be:

• Incompleteness. Missing requirements, features or use cases or there any information missing from their individual descriptions (see verification checklist in Section 4.6).

• Defect. It is with relation to inconsistency, ambiguity, traceability, standardization (see validation checklist in Section 4.6).

• Improvement. Evolution over time.

Requirements management requires traceability information to be recorded. Thus, next section presents the guidelines for traceability regarding to the requirements.

### 4.7.1 Traceability

Links are established that explicitly model interrelationships among information in the different work products. The traceability objectives are:
• determine the impacts of proposed changes (Ramesh et al., 1997);

• link variabilities among different SPL assets, make easy future decisions in the products instantiation; and

• demonstrate that each requirement has been satisfied (Ramesh et al., 1997).

This process suggests a traceability modeling in which are defined two entities: *Trace* and *Object*. *Trace* entity represents the traceability link between two objects (Object A and Object B). An *Object* can have many trace links. In RE, a *Object* can be a UC, a Requirement, a Feature, or a Variable Part of UC or Requirement. The Variable Part corresponds to a VP or a variant within specification (e.g. text fragment, actors, pre conditions, and so on). Thus, the relationship among *Objects* is a many-to-many association. The traceability metamodel is presented in the Figure 4.18.

![Figure 4.18 Traceability metamodel.](image)

All RE work products must be traceable. However, traceability must be maintained only in the level of direct trace link. Indirect traceability is found by inference from objects trace links. Some examples can be observed:

- A use case UC1 has direct trace link with its variable part VP1. Whether VP1 is directly traced with a feature F1, then UC1 is indirectly traced with F1. It represent a transitive relationship, \( T = (UC1,VP1) \) and \( (VP1,F1) \rightarrow (UC1,F1) \).
• Whether requirement R1 is directly traced with feature F2, and use case UC3 is directly traced R1, the F2 is indirectly traced with UC3. T = (UC3, R1) and (R1, F2) → (UC3, F2)

All Traces are characterized by the attribute relationship type. The following relationship types are defined:

• **External Association.** This relationship expresses external association among different work products, i.e. relationship between feature and requirement, feature and requirement variation point, feature and requirement variant, feature and UC, feature and UC variation point, feature and UC variant, requirement and UC, UC and requirement variation point, or UC and requirement variant. For example, requirement User Maintain is associated with feature Authentication. It implicates that when feature Authentication is bound the requirement User Maintain also is automatically bound.

• **Composition.** This relationship is used to express the whole-part relationship between requirement and its variation point, use case and its variation point, or variation point (of the requirement or UC) and its variant (whether VP has more than one variant). In the relation (A, B), the object A is the parent-role and object B is the child-role.

• **Implication.** This relation means that it must make a choice about the variable object A only if an object B is chosen. This relationship type is defined only among objects of a same work product, i.e. feature and feature, requirement and requirement, requirement and requirement variable part (variant or VP), requirement variable part and requirement variable part, UC and UC, UC and UC variable part, and UC variable part and UC variable part. In the relation (A, B), the object A is the requester and the object B is the required object. For example, Login/Logout (A) requires User Maintain (B).

• **Exclusion.** This relation means that it must make a choice about a variable object A only if object B is not chosen. This relationship type is defined only among objects of a same work product, in the same manner that Implication. In the relation (A, B), the object A is the excluder and the object B is the excluded object.

• **Internal Association.** It is defined for any another direct relationship among particular objects of a same work product that are not related by implication,
exclusion and composition. This relationship type can be defined between feature and feature, UC and UC, or requirement and requirement.

This traceability model enables to link variabilities in different work products, so it makes easier the products derivation. The selection of variability can be analyzed according to its trace links, identifying the impact of each decision.

### 4.7.2 Traceability Tool

In order to support the proposed traceability model, a tool was implemented as product of this work: the **Software Product Line Traceability Tool (SPLiTT)**<sup>3</sup>. It enables the traceability among objects of different assets according to the guidelines of Section 4.7.1. It presents the following features:

- Create, retrieve, update and delete asset group (e.g. feature, requirement, use case, component, test case, class);

- Create, retrieve, update and delete asset type (e.g. feature, requirement, variant of requirement, VP of requirement). An asset type composes an asset group (e.g. requirement, variant of requirement and VP of requirement compose the requirement asset group);

- Create, retrieve, update and delete asset (e.g. RF1 - door control);

- Create, retrieve, update and delete trace link according to the rules for External Association, Composition, Implication, Exclusion and Internal Association presented in previous section.

In this process, the tool is used to trace links among the objects in the Model Scope (Section 4.3), Define Requirements (Section 4.4) and Define Use Cases (Section 4.5) activities.

### 4.8 Chapter Summary

This Chapter presented the proposed process to support requirements engineering for software product lines. The process is composed of a set of activities, tasks, guidelines, roles, inputs and outputs addressed to treat peculiar aspects of this context.

---

<sup>3</sup>SPLiTT can be accessed at [http://jaqueira.cin.ufpe.br/dfs](http://jaqueira.cin.ufpe.br/dfs).
According to the defined criteria and the analyzed approaches in the systematic review (Chapter 3), the RiPLE-RE presents a result more satisfactory than the other approaches, as can be seen in Table 4.8. Its activities cover the whole RE lifecycle and are defined in a systematic way. The process integrates different models (features, requirements and use cases) to represent multiple viewpoints for the stakeholders. The requirements and use cases are specified in a structured way, enabling the their instantiation for SPL and product. Furthermore, the different models enable dealing with variabilities of high and low granularity. Regarding to its variability strategy, each model captures commonalities and variabilities, a traceability model is used to link decisions among different assets, so this strategy reduces the number of decisions that must be made.

Next chapter presents an experimental study in order to evaluate the viability of the use of the proposed process in software product lines projects.
### Table 4.8 Comparison with other approaches

<table>
<thead>
<tr>
<th>Approach</th>
<th>Activities</th>
<th>Disciplines</th>
<th>Does it define Roles?</th>
<th>Does it define Guidelines?</th>
<th>Does it define Inputs and Outputs?</th>
<th>Models and Techniques</th>
<th>Does it link decisions among different assets?</th>
</tr>
</thead>
<tbody>
<tr>
<td>FODA</td>
<td>Model Domain</td>
<td>Analysis, Specification, Validation, Elicitation</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Feature Model</td>
<td>No</td>
</tr>
<tr>
<td>FAST</td>
<td>Define Decision Model, Analyze Commonality</td>
<td>Specification, Analysis, Elicitation, Verification</td>
<td>Yes</td>
<td>Yes</td>
<td>Partially</td>
<td>Decision Model</td>
<td>No</td>
</tr>
<tr>
<td>FeatureB</td>
<td>Model Feature, Model Use Case</td>
<td>Specification, Analysis, Elicitation, Verification</td>
<td>No</td>
<td>Partially</td>
<td>Partially</td>
<td>Feature Model, Use Case</td>
<td>No</td>
</tr>
<tr>
<td>FORM</td>
<td>Identify Feature, Classify Features, Validate Model</td>
<td>Specification, Analysis, Elicitation, Verification</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Feature Model</td>
<td>No</td>
</tr>
<tr>
<td>VODRD</td>
<td>Characterize the domain, Document the viewpoints, Analyze the viewpoints</td>
<td>Specification, Analysis, Elicitation, Verification</td>
<td>No</td>
<td>Partially</td>
<td>Partially</td>
<td>Viewpoint</td>
<td>No</td>
</tr>
<tr>
<td>Odyssey</td>
<td>Model Feature, Model Use Case</td>
<td>Specification, Analysis, Elicitation, Management</td>
<td>Partially</td>
<td>Partially</td>
<td>Partially</td>
<td>Feature Model, Use Case</td>
<td>No</td>
</tr>
<tr>
<td>Pulse-CDA</td>
<td>Refine scope definition, Elicit domain knowledge, Model domain knowledge</td>
<td>All</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Use Case, Decision Model</td>
<td>Yes</td>
</tr>
<tr>
<td>DH</td>
<td>Represent definition hierarchy</td>
<td>Specification, Analysis</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Goal-based</td>
<td>No</td>
</tr>
<tr>
<td>Goal-oriented</td>
<td>Model Goal</td>
<td>Specification, Analysis</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Goal-based</td>
<td>No</td>
</tr>
<tr>
<td>PLA</td>
<td>Model Feature, Model Use Case, Model Object, Verify Models</td>
<td>Specification, Analysis, Elicitation, Management, Verification</td>
<td>Yes</td>
<td>Yes</td>
<td>Partially</td>
<td>Feature Model, Use Case</td>
<td>No</td>
</tr>
<tr>
<td>PRS</td>
<td>Organize the Domain Definition, Create a Decision Model, Encapsulate Variations in CoRE, Define variations, Provide Traceability</td>
<td>Specification, Management, Analysis</td>
<td>No</td>
<td>Partially</td>
<td>Partially</td>
<td>Decision Model, PRS case</td>
<td>No</td>
</tr>
<tr>
<td>DRAMA</td>
<td>Analyze Domain Requirement, Analyze Commonalities and Variabilties, Represent Domain Requirement</td>
<td>Elicitation, Analysis, Specification</td>
<td>No</td>
<td>Partially</td>
<td>Partially</td>
<td>Goal-based</td>
<td>No</td>
</tr>
<tr>
<td>PLUS</td>
<td>Model Requirements, Model Product Line Analysis, Test Product Line</td>
<td>Specification, Analysis, Elicitation, Verification</td>
<td>No</td>
<td>Yes</td>
<td>Partially</td>
<td>Feature Model, Use Case</td>
<td>No</td>
</tr>
<tr>
<td>PLUSS</td>
<td>Model Feature, Model Use Case, Manage Change</td>
<td>Specification, management, analysis</td>
<td>No</td>
<td>Partially</td>
<td>Partially</td>
<td>Feature Model, Use Case, Change Case</td>
<td>No</td>
</tr>
<tr>
<td>PR</td>
<td>Identify domain requirements, Refined domain requirements, Model domain use case</td>
<td>Elicitation, Analysis, Specification</td>
<td>No</td>
<td>Yes</td>
<td>Partially</td>
<td>Use Case, Prinve Requirement</td>
<td>No</td>
</tr>
<tr>
<td>SPLE</td>
<td>Analyze variability, Analyze Priority-based, Model Requirements, Define Traceability</td>
<td>All</td>
<td>No</td>
<td>Partially</td>
<td>Yes</td>
<td>Feature Model, Use Case, Origional Variability Model</td>
<td>Yes</td>
</tr>
<tr>
<td>RIPLE-RE</td>
<td>Model Scope, Define Requirements, Define Use Cases</td>
<td>All</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Feature Model, Functional and Non-Functional Requirements, Use Case, SPL Traceability Model</td>
<td>Yes</td>
</tr>
</tbody>
</table>
A process will evolve over time based upon feedback and learning from applying the ideas and analyzing the results. In this context, empirical studies are very important since the progress in any discipline depends on our ability to understand the basic units necessary to solve a problem (Basili, 1996).

In this sense, this chapter presents an experimental study in order to evaluate the viability of the RiPLE-RE in software product lines projects. Before discussing the experimental study defined, it is necessary to introduce some definitions to clarify its elements.

5.1 Introduction

An experimental study was performed to ensure the proposal viability and avoiding unexpected results. It is based on a rigorous and systematic experimental process (Wohlin et al., 2000).

According to Wohlin et al. (Wohlin et al., 2000), when conducting an experiment, the goal is to study the outcome when are varied some of the input variables to a process. These variables can be of two kinds: independent and dependent.

The variables that are objects of the study which see the effect of the changes in the independent variables are called dependent variables. Often there is only one dependent variable in an experiment. All variables in a process that are manipulated and controlled are called independent variables.

Experiments study the effect of changing one or more independent variables. Those variables are called factors. The other independent variables are controlled at a fixed level during the experiment, or else it would not be possible to determine if the factor or another variable causes the effect. A treatment is one particular value of a factor.
The treatments are being applied to the combination of objects and subjects. An object can, for example, be a requirement specification that will be constructed using different representation methods. The people that apply the treatment are called subjects. At the end, an experiment consists of a set of tests where each test is a combination of treatment, subject and object.

5.2 The Experimental Study

The experimental process is divided into the following activities, according to (Wohlin et al., 2000): the definition is the first step, where the experiment is defined in terms of problem, objective and goals. The planning comes next, where the design of the experiment is determined, the instrumentation is considered and the threats to the experiment are evaluated. The operation of the experiment follows from the design. In the operational phase, measurements are collected, analyzed and evaluated in the analysis and interpretation. Finally, the results are presented and packaged in the conclusion.

This experimental study is based on the model proposed in (Wohlin et al., 2000). Additionally, the experiment defined in (Almeida, 2007) was used as inspiration. The definition and planning activities will be described in future tense, showing the logic sequence between the planning and operation.

5.2.1 The Definition

The definition determines the foundation for the experiment, presenting why the experiment will be conducted. In order to define the experiment, the Goal Question Metric (GQM) paradigm (Basili et al., 1994) was used. The GQM is based upon the assumption that for an organization to measure in a purposeful way it must first specify the goals for itself and its projects, then it must trace those goals to the data that are intended to define those goals operationally, and finally provide a framework for interpreting the data with respect to the stated goals.

According to (Basili et al., 1994), the result of the application of the GQM is the specification of a measurement system targeting a particular set of issues and a set of rules for the interpretation of the measurement data. The resulting measurement model is composed of three levels:

- **Goal:** a goal is defined for an object, for a variety of reasons, with respect to various models of quality, from various points of view, relative to a particular environment;
• **Question**: a set of questions is used to characterize the way the assessment/achievement of a specific goal is going to be performed based on some characterizing model; and

• **Metric**: a set of data is associated with every question in order to answer it in a quantitative way.

The next section presents the experiment definition using the GQM paradigm.

### 5.2.1.1 Goal

$G_1$. To analyze the requirements engineering process for the purpose of evaluating it with respect to its usability, effectiveness and efficiency from the point of view of the researcher in the context of software product lines project.

### 5.2.1.2 Questions

To achieve this goal, we defined quantitative and qualitative questions. The first ones are related to the data collected during the period that the experiment will be executed, and the last ones concerned with the subjects’ feedback about the new process adoption. The questions are described according to the experiment quality focus, which is the usability (if the process is easy to use), effectiveness (how effective the process is, whether it does what is hoped) and efficiency (how efficient the process is, whether it produces results with little wasted effort), as follows.

* **Usability.** $Q_1$. Do the subjects have difficulties to understand or apply the process? $Q_2$. Are the subjects satisfied in using the process?

* **Effectiveness.** $Q_3$. Does the process aid to generate domain requirements specification in compliance with the process representation standards?

* **Efficiency.** $Q_4$. How much effort does it take to apply the process?

### 5.2.1.3 Metrics

The previous questions are answered according to metrics.

According to (ISO/IEC TR 9126, 2000), the usability can be measured by understandability and subjects’ satisfaction metrics, which will be assessed informally, examining the opinion of each subject. Thus, the following qualitative metrics are proposed.
$M_1$. In order to identify possible misunderstanding or applicability problem of the process, it is necessary to identify and analyze the difficulties found by users learning and applying the process.

$Diff =$ subjects that had difficulties to understand or apply the process and the difficulty distribution.

$M_2$. Satisfaction is the user’s response to interact with the process, and it includes attitudes towards the use of the process. Thus, this metric is proposed to evaluate subjects’ satisfaction in using the process.

$Sat =$ subjects’ satisfaction distribution according to a defined scale: very satisfied, satisfied, impartial, unsatisfied, very unsatisfied.

The effectiveness is evaluated regarding to process potential to generate requirements in compliance with its standards. Thus, it will adopt the quantitative metrics Defect Index and Error Distribution (IEEE STD 982.2, 1988).

$M_3$. Error distribution allows ranking of the predominant failure modes (IEEE STD 982.2, 1988). The search for the causes of failures involves the error analysis of the defect data collected during each phase of the software development. According to (Li and Smidts, 2003), it is an of the top measures to requirements phase. It is defined as follows:

$ED =$ error distribution, i.e., number of errors by cause category, which can be lack of experience, lacks in input documentation, lacks in documentation standards, omissions in analysis, and faults in input documentation.

Thus, in this aspect, the process effectiveness is analyzed regarding to error causes proportion directly related to the process. We hope that the majority of the error causes are directly related to lack of experience and omissions in analysis of the subjects.

$M_4$. The Defect Index measures the average severity of the defects found. It is based on (IEEE STD 982.2, 1988) and applies the following primitives:

$d =$ Total number of defects detected during the RE phase.

$c =$ Number of critical defects found.

$m =$ Number of medium defects found.

$t =$ Number of trivial defects found.
\( w_1 = \) Weighting factor for critical defects (value is 10).
\( w_2 = \) Weighting factor for medium defects (value is 3).
\( w_3 = \) Weighting factor for trivial defects (value is 1).

\[ DI = w_1 \frac{c}{d} + w_2 \frac{m}{d} + w_3 \frac{t}{d} \]

Its interpretation is based on the magnitude of the defect index: the smaller the number, the better. The boundaries of DI are \( 0 \leq DI \leq W_1 \). The baseline number of the Defect Index should be computed from past projects (IEEE STD 982.2, 1988). However, it is the first project using the process and there is not a base value as reference. Therefore, the process effectiveness will be analyzed regarding to weighting factor. If the value found is nearer of the weighting factor for critical defects (> 6.5) than of the others, then it is considered as an indicative that the process needs be improved.

The efficiency is measured regarding to effort in the use of the process, according to the following quantitative metric.

\[ M_5 \] Mid effort (hours spent) with Requirements Engineering, measured regarding to whole project, i.e. the sum of worked hours of every subject in the whole project lifecycle.

\[ Eff = \frac{TotalHoursSpentwithRE}{TotalTimeSpentintheProject} \]

In traditional software development with large software, about 15% of the total budget is taken up by RE activities (Kotonya and Sommerville, 1998). This excludes the costs of detailed system specification. For smaller systems which are mostly software, the requirements costs are usually less than this, around 10% of the total budget of the system (Kotonya and Sommerville, 1998). In a SPL, the quantity of products is major and it has variabilities and more requirements, so the value below 30% is acceptable, including all RE activities.

### 5.2.2 The Planning

After the definition of the experiment, the planning is started. The definition determines the foundations for the experiment, the reason for it, while the planning prepares for how the experiment is conducted.

The planning phase is divided into following steps. **Context Selection**, i.e. environment in which the experiment will be executed. Next, the **Hypothesis Formulation** and
the Variables Selection take place. The Selection of Subjects is carried out. Next, the Instrumentation prepares for the practical implementation of the experiment. Finally, the Validity Evaluation aims at checking the validity of the experiment.

5.2.2.1 Context Selection

The experimentation will be conducted in an university lab that simulates a software factory, as part of a post-graduation course (Software Reuse: Theory and Practice) at Federal University of Pernambuco, Brazil (Lisboa et al., 2008). It consists of a project to build a SPL in the rental domain. Initially, the subjects will be trained in several aspects of SPL and in the applied processes and next, they will perform the SPL project.

Thus, the experiment context is characterized as being an off-line study (not industrial project), based on real-world problems and staffed by M.Sc. students and Ph.D. students. It will be conducted as single object study, i.e., it examines an object on a single team and a single project.

The SPL project will be performed with support of the RiPLE and RiDE (Almeida, 2007) processes. The RE activities will be performed according to the RiPLE-RE, object of study of this experiment. As pre-conditions for running this object of study, the SPL scope must be defined.

5.2.2.2 Hypot:heses Formulation

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., 2000). Two hypothesis types have to be formulated: Null Hypothesis, which the experimenter wants to reject with as high significance as possible; and Alternative Hypothesis, which is in favor of which the null hypothesis is rejected.

In this study, the null hypotheses determine that the use of the process in SPL projects does not produce benefits that justify its use, presenting a poor usability, effectiveness and efficiency. Thus, the following hypotheses are defined:

\[ H_0_1 : \mu_{ErrorsDueToExperienceAndOmissions} < 50\% \]
\[ H_0_2 : \mu_{DI} > 6.5 \]
\[ H_0_3 : \mu_{Eff} \geq 30\% \]

The alternative hypotheses determine that the use of the process produces benefits that justify its use, presenting a good usability, effectiveness and efficiency. Thus, the following hypotheses are defined:

\[ H_1_1 : \mu_{ErrorsDueToExperienceAndOmissions} \geq 50\% \]
\[ H_{12} \mu_{DI} \leq 6.5 \]
\[ H_{13} \mu_{Eff} < 30\% \]

5.2.2.3 Variables Selection

In the variables selection, we choose the independent and dependent variables. All variables in a process that are manipulated and controlled are called independent variables. In this study, the independent variables are the experience of the subjects, the SPL requirements, and the proposed process.

The dependent variables are the variables that we want to study to see the effect of the changes in the independent variables. In this study, the dependent variables are the usability, effectiveness and efficiency of the proposed process.

5.2.2.4 Selection of Subjects

The subjects of the study are M.Sc. students and Ph.D. students. They will be requested to act as the roles defined in the process according to their experience and interest. However, during the project, one subject could have more than one role, as well as the same role could be undertaken by more than one subject.

Hence, the subjects are chosen based on convenience, i.e., the subjects are students taking the course.

5.2.2.5 Experiment Design

A design of an experiment describes how the tests are organized and run. The general design principles are randomization, blocking and balancing.

Randomization. This technique can be used in the selection of the subjects. The object is not assigned randomly to the subjects. They are the subjects who chose to take the course. Thus, the assignments order is not important, since the objective of the study is not to evaluate the RiPLE-RE vs. something else. The measures used in the evaluation are the results of performing the RE activities.

Blocking. Blocking is used to systematically eliminate the undesired effect in the comparison among the treatments. In this study, it was not identified the necessity of dividing the subjects into blocks, since the study will evaluate just one treatment, the value of the factor is unique, which is the RiPLE-RE process.

Balancing. In some experiments, balancing is desirable because it both simplifies and strengthens the statistical analysis of the data. However, the experimental study is
based on a course, so we have been unable to influence the background of participants who have signed up for the course, and consequently unable to balance the data set.

5.2.2.6 Instrumentation

In order to guide the participants in the experiment, we will provide the process complete description, with all support material, such as templates, checklists and available tools. The participants also will be trained. The training of the subjects using the process will be conducted in a classroom at the university. The training will be divided in two steps: in the first one, concepts related to software reuse, variability and SPL will be explained. Next, the RiPLE-RE process will be discussed.

Measurement instruments will be used for data collection. All the subjects will receive a questionnaire (QT1) to evaluate their educational background, participation in software development projects, experience in RE and reuse. In addition, the subjects will receive a second questionnaire (QT2) for the evaluation of subjects’ satisfaction and difficulties using the proposed process. These questionnaires compose the Appendix A. To guarantee more precision for the data collected, specially data related to the effort, the subjects will be oriented to use timesheet to register the time elapsed while performing the experiment. The subjects also will use verification reports (see Appendix B) to register the problems found, aiding to analyze the data related to process effectiveness.

Before performing the study, a pilot project will be conducted with the same structure defined in this planning. The pilot project will be performed by a single subject, who will be trained on how to use the proposed process. For this project, the subjects will be observed by the responsible researcher. In this way, the pilot project will be a study based on observation, aiming to detect problems and improve the planned material before its use.

5.2.2.7 Validity Evaluation

A fundamental question concerning results from an experiment is how valid the results are. In this study, we have the four types of threats to validity the experiment, as follows.

Internal Validity. Threats to internal validity are influences that can affect the independent variable with respect to causality, without the researcher’s knowledge (Wohlin et al., 2000). In this study, the following threats to internal validity are considered:

• Maturation. This is the effect that subjects react differently as time passes. Some subjects can be affected negatively (tired or bored) during the experiment, and their
performance may be below normal. They also can be affected positively during the course of the experimentation, increasing their learning curve. Subjects’ grade will depend of their participation in the whole project as an attempt to overcome this threat. Furthermore, the subjects who performed the experiment are volunteers, so they should have at least some interest in the study. About the increase of the learning, we do not consider this threat important for the experiment, because there is a single treatment;

- **Mortality.** If too many people leave the study, this can be a threat to the validity. We have not seen any systematic trend in people leaving the course. In our knowledge, people who left the course did this independently of the used process. Therefore, this threat is not considered important for the experiment.

**External Validity.** Threats to external validity are conditions that limit out ability to generalize the results of our experiment to industrial practice (Wohlin et al., 2000), as follows:

- **Generalization of subjects.** The study will be conducted with M.Sc. students and Ph.D. students who normally have similar knowledge in RE. Thus, the subjects will not be selected from a general population. In this case, if these students succeed using the process, we cannot conclude that a practicing requirements analyst would use it successfully too. On the other hand, negative conclusions have external validity, i.e. if the students fail in using the process, then this is strong evidence that a practicing requirements analyst would fail too;

- **Generalization of scope.** The experiment will be conducted on a defined time according to the schedule of the course, which could affect the experiment results. The SPL scope will be defined according to this schedule to guarantee the complete execution of the project. It also will depend on the students productivity. Thus, this scenario could have a toy size that will limit the generalization. However, negative results in this scope is a strong evidence that in a bigger scope would fail too.

**Conclusion Validity** is concerned with the relationship between the treatment and the outcome, and determines the capability of the study to generate conclusions (Wohlin et al., 2000). Thus, the following threats to conclusion validity are considered:

- **Heterogeneity of subjects.** A group with very heterogeneity presents a risk to the conclusion validity, since the variation due to individual differences is larger than due to the process. This study is composed by M.Sc. students and Ph.D. students
who normally have similar knowledge and background, which aid to reduce the heterogeneity. On the other hand, it also reduces the validity external of the experiment, since the subjects are not selected from a general enough population.

- **Experience of the subjects.** Subjects without experience also can affect this validity, since it is harder for them to understand the process. To mitigate the lack of experience, we will provide training in SPL and requirements engineering for SPL.

- **Reliability of measures.** Once the measurement instrumentation is bad, it can bring us no reliable data. In order to mitigate this threat, the measurement instruments will be validated with three members of the RiSE group, in order to find possible errors;

- **Participation of subject in the pilot project.** One subject of the experiment participated in the pilot project, so it can affect the conclusion validity, since he can perform the process based on the experience previously gained from the project. In order to mitigate this threat, we will notify the subject about this risk and will request his help to not influence other subjects.

**Construct Validity** concerns generalizing the results of the experiment to the concept or theory behind the experiment (Wohlin et al., 2000), as follows:

- **Data Bias.** The experiment is part of a course, where the students are graded. This implies that the students may bias their data, as they believe that it will give them a better grade. In order to mitigate it, in the beginning of the course, we will emphasized that the grade depends of subjects’ participation and effort, timely and proper delivery, analyzing also if it is in compliance with the process.

### 5.2.3 The Operation

In the operation phase, the treatment (RiPLE-RE process) is applied to the subjects. This phase consists of three steps: **preparation** where subjects are chosen and informed and the material is prepared, **execution** where the subjects perform their tasks according to the treatment and the data is collected, and **data validation** where the collected data is validated.

#### 5.2.3.1 Preparation

The subjects were seven M.Sc. students and two Ph.D. students. They correspond to all students taking the reuse course in the postgraduate curriculum, representing a non-
random subset from the universe of students from Software Engineering. Thus, they were
volunteers with at least some interest in the study.

The students were informed that we would like to investigate the outcome of the
process application. However, they were not conscious of what aspects we intended to
study, i.e., they were not aware of the hypotheses stated.

Before the experiment can be executed, all experiment instruments must be prepared
and ready. Thus, all instrumentation defined in Subsection 5.2.2.6 were provided. Addition-
ally, the subjects could use any tools and environments for RE, except by traceability,
that we provide a tool according to the process guidelines.

5.2.3.2 Execution

The experimental study was conducted during the second semester of 2008, from August
to December, within a postgraduate course at Federal University of Pernambuco. Initially,
the subjects were trained in several aspects of SPL and in the applied processes (from
August to October in 2008), and after, they performed the SPL project from October to
December in 2008.

Most of the students had participated in industrial projects. However, the subjects had
low or none industrial experience in reuse activities, such as component development
and SPL development. On the other hand, seven participants are members of the RiSE
group, and their research area involve these aspects, which give them theoretical know-how.
Regarding to RE, the majority has the theoretical knowledge, but low industrial practice.
Table 5.1 shows a summary of subjects’ profile.

In the initial phase of the project, roles were distributed among the subjects, according
their experience as well as interest. Considering the reduced number of members, only
the essential roles were used according to adopted processes, as follows: SPL Manager,
Domain Analyst, Requirements Analyst, Architect, Configuration Manager, SQA and
Developer. Regarding to RE main role, there were two Requirement Analysts, but the
whole team worked with the requirements activities.

As pre-conditions to start RE, activities regarding to the scoping were performed.
The team analyzed four products in the rental domain (library, car rental, movies rental
and real estate), through interviews with domain experts. During scoping, the domain
features were identified and a product map was generated with the features for the SPL.

After scoping, the RiPLE-RE was applied. A feature model was created with the
support of a tool - ToolDay (Lisboa et al., 2007), representing 55 features: 20 optional,
11 or and 24 mandatory. Based on the feature model, the requirements were elicited and
described, representing 38 functional requirements and 15 non-functional requirements.

However, due to the time constraints for the project development phase, the team did not implement all requirements. In this way, the team described only the relevant use cases for the set of prior requirements, resulting in 15 specified use cases, which were derived from 13 requirements.

### 5.2.3.3 Data Validation

Data was collected from 9 subjects. However, data from 2 subjects (ID 4 and 9, see Table 5.1) was removed, due to that they did not participate of the whole study and they did not answer the questionnaire related to satisfaction and difficulties in the use of the process. Thus, this may have affected the data validation. Therefore, we had 7 subjects for statistical analysis and interpretation of the results.

<table>
<thead>
<tr>
<th>ID</th>
<th>Year since gratuation</th>
<th>Participation in Industrial Projects</th>
<th>Experience in RE</th>
<th>Experience in Software Reuse</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>1 medium complexity 4 high complexity</td>
<td>academic - low commercial - medium</td>
<td>academic - none commercial - none</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>5 low complexity 2 medium complexity</td>
<td>academic - medium commercial - medium</td>
<td>academic - low commercial - low</td>
</tr>
<tr>
<td>3</td>
<td>1.7</td>
<td>3 low complexity 1 medium complexity</td>
<td>academic - medium commercial - low</td>
<td>academic - medium commercial - low</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>1 low complexity 3 medium complexity 3 high complexity</td>
<td>academic - medium commercial - medium</td>
<td>academic - medium commercial - low</td>
</tr>
<tr>
<td>5</td>
<td>0.8</td>
<td>-</td>
<td>academic - medium commercial - none</td>
<td>academic - medium commercial - none</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>-</td>
<td>academic - medium commercial - low</td>
<td>academic - low commercial - none</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>1 high complexity</td>
<td>academic - low commercial - low</td>
<td>academic - medium commercial - low</td>
</tr>
<tr>
<td>8</td>
<td>0.9</td>
<td>1 low complexity 1 medium complexity</td>
<td>academic - low commercial - low</td>
<td>academic - low commercial - low</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>1 low complexity</td>
<td>academic - low commercial - low</td>
<td>academic - low commercial - low</td>
</tr>
</tbody>
</table>
5.2.4 Analysis and Interpretation

After collecting experimental data in the operation phase, we want to be able to draw conclusions based on this data. The analysis and interpretation of the data are presented as follows.

5.2.4.1 Difficulties to understand or apply the process

Analyzing subjects’ answers for the difficulties in the use of the RiPLE-RE process, it was identified that all subjects had some problem in the RE activities. However, two subjects (ID 1, 2) had difficulties related to the supporting tool. The others had problems to understand or apply the process, representing approximately 71.4% of the total. The two subjects that did not present difficulties to understand the RiPLE-RE have a good experience in software industrial projects (5 to 6 projects), thus it aid the understanding of the process.

Figure 5.1 shows the histogram with the distribution density of the found difficulties.

According to the Figure, one subject (ID 7) had difficulty to represent variabilities, specially at the beginning, when the team was a little bit confused with the features representation types, factor caused by the lack of experience in both SPL concepts and the rental domain. For another subject (ID 3), the process must improve the description about the use of the binding-time. Three subjects (ID 3, 6, 8) reported difficulty in the use of template in XML, which is more complex due to use of tags. Some subjects (ID 1, 2, 6) had difficulties in the usage of the process due to lack of a tool to support the full process. Three subject (ID 3, 5, 7) had difficulties to understand the verification report.

Figure 5.1 Difficulties Histogram.
goal, so they had problems is its use. Two subjects (ID 6, 7) reported that the relationship among assets (features, requirements and use cases) must be improved. At the end, two subjects (ID 5, 8) reported that they had rework in the specification.

5.2.4.2 Subjects’ Satisfaction

Subjects’ satisfaction was evaluated according to the defined scale (very satisfied, satisfied, impartial, unsatisfied and very unsatisfied). Regarding to results, 85% (ID 1, 2, 3, 5, 7, 8) of the participants were satisfied and 15% (ID 6) were impartial with the use of the process. Figure 5.2 presents the frequency of the subjects’ satisfaction according to the defined scale (very satisfied, satisfied, impartial, unsatisfied and very unsatisfied).

![Figure 5.2](image)

Figure 5.2 Frequency of the subjects’ satisfaction.

In general, the subjects reported that the process has a well-detailed set of guidelines and examples which enables that the requirement phase follows a correct flow, the process fulfilled their expectations and it was important for the other project steps.

5.2.4.3 Error Distribution

From the analysis of the verification reports created by the work products reviewers, we analyzed the error causes of the features, requirements and use cases. Thus, we found a total of 64 errors according to the following distribution: 10 errors due to lack of experience, 41 due to omissions in analysis, 3 by fault in input documentation, and 10 due to lacks in documentation standards. In these errors, the problems types found were 19 incompleteness, 18 inconsistencies, 18 lacks of standardization and 9 ambiguities. The density of this distribution can be seen in Figure 5.3.
These results show that approximately 80% of the defects were directly related to experience and omission of the team. Thus, it rejects the null hypothesis: $\mu_{ErrorsDueToExperienceAndOmissions} (\sim 80\%) < 50\%$.

The concentration of errors related to omission in analysis can be due to the subjects’ ability and interest in RE. All subjects participated in the RE activities, however, some of them cannot have profile for requirements analyst.

A less number of errors was caused by problems in the process. Therefore, the errors in RE could be reduced with the increase of the learning curve as time passes.

### 5.2.4.4 Defect Index

The average severity of the RE defects was analyzed according to the boundaries of $0 \leq DI \leq 10$. In general, we identified a Defect Index ($DI$) of 4.31, based on 64 defects found (18 critical, 25 medium, 21 trivial). Regarding to the analysis of the defect index in each RE work product, we identified a DI of 5.42 (10 defects: 8 critical, 6 medium, 5 trivial) in features, 3.9 (30 defects: 7 critical, 12 medium, 11 trivial) in requirements and 3.73 (15 defects: 3 critical, 7 medium, 5 trivial) in use cases. Table 5.2 shows the descriptive statistics with the data collected during the experimental study.

The defect index mean rejects the null hypothesis: $\mu_{DI}(\sim 4.35) > 6.5$. It indicates that the DI found represents a moderate severity. The standard deviation is small, thus, the defect index and error distribution results validate the process potential to aid to generate domain requirements specification in compliance with its standards.
### 5.2.4.5 Effort

To develop this project, it was spent approximately 590 hours, the sum of worked hours of every member in each project lifecycle phase, as can be seen in Table 5.3. In the requirements phase were spent approximately 217 hours, a mid effort of approximately 36% regarding the whole project. Thus, the effort found confirms the null hypothesis: $\mu_{E_{f f}}(36\%) \geq 30\%$.

<table>
<thead>
<tr>
<th>Table 5.2</th>
<th>Results for Defect Index.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measure</td>
<td>Defect Index</td>
</tr>
<tr>
<td>Mean</td>
<td>4.35</td>
</tr>
<tr>
<td>Maximum</td>
<td>5.42</td>
</tr>
<tr>
<td>Minimum</td>
<td>3.73</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>0.87</td>
</tr>
<tr>
<td>Defect Index</td>
<td>4.31</td>
</tr>
<tr>
<td>Null Hypothesis</td>
<td>$\mu_{df} &gt; 5$</td>
</tr>
</tbody>
</table>

### Table 5.3 | Effort in the use of the process. |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Effort</td>
<td>Hours</td>
</tr>
<tr>
<td>Scoping</td>
<td>111.52</td>
</tr>
<tr>
<td>Requirements</td>
<td>217.01</td>
</tr>
<tr>
<td>Design</td>
<td>74.30</td>
</tr>
<tr>
<td>Implementation</td>
<td>149.30</td>
</tr>
<tr>
<td>Evolution</td>
<td>7.20</td>
</tr>
<tr>
<td>Others</td>
<td>30.40</td>
</tr>
</tbody>
</table>

In RE, all SPL features and requirements were specified. However, due to the time constraints, some project steps were performed in a reduced SPL scope, such as use cases specification, design and implementation. Thus, the effort was not proportional in the whole project. Therefore, these results are not conclusive to the process efficiency.

### 5.3 Conclusion

In general, the process produces benefits that justify its use. The results of this experimental study indicate that the process allows performing the RE in the SPL context with a good effectiveness. Its efficiency can be increased with the use of an appropriate supporting tool. On the other hand, the aspects related to usability needs to be improved due to difficulties of the subjects to understand or apply the process. Despite of the good subjects’ satisfaction in using the process, the participants had some difficulties in the process understanding, mainly, in the concepts that were sometimes new to them. These results are very important to recalibrate the training in the future.
Even with a small number of subjects, it can be valid to make some correlations based on the profile of the subjects and the results obtained. It was noted that subjects who have more participation in industrial projects have reported less difficulties to understand or apply the process than subjects with less experience. The experience in requirements engineering also influenced these results: subjects who have more experience in RE have less difficulties than subjects with less experience. Another tendency is that subjects with more years since graduation (the subjects varied from 0.8 years until 5 years) also have less difficulties than subjects that have concluded their course more recently. Thus, this suggests that the proposed process is better understood and applied by experienced users.

However, the analysis of this experimental study cannot be considered conclusive. Its particular context (population and scope) limits the external validity of the results. Positive results do not guarantee success in all contexts. On the other hand, the negative results are strong evidence of fail in other contexts. Furthermore, we did not use hypothesis tests for some metrics, such as defect index, error distribution and effort due to lack of data distribution.

5.4 Lessons Learned

After concluding the experimental study, we identified some aspects that should be considered in order to repeat the experiment, since they were seen as limitations of the first execution.

Separated Roles. During the project, some subjects were associated to more than one role, for example, manager and requirements analyst. This could have a negative impact on the project, mainly related to overworking. Thus, this aspect should be re-analyzed and a possible solution can be allocating subjects in only a role.

Verification Report. Some defects reported as lacks in documentation standards or fault in input documentation were contradictory, because the defect cause was ambiguous in the verification report. The term “documentation” was confused by the reviewers, which can be the domain requirements specification or the process document. It shows that the verification report template must be improved.

Training. Several subjects had difficulties to understand new concepts related to SPL, in special, the variabilities and commonalities analysis. Thus, it indicates that the training must be improved, mainly in the concepts that are specific for SPL. It could include a complete and detailed example covering the whole process.

Templates. Several trivial defects were related to failures in the use of the XML
template, which is more complex due to use of tags. However, it makes the information extraction and variations choice easier, aiding in the products assets derivation by a supporting tool. An alternative to reduce this complexity is the use of a tool with a friendly interface to record the data in the XML file.

**Supporting tool.** The use of an appropriate supporting tool is essential to subjects’ productivity, reducing rework in the RE activities. Thus, it shows the need of an effective tool to support the whole process, facilitating the products assets derivations, the record of the data in XML file, and the integration and traceability of the work products and variabilities.

### 5.5 Chapter Summary

This chapter presented the definition, planning, operation, analysis and interpretation, and conclusion of an experimental study to evaluate the RiPLE-RE process. The study, summarized in Table 5.4, analyzed, from the perspective of researchers, the process usability, effectiveness and efficiency for the context of software product line project.

<table>
<thead>
<tr>
<th>Null Hypothesis</th>
<th>Result</th>
<th>Rejection</th>
</tr>
</thead>
<tbody>
<tr>
<td>H01</td>
<td>$\mu_{\text{errors due to lack of experience and omissions in analysis}} &lt; 50%$</td>
<td>$\sim 80% &lt; 50%$</td>
</tr>
<tr>
<td>H02</td>
<td>$\mu_{\text{obj}} &gt; 5$</td>
<td>$\sim 4.35 &gt; 6.5$</td>
</tr>
<tr>
<td>H03</td>
<td>$\mu_{\text{eff}} \geq 30%$</td>
<td>$\sim 36% \geq 30%$</td>
</tr>
</tbody>
</table>

The qualitative analysis showed that the process has a greater potential according to users’ satisfaction. On the other hand, some difficulties were reported by the subjects, indicating improvements in the process. Regarding to quantitative analysis, the experiment demonstrated that the process has a good effectiveness. However, its efficiency can be improved with the use of an appropriate supporting tool.

Even with the reduced number of subjects participating to the experiment, and the identified validity threats, the results suggest that the proposed RE process for SPL may be viable after some calibrations. Nevertheless, it is important to state that this experiment was performed in a particular context. To analyze possible benefits and trade-offs of the proposed process, more experiments should be conducted in the future.

The next chapter will present the conclusions of this work, its main contributions and directions for future work.
The implications to reduce development time and improve product quality make the software reuse approach very attractive for software organizations. Nowadays, some organizations are adopting software product lines approaches as a way to achieve these benefits.

Regarding to the requirements engineering, there are challenges to be addressed by the SPL approaches. Chapter 3 presented these problems, emphasizing the lack of strategies for linking the variabilities from different core assets, providing support for instantiation of the products. Moreover, the approaches analyzed in this study present some gaps in the RE activities definition.

From this scenario, we proposed a new process addressed to solve these problems, which is defined in a systematic manner. The proposed process was evaluated in an experimental study, which presented the benefits in using the process and suggestions for its improvement.

In this Chapter, the conclusion of this work is presented. Thus, the Chapter remainder is organized as follows. The research contributions are highlighted in Section 6.1. The related work to RiPLE-RE is described in Section 6.2 and the future work is listed in Section 6.3. Academic contributions are listed in Section 6.4 and the concluding remarks of this dissertation are described in Section 6.5.

6.1 Research Contributions

The main contributions of this work can be split into the following aspects: i. an extensive systematic review on requirements engineering for software product lines; ii. the definition and systematization of a requirements engineering process for software product lines; iii. the development of a tool to support the proposed traceability model;
the execution of an experimental study which evaluated the proposed process. These
contributions are further described next.

- **Systematic Review on Requirements Engineering approaches for Software
  Product Lines.** Through this review, sixteen approaches were identified and ana-
  lyzed according to the aspects related to the review research questions. The analysis
  results can be used to guide the definition of a new requirements engineering ap-
  proach for SPL. In addition, they also help practitioners to choose the best practices
  and approaches for this context;

- **RiPLE-RE.** After the systematic review, its results were used as inputs for the
  definition of RiPLE Requirements Engineering process (RiPLE-RE). It is defined
  in a systematic manner, with activities, tasks, inputs, outputs, roles and guidelines
  addressed to the context of software product lines;

- **SPLiTT.** In order to support the RiPLE-RE traceability model, a tool was im-
  plemented as product of this work: the Software Product Line Traceability Tool
  (SPLiTT);

- **Experimental Study.** An experimental study was applied in academic environment
  in order to evaluate the proposed process. This initial validation of RiPLE-RE
  helped in the improvement of the process, since the findings suggested some
  modifications in the activities description to improve understandability, and in the
  templates adopted by the process.

### 6.2 Related Work

Some Requirements Engineering approaches for SPL were identified through the system-
atic review, as described in Chapter 3. However, among the approaches included in this
review, there are many gaps in the requirements engineering activities. In general, there
is not a complete and systematic RE process for SPL. Thus, the output of the systematic
review served as a foundation for the RiPLE-RE definition, which is focused on some of
the questions raised in the systematic review. Thus, RiPLE-RE presents its activities in
a way more systematic than the other approaches and it proposes a new strategy to link
the decisions among variabilities with the its SPL traceability model. Furthermore, the
requirements and use cases are specified in a structured way, enabling their instantiation
for SPL and product. However, some limitations are identified in the process, as can be
seen in the experimental study (Chapter 5) and in future work, which is presented in next section.

6.3 Future Work

Due to the time constraints imposed on a M.Sc. degree, this work can be seen as an initial step towards the efficient, usable and effective requirements engineering process in the context of software product lines. Thus, there are interesting topics to improve what was started, and new paths to explore. Thus, the following issues should be investigated as future work:

- **Features Interaction.** The RiPLE-RE defines dependencies related to constraints (implication, exclusion, implemented-by) and refinements (composed-of, generalization). However, one dependency may affect the validity of other dependencies (Sinnema *et al.*, 2004), causing impact in a SPL, affecting, for example, the reusable assets. In this way, conflicts in most of the cases can be dealt through a management of feature dependencies. This management is a complex task, but it need be integrated to the RE process.

- **Product Derivation.** This work provides assets for supporting the product derivation. However, it does not defines activities to perform development with reuse. Thus, this aspect must be addressed to cover the entire SPL lifecycle.

- **Metrics.** This dissertation proposed some metrics to evaluate RiPLE-RE use in the experimental study, however these metrics were never used before, therefore they need to be refined and reproduced. This metric set could be also increased by several other metrics to measure requirements engineering for SPL.

- **Application of RiPLE-EM in an industrial context.** This dissertation presented the definition, planning, operation, analysis and interpretation of an experimental study. However, new experimental studies are necessary in conjunction with the industry, in different contexts, with a more significant number of subjects, in order to refine the process and to improve the experiment design. We believe that more experiments must be performed, taking into account the lessons learned with the first experiment, thus more concrete conclusions can be drawn.

- **Support tool.** The use of an appropriate supporting tool is essential to efficiency of a requirements engineering process. Thus, it shows the need of a tool to support
the whole process, integrating its work products and variabilities, and facilitating the products assets derivations with the use of XML documents. It will be extended from tool proposed in (Lisboa et al., 2007).

6.4 Academic Contributions

As an intermediate result of the work presented in this dissertation, the following contributions can be enumerated:

- (Neiva et al., 2009) An Experimental Study on Requirements Engineering for Software Product Lines, In 35th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), Service and Component Based Software Engineering (SCBSE) Track.

Furthermore, the co-participation on the following publication contributed for acquiring experience and knowledge in the software product line and software reuse area:


6.5 Concluding Remarks

This work presented a requirements engineering process for software product lines, which can be seen as a systematic way to guide requirements elicitation, specification, analysis, negotiation, verification and management. This process was based on an extensive systematic review, whose results can be used for a valid set of activities, techniques and models for requirements engineering in SPL.

Additionally, an experimental study was conducted to evaluate the effectiveness, efficiency and usability of the RiPLE-RE process. The qualitative analysis showed that the process has a good potential according to users’ satisfaction. On the other hand, some difficulties were reported by the subjects, indicating improvements in the process. Regarding to quantitative analysis, the experiment demonstrated that the process has a good effectiveness. However, its efficiency can be improved with the use of an appropriate supporting tool.

Therefore, this dissertation can be considered a relevant contribution to the area of software reuse and requirements engineering.


REFERENCES


REFERENCES


REFERENCES


As part of the experiment instrumentation, detailed in Chapter 5, two questionnaires were defined, and applied to the subjects. The next sections lists all questions inside each questionnaire.

The first questionnaire (detailed in Section A.1) was intended to collect data about the subjects background, and the second one (detailed in Section A.2) was intended to collect information about RiPLE-RE use.

### A.1 QT1 - Background Questionnaire

**Amount of Years since graduation:**

1. How many commercial software projects did you participate after graduation?

<table>
<thead>
<tr>
<th>Category</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low Complexity:</td>
<td></td>
</tr>
<tr>
<td>Medium Complexity:</td>
<td></td>
</tr>
<tr>
<td>High Complexity:</td>
<td></td>
</tr>
</tbody>
</table>

2. About your personal experience in Requirements Engineering (mark x):

<table>
<thead>
<tr>
<th>Area</th>
<th>None</th>
<th>Low</th>
<th>Medium</th>
<th>High</th>
<th>Years</th>
</tr>
</thead>
<tbody>
<tr>
<td>Academic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Commercial</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3. About your personal experience in software reuse (mark x):
4. Please, inform which techniques/methods you know in requirements engineering and software reuse areas.

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________
A.2 QT2 - RiPLE-EM Analysis Questionnaire

Regarding the requirements engineering using the RiPLE-RE process, please answer the following questions:

1. Did you have any difficulties in understanding or applying the requirements engineering process (RIPLE-RE)? Which one(s)?

2. In your opinion, what are the strengths of the requirements engineering process?

3. In your opinion, what are the weak points of the requirements engineering process?

4. Which improvements would you suggest for the requirements engineering process?

5. What is your satisfaction in using the RiPLE-RE process?
   ( ) Very satisfied ( ) Satisfied ( ) Impartial
   ( ) Unsatisfied ( ) Very Unsatisfied
   Please, justify: ________________________________
In this appendix, the templates of RiPLE-RE will be described and detailed.

### B.1 Domain Requirements Template

In order to document all domain requirements, a template in XML is proposed.

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<requirements>
  <requirement id="FRX" type="Functional | Non-Functional" implication="" exclusion="" binding-time="">
    <name />
    <variability-type>Optional | Mandatory | Variant</variability-type>
    <priority>High | Medium | Low</priority>
    <rationale />
  </requirement>
  <description>
    <text />
    <vp id="FRX.VPY" binding-time="" implication="" exclusion="" />
    <description />
    <cardinality min="" max="" />
    <variant id="FRX.VPY.V1" implication="" />
    <variant id="FRX.VPY.V2" />
  </vp>
  <text />
  </description>
</requirements>
```

**Figure B.1** Domain Requirements Template

### B.2 Domain Use Cases Template

In order to document all domain use cases, a template in XML is proposed.
<ucs>
  
  - <uc id="UCX" implication="" exclusion="" binding-time=""">
      <name>name</name>
      <variability-type>Mandatory|Variant</variability-type>
      <rationale />
      <actors />
      <dependency />
      <precondition />
      
      - <main-flow>
        <f />
        <f>[UCX.VPX]</f>
        <extension-point id="EP1" />
        <f />
      </main-flow>
      
      - <alternative-flow id=""/>
        <f />
      </alternative-flow>
      
      - <alternative-flow id=""/>
        <f />
      </alternative-flow>
      
      - <exception-flow id=""/>
        <f />
      </exception-flow>
      <postcondition />
    </uc>

  </ucs>

Figure B.2 Domain Use Cases Template
B.3 Verification Report Template

In order to document all errors found in the feature model, domain requirements and domain use cases, a verification report template is proposed.

Reviewer:
Date:
Reviewed items:

- **Features**: <set of features (id). It can be described by feature range, e.g. F1 to F20>
- **Requirements**: <set of requirements (id). It can be described by requirement range>
- **Use cases**: <set of use cases (id). It can be described by use cases range>

Found Problems:

- **Item Id**:
- **Problem**:
  - **Problem Type**: <incompleteness, inconsistency, ambiguity, traceability and standardization>
  - **Problem Severity**: <serious, medium, trivial>
  - **Problem Cause**: <lack of experience, lacks in input documentation, lacks in documentation standards, omissions in analysis, fault in input documentation>
Systematic Review Quality Criteria

This appendix lists the quality criteria used to select primary studies in the systematic review procedure executed and detailed in Chapter 3. The evaluation results of the criteria also are presented in this appendix.

Table C.1 presents quality criteria for requirements engineering approaches for software product line. The scope presented in Table represents where should be applied the quality criteria.

<table>
<thead>
<tr>
<th>Quality Criteria</th>
<th>Options</th>
<th>Score</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>QC1 Are the roles described?</td>
<td>Yes. They are explicitly defined in the study.</td>
<td>1</td>
<td>Q1</td>
</tr>
<tr>
<td></td>
<td>Partly. They are implicit.</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>No. They are not defined and cannot be readily inferred.</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>QC2 Are the guidelines described?</td>
<td>Yes. They are explicitly defined for all activities found in the study.</td>
<td>1</td>
<td>Q1</td>
</tr>
<tr>
<td></td>
<td>Partly. They are defined for some activities found in the study.</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>No. They cannot be found in study activities.</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>QC3 Are the inputs and outputs described?</td>
<td>Yes. Inputs and outputs are explicitly defined.</td>
<td>1</td>
<td>Q1</td>
</tr>
<tr>
<td></td>
<td>Partly. They are implicit.</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>No. They are not defined and cannot be readily inferred.</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>QC4 How configurable is the variability model?</td>
<td>High. Its variability model link trace among variations in different core assets, enabling easy core asset configuration.</td>
<td>1</td>
<td>Q3</td>
</tr>
<tr>
<td></td>
<td>Medium. Its variability model link trace partly among variations and core assets.</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Low. Its variability model does not link trace among variations in different core assets. Or it does not present variability model.</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>QC5 Where was the approach experimented?</td>
<td>It is not experimented.</td>
<td>0</td>
<td>Approach</td>
</tr>
<tr>
<td></td>
<td>No information about it.</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Academia. Experiments in academia using student participants are unlikely to be representative.</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Industry. Experiments in industry are more representative because it uses real case.</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Academia and Industry.</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>QC6 How well has detail of study?</td>
<td>High. Study is well detailed and it is easily understood.</td>
<td>1</td>
<td>Approach</td>
</tr>
<tr>
<td></td>
<td>Medium. Study is little detailed, but it can be understood.</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Low. Study presents an insufficient detail level to its understanding.</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>QC7 Is approach supported by tool?</td>
<td>Yes. Entire approach is supported by tool.</td>
<td>1</td>
<td>Approach</td>
</tr>
<tr>
<td></td>
<td>Partly. Part of the approach is supported by tool.</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>No. Approach is not supported by tool.</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Table C.1 Quality Criteria

The results of the application of these quality criteria are presented in Table C.2.

This result shows an overview about the quality score of the studies. However, it is difficulty establishing a reliable relationship between final quality score and the real quality of each study. For example, it does not evaluate the completeness of the study.
## APPENDIX C. SYSTEMATIC REVIEW QUALITY CRITERIA

<table>
<thead>
<tr>
<th>Approaches</th>
<th>Roles</th>
<th>Guidelines</th>
<th>Inputs/Outputs</th>
<th>Configurability</th>
<th>Experiment</th>
<th>Detail level</th>
<th>Support tool</th>
<th>Quality Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>FODA</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Medium</td>
<td>No information.</td>
<td>Medium</td>
<td>No</td>
<td>3.5</td>
</tr>
<tr>
<td>FAST</td>
<td>yes</td>
<td>yes</td>
<td>Partially</td>
<td>Medium</td>
<td>Industry</td>
<td>High</td>
<td>Partially</td>
<td>5.5</td>
</tr>
<tr>
<td>FeatuRSEB</td>
<td>no</td>
<td>Partially</td>
<td>Partially</td>
<td>Medium</td>
<td>Industry</td>
<td>Medium</td>
<td>Partially</td>
<td>3.0</td>
</tr>
<tr>
<td>FORM</td>
<td>yes</td>
<td>yes</td>
<td>Yes</td>
<td>Medium</td>
<td>Academia and Industry</td>
<td>High</td>
<td>No</td>
<td>5.5</td>
</tr>
<tr>
<td>VODRD</td>
<td>no</td>
<td>Partially</td>
<td>Partially</td>
<td>Medium</td>
<td>Industry</td>
<td>Medium</td>
<td>Partially</td>
<td>3.5</td>
</tr>
<tr>
<td>Odyssey</td>
<td>Partially</td>
<td>Partially</td>
<td>Partially</td>
<td>Medium</td>
<td>Academia</td>
<td>Medium</td>
<td>Partially</td>
<td>3.5</td>
</tr>
<tr>
<td>PuLSE-CDA</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>High</td>
<td>industry</td>
<td>Medium</td>
<td>Yes</td>
<td>4.5</td>
</tr>
<tr>
<td>DH</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Low</td>
<td>No information.</td>
<td>Medium</td>
<td>Yes</td>
<td>2.0</td>
</tr>
<tr>
<td>Goal-oriented</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Medium</td>
<td>Academia</td>
<td>Medium</td>
<td>Yes</td>
<td>2.5</td>
</tr>
<tr>
<td>PLA</td>
<td>yes</td>
<td>yes</td>
<td>Partially</td>
<td>Medium</td>
<td>Industry</td>
<td>High</td>
<td>no</td>
<td>4.41</td>
</tr>
<tr>
<td>PRS</td>
<td>No</td>
<td>Partially</td>
<td>Partially</td>
<td>Medium</td>
<td>Industry</td>
<td>Medium</td>
<td>No</td>
<td>3.2</td>
</tr>
<tr>
<td>DRAMA</td>
<td>No</td>
<td>Partially</td>
<td>Partially</td>
<td>Medium</td>
<td>Academia and Industry</td>
<td>Medium</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>PLUS</td>
<td>No</td>
<td>Yes</td>
<td>Partially</td>
<td>Medium</td>
<td>Industry</td>
<td>High</td>
<td>No</td>
<td>4</td>
</tr>
<tr>
<td>PLUSS</td>
<td>no</td>
<td>Partially</td>
<td>Partially</td>
<td>Medium</td>
<td>Industry</td>
<td>Medium</td>
<td>Partially</td>
<td>3.5</td>
</tr>
<tr>
<td>PR</td>
<td>no</td>
<td>yes</td>
<td>Partially</td>
<td>Medium</td>
<td>Industry</td>
<td>Medium</td>
<td>Yes</td>
<td>4.5</td>
</tr>
<tr>
<td>SPLE</td>
<td>no</td>
<td>Partially</td>
<td>yes</td>
<td>High</td>
<td>Industry</td>
<td>High</td>
<td>No</td>
<td>4.5</td>
</tr>
</tbody>
</table>

Table C.2 Quality Criteria Results