Pós-Graduação em Ciência da Computação

"TIRT: A Traceability Information Retrieval Tool for Software Product Lines Projects"

By

Wylliams Barbosa Santos

M.Sc. Dissertation

Federal University of Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, SEPTEMBER/2011

Federal University of Pernambuco
Center for Informatics
Graduate in Computer Science

Wylliams Barbosa Santos

"TIRT: A Traceability Information Retrieval Tool for Software Product Lines Projects"

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

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

RECIFE, SEPTEMBER/2011

Catalogação na fonte
Bibliotecária Jane Souto Maior, CRB4-571

Santos, Wylliams Barbosa

TIRT: A traceability information retrieval tool for software product lines projects / Wylliams Barbosa Santos - Recife: O Autor, 2011.

xvi, 115 folhas : il., fig., tab.

Orientador: Silvio Romero de Lemos Meira.

Dissertação (mestrado) - Universidade Federal de Pernambuco. CIn, Ciência da Computação, 2011.

Inclui bibliografia e apêndice.

1. Ciência da Computação. 2. Engenharia de software. I. Meira, Silvio Romero de Lemos (orientador). II. Título.

004 CDD (22. ed.) MEI2011 – 150

Dissertação de Mestrado apresentada por **Wylliamas Barbosa Santos** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **"TIRT: A Traceability Information Retrieval Tool for Software Product Lines Projects"**, orientada pelo **Prof. Silvio Romero de Lemos Meira** e aprovada pela Banca Examinadora formada pelos professores:

Profa. Carina Frota Alves
Centro de Informática / UFPE

Prof. Patrick Henrique da Silva Brito
Instituto de Computação / UFAL

Prof. Silvio Romero de Lemos Meira
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 13 de setembro de 2011.

Prof. Nelson Souto Rosa
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

I specially dedicate this dissertation with love and affection to my father Rosenildo (In Memoriam). The best man and friend I ever met.

Acknowledgements

Firstly to God for every blessing, protection, love and strength throughout my life.

To my father Rosenildo *(in memoriam)*, that has always guided me, supported and encouraged me every step of my life. I would like to thank him for his reference of love, happiness, father and teacher, as well as their advices, words of encouragement and support during the master degree. Finally, I would like to dedicate this work to my great, eternal and beloved father Rosenildo, because this is the realization of our dream!

To my mother for all support, love and friendship. I am profoundly grateful for her wisdom in understanding the distance that separates us, especially in the most delicate and difficult moments we faced. Mother, know that even with the distance will always be together and great friends. I also dedicate this work with all the love and affection to my mother and friend Graça.

My friend and bride Deborah, for all love, affection, support and friendship during this the master degree, as well as by welcome and friendly words in the most adverse moments.

My brother Wylker, for friendship during our lives and especially during these years we lived in Recife.

To my uncles Marcos, Kêda and their entire family, my eternal gratitude for all comfort and affection during the most delicate moment of my life. All these feelings were channeled in a positive way, giving me strength and support for achieving this work.

To my friends from Maceió for all support: Elinaldo da Mota, Fabiano Wandeley, Telma Melo, Marcelo Feijó, Giordany Corado, Thiago Alves and Bruno Correia.

All RiSE members which contributed with valuable suggestions, comments and reviews in this work, especially, to friends: Leandro Souza, Jonatas Bastos, Iuri Santos, Thiago Fernandes, Ivonei Freitas, Raphael Pereira, Ivan Machado and Paulo Silveira.

Last but not least, my sincere thanks to my co-advisor Eduardo Almeida, for all guidance, support and teachings. I am also very grateful to my advisor Silvio Meira who accepted me as his student.

Agradecimentos

Primeiramente a Deus, por toda benção, proteção, amor e força durante toda minha vida.

Ao meu pai Rosenildo *(in memoriam)*, que sempre me guiou, apoiou e incentivou-me em todas as etapas de minha vida. Gostaria de agradecê-lo pelo seu referencial de amor, alegria, pai e mestre, assim como seus conselhos, palavras de incentivo e apoio durante a jornada do mestrado. Por fim, gostaria de dedicar este trabalho ao meu grande, eterno e amado pai Rosenildo, pois este é a concretização do nosso sonho!

À minha mãe por todo suporte, amor e amizade. Sou imensamente grato por sua sabedoria em compreender a distância que nos separa, principalmente nos momentos mais delicados e difíceis que enfrentamos. Mãe, saiba que independente da distância sempre estaremos juntos e seremos grandes amigos. Também dedico este trabalho com todo amor e carinho à minha mãe e amiga Graça.

À minha amiga, companheira e noiva Deborah, por todo amor, carinho, suporte e amizade durante essa jornada, assim como pelas palavras amigas e acolhimento nos momentos mais adversos.

Ao meu irmão Wylker, pelo companheirismo e amizade durante nossas vidas e principalmente ao longo desses anos que moramos em Recife.

Aos meus tios Marcos, Kêda e toda sua família, minha gratidão eterna, por todo conforto e carinho durante o momento mais delicado de minha vida. Todos esses sentimentos foram canalizados de forma muito positiva, dando-me forças e suporte para concretização deste trabalho.

Aos meus amigos de Maceió por todo apoio: Fabiano Wandeley, Elinaldo da Mota, Telma Melo, Marcelo Feijó, Bruno Correia, Giordany Corado e Thiago Alves.

A todos os membros do RiSE que contribuíram com valiosas sugestões, comentários e revisões neste trabalho. Em especial, aos amigos: Leandro Souza, Jonatas Bastos, Iuri Santos, Thiago Fernandes, Ivonei Freitas, Raphael Pereira, Ivan Machado e Paulo Neto.

Por fim, mas não menos importante, meus sinceros agradecimentos ao meu co-orientador Eduardo Almeida, por todo direcionamento, apoio e ensinamentos. Também sou muito grato ao meu orientador Silvio Meira que me acolheu como seu aluno de mestrado.

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

Linha de Produto de Software - SPL tem provado ser a metodologia para o desenvolvimento de uma diversidade de produtos de software e sistemas com custos mais baixos, em menor tempo, e com maior qualidade. Numerosos relatos documentam significativas experiências adquiridas através da implantação de linhas de produtos na indústria de software.

Neste cenário, a rastreabilidade se refere à capacidade de conectar e preservar o rastro de transformação de diferentes artefatos de softwares, portanto, é considerada uma condição necessária para preservar a consistência dos artefatos durante a implementação, reduzindo o tempo e o custo de desenvolvimento da SPL. No entanto, a adoção e manutenção da rastreabilidade no contexto das linhas de produtos são consideradas tarefas difíceis, devido ao grande número e heterogeneidade dos artefatos desenvolvidos. Além disso, a criação manual e manutenção das relações de rastreabilidade são difíceis, propensa a erros, lenta e complexa.

Neste sentido, esta dissertação propõe diferentes cenários de recomendação de rastreabilidade, onde os artefatos podem estar relacionados. Além disso, os requisitos, projeto e desenvolvimento de uma ferramenta também são propostos. Esta ferramenta foca nas atividades de manutenibilidade, relacionadas à rastreabilidade entre diferentes artefatos de uma SPL através do seu sistema de recomendação. Assim, o tempo gasto nessas atividades pode ser reduzido e menos propenso a erros. Finalmente, este trabalho também apresenta um estudo experimental inicial, a fim de identificar a viabilidade da ferramenta e cenários de rastreabilidade propostos.

Palavras-chave: Rastreabilidade, Linha de Produto de Software, Ferramenta, Recuperação de Informação, Rastreabilidade em Linha de Produto de Software

Abstract

Software Product Line - 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. Numerous reports document the significant achievements and experience gained by introducing software product lines in the software industry.

In this scenario, traceability refers to the ability to link and preserve the trace of transformation among different software artefacts, thus it is considered a necessary precondition for preserving the consistency of the complete family model during development, reducing the development time and cost in the SPL. However, the adoption and maintenance of traceability in the context of product lines is considered a difficult task, due to the large number and heterogeneity of assets developed during product line engineering. Futhermore, the manual creation and management of traceability relations is difficult, error-prone, time consuming and complex.

In this sense, this dissertation proposes different scenarios of traceability recommendation, which the core assets can be related. In addition, the requirements, design and implementation of a tool is also proposed. It focus on the maintenance activities regarding to the traceability relationship among the different core assets in a SPL through its recommendation system. Thus, the time spent in these activities can be reduced and less error prone. Finally, this work also presents an initial experimental study in order to identify the viability of the proposed tool and traceability scenarios.

Keywords: Traceability, Software Product Lines, Tool, Information Retrieval, Traceability in Software Product Lines

Table of Contents

List of Figures

List of Tables

List of Acronyms

AJAX	Asynchronous Javascript and XML
AOSD	Aspect-Oriented Software Development
C.E.S.A.R.	Recife Center For Advanced Studies and Systems
CBD	Component-Based Development
DE	Domain Engineering
FR	Functional Requirements
GQM	Goal Question Metric
IR	Information Retrieval
J2EE	Java 2 Enterprise Edition
JSF	Java Server Faces
LSI	Latent Semantic Indexing
MVC	Model-View-Controller
NFR	Non-Functional Requirements
ORM	Object-Relational Mapping
PLE	Product Line Engineering
RiPLE	Rise Product Line Engineering Process
RiSE	Reuse in Software Engineering Labs
RM	Risk Management
SE	Software Engineering
SPL	Software Product Lines
SPLMT	Software Product Lines Management Tool
SQL	Structured Query Language

ToolDAy Tool for Domain Analysis

TIRT Traceability Information Retrieval Tool

UFPE Federal University of Pernambuco

VSM Vector Space Model

1 Introduction

Life Success comes from Leaving Your Comfort Zones.
—JONATHAN FARRINGTO

Since the time that software development started to be discussed within the industry, researchers and practitioners have been searching for methods, techniques and tools that would allow for improvements in costs, time-to-market and quality. Thus, an envisioned scenario was that managers, analysts, architects, developers and testers would avoid performing the same activities over and over. In this way, performing some kinds of reuse, costs would be decreased, because the time that would have been necessary to repeat an activity could be invested in others relevant tasks (Almeida *et al.*, 2007).

However, these benefits are not assured by the application of ad-hoc reuse, which address a not systematic, and generally restricted to source code. Systematic software reuse is a technique that is employed to address the need for the improvement of software development quality and efficiency, without relying on individual iniciative (Almeida *et al.*, 2007).

In this context, Software Product Lines Engineering has proven to be the methodology for developing a diversity of software products and software-intensive systems at lower costs, in short time and with higher quality. Numerous reports document the significant achievements and experience gained by introducing Software Product Lines in the software industry (Pohl *et al.*, 2005). However in the context of Software Product Lines (SPL), one asset can be used for many applications in the product development, for this reason, the traceability becomes more important in a Product Line environment.

Thus, the focus of this dissertation is to investigate the state-of-art of traceability in the context of Software Product Lines and facilitate the maintenance activities of a SPL based of a set recommendations senarios.

This Chapter contextualizes the focus of this dissertation and starts by presenting its motivation in Section 1.1 and a clear definition of the problem in Section 1.2. An overview of the proposed solution is outlined 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 of this work and, finally, Section 1.6 outlines the remainder structure of this dissertation.

1.1 Motivation

Software Product Lines (SPL) allow companies to achieve significant improvements in time-to-market, cost, productivity, and system quality (Clements and Northrop, 2001). For this reason, Product Line systems have been recognized as an important paradigm for software systems engineering. In the last years, a large number of methodologies and approaches have been proposed to support the development of software systems based on Product Line development, such as: FODA (Kang *et al.*, 1990a), RSEB (Griss *et al.*, 1998), FORM (Kang *et al.*, 1990b), FAST (Gupta *et al.*, 1997), PLUS(Eriksson *et al.*, 2005), and RiPLE (Neiva, 2008; Oliveira, 2009; Balbino, 2010; Machado, 2010).

On the other hand, traceability has been recognized as an highly important activity in software development (Ramesh and Jarke, 2001). In general, traceability relations can improve the quality of the product being developed, and reduce the development time and cost. In particular, traceability relations can support evolution of software systems, reuse of parts of the system by comparing components of the new and existing systems, validation that a system meets its requirements, understanding of the rationale for certain design and implementation decisions in the system, and analysis of the implications of changes in the system (Jirapanthong and Zisman, 2005).

However, the support for traceability in contemporary software engineering environmens is not always satisfactory due the lack of tool support, and a general perception that the effort to maintain traceability was excessive in respect to its benefits (Cleland-Huang, 2006). Other difficulties linked to traceability in SPL are: (a) there is a large number and heterogeneity of documents *(features, requirements, use cases, among others)*, even more than in traditional software development; (b) one needs to have a basic understanding of the variability consequences during the different development phases; (c) one needs to establish relationships between product members and the product line architecture, or relationships between the product members themselves; and (d) there is still poor general support for managing requirements and handling complex relations (Anquetil *et al.*, 2008). In addition, the lack of automation support for traceability, maintaining links between artefacts is a tedious and time consuming job (Abid, 2004). According to Spanoudakis and Zisman (2004), despite its importance and results from several years of research, empirical studies of traceability needs, and practices in industrial organizations have indicated that traceability support is not always satisfactory. As a result, traceability is rarely established in existing industrial settings.

Thus, in this work we try to mitigate the dificulties in the maintenance activities in SPL environments, based on a set of recommendations scenarios proposed in Chapter 4. Both Software Product Lines and Traceability are futher discussed in Chapters 2 and 3 repectively.

1.2 Problem Statement

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

This work investigates the state-of-art of Traceability in the context of Software Product Lines, provides a tool to support and reduce the effort spent in the traceability maintenance. This tool emcompasses some features, such as: a set of recommendations scenarios of traceability and the standarlized vocabulary for recording the artefacts. Moreover, the Traceability Information Retrieval Tool (TIRT) also emcompasses some features to aid in impact analysis.

1.3 Overview of the Proposed Solution

In order to accomplish the goal of this dissertation, the TIRT is proposed. The proposed tool supports the recommendation of core assets traceability in order to assist engineers in the maintenance activity. This Section presents the context where it is regarded and outlines the proposed solution.

1.3.1 Context

This dissertation is part of the Reuse in Software Engineering Labs (RiSE) [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. However, it is

[1] http://www.rise.com.br

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.

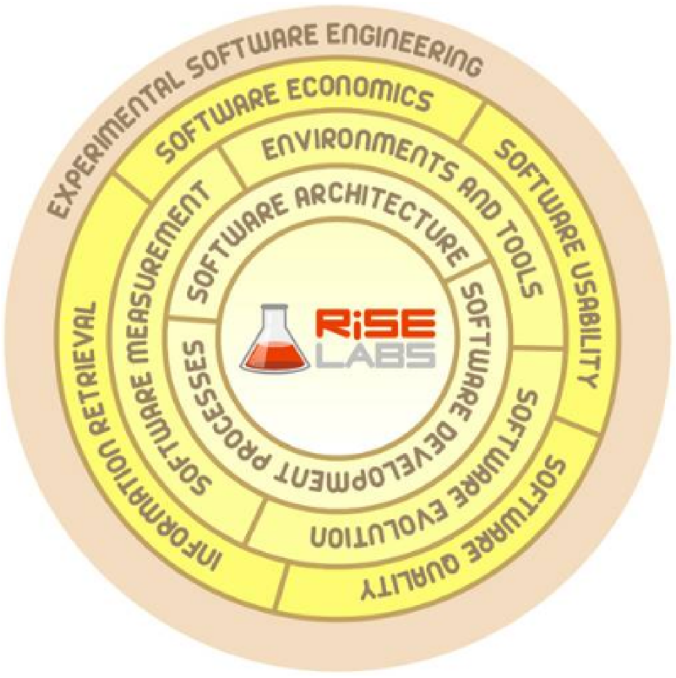


Figure 1.1 RiSE Labs Influences

Based on these areas, the RiSE Labs is divided in several projects, as shown in Figure 1.2. This framework encompasses many different projects related to software reuse and software engineering.

- **RiSE Framework:** Involves reuse processes (Almeida *et al.*, 2004; Nascimento, 2008), component certification (Alvaro, 2009) and reuse adoption process (Garcia, 2010).

- **RiSE Tools:** Research focused on software reuse tools, such as the Admire Environment (Mascena, 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), and the Legacy InFormation retrieval Tool (LIFT) (Brito, 2007), and the Tool for Domain Analysis (ToolDAy) (Lisboa, 2008).

- **RiPLE:** Development of a methodology for Software Product Lines (Filho *et al.*, 2008), composed of scoping (Balbino, 2010), requirements engineering (Neiva, 2008), design (Cavalcanti, 2010), implementation, test (Neto, 2010; Machado, 2010), and evolution management (Oliveira, 2009).

- **SOPLE:** Development of a methodology for Software Product Lines based on services (Ribeiro, 2010; Medeiros, 2010), with some ideas of RiPLE;
- **MATRIX:** Investigates the area of measurement in reuse and its impact on quality and productivity;
- **BTT:** Research focused on tools for detection of duplicated change requests (Cavalcanti, 2009; Cunha, 2009), based on text mining and software vizualization technique.
- **Exploratory Research:** Investigates new research directions in software engineering and its impact on reuse;
- **CX-Ray:** Focused on understanding the Recife Center For Advanced Studies and Systems (C.E.S.A.R.), and its processes and practices in software development. C.E.S.A.R. [2] is a CMMi level 3 company with around 700 employees.

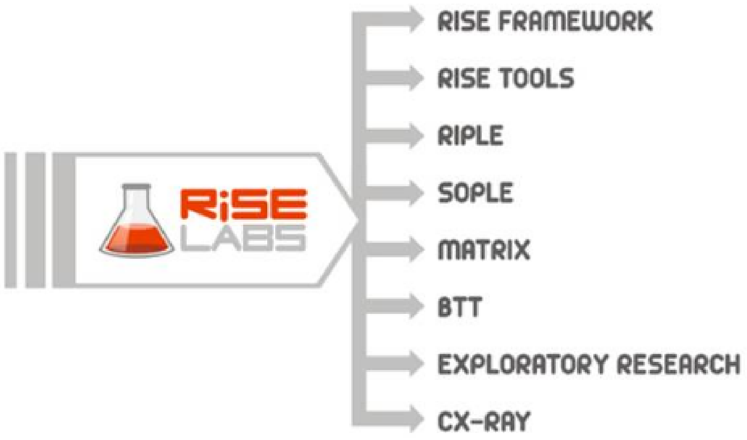


Figure 1.2 RiSE Labs Projects

This dissertation is part of the **RiSE Tools** and **RiPLE** process. Its goal is to provide a tool for search and maintainability of traceability between different core asset with the objective of avoiding the lose of tracebility information. In additional, this dissertation defines an approach to traceability recommendation of core assets identified in the **RiPLE**

[2]http://www.cesar.org.br

process, such as: *features*, *requirements* and *use cases*. This work was conducted in the context of a Software Product Lines environment, where the large number and heterogeneity of core assets generated during the development of product line systems may cause difficulties to identify common and variable aspects among applications, and reuse of core assets available under the product line architecture (Jirapanthong and Zisman, 2005). In this context, is possible to see that the recommendation of traceability links between these artefacts facilitate the matainance activity, and thus a systematic and consistent reuse (Pohl *et al.*, 2005).

1.3.2 Outline of the Proposal

Thus, the goal of this dissertation is to develop a solution that consists in a Web based application that enables engineers involved in the core assets maintenance to perform such task more effectively. This is possible with the *set of traceability recommendations scenarios*, the *standardized vocabulary for recording the core assets*, and the *feature to aid in impact analysis*, encompassed in the TIRT tool.

Traceability has been recognized as an important activity in SPL, thus the essence of this work is to make the maintenance activity of traceability in the context of Software Product Line less expensive and error-prone.

1.4 Out of Scope

As the traceability process in the context of Software Product Lines is part of a broader context, a set of related aspects are left out of its scope. Thus, the following issues are not directly addressed by this work:

- **Product Development:** The creation of individual products by reusing the artefacts is an important issue in a SPL. However, this aspect can be as complex as core assets development, thus, it is out of the scope of this work. In this context, the core assets focused in this work are: *features*, *requirements* and *use cases*.

- **Analyzes of Efficiency:** The TIRT tool uses a Vector Space Model (VSM) (Salton *et al.*, 1975) to represent core assets and perform searches that better meets our necessity. However, the activity of analyzing how efficient is the model is out of scope of this work. The work proposed by Salton *et al.* (1975), discusses for example, the efficiency of this model.

- **Evolution Management:** The SPL evolution control is ensured by appropriate practices of changes management. Inside the evolution management, it is possible to identify some important research topics, such as: change management, build management and release management. In this context, a recent work developed in the RiSE Labs (Oliveira, 2009) presented the RiPLE-EM process that focus on a systematic way to guide and manage the evolution of every management and release management activities. For this reason, evolution management is out of scope of this work. However, this work proposes an approach to mitigate the change impacts in the context of TIRT tool.

1.5 Statement of the Contributions

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

- **An analysis of the state-of-art** of the traceability area in the context of Software Product Lines Engineering. This study was conducted using some good practices of the Mapping Study proposed by Petersen *et al.* (2007), allowing an overview of the work in the literature and gray literature.
- **The TIRT tool** to support the maintenance of traceability links. It specifies, designs and implements a solution based on Text Mining (Feldman and Sanger, 2007) and Keywords search techniques, with the objective of recommending possible traceability links between different core assets.
- **Scenarios for traceability recommendation**, based on the metamodel developed in the RiSE Labs (Cavalcanti *et al.*, 2011).
- **Standardized vocabulary for recording the artefacts**, contributing to the efficiency of traceability recommendations.
- Description of some scenarios to support the process of **Impact Analysis** in the context of traceability maintenance, implemented in the TIRT tool.
- The definition, planning, analysis of an **experimental study** in order to evaluate the proposed tool and approaches.

1.6 Dissertation Structure

The remainder of this dissertation is organized as follows:

- **Chapter 2:** discusses the concepts of Software Reuse and outlines the main topics of SPL, such as: the SPL motivations, the SPL strategies and the essential activities.
- **Chapter 3:** introduces the concepts of Software Traceability and the main topics in the context of Software Product Lines. The concept of Impact Analysis also are discussed in this Chapter.
- **Chapter 4:** presents the functional and non-functional requirements proposed for the TIRT tool as well as architecture, the set of frameworks, technologies used during the TIRT implementation. Additionally, the proposals for *traceability recommendation* and the *standardized vocabulary for recording the artefacts* are also explored.
- **Chapter 5:** presents the definition, planning, operation, analysis and interpretation of the experimental study which evaluates the TIRT's tool and proposed approaches.
- **Chapter 6:** concludes this dissertation, summarizing the findings and proposing future enhancements to the solution, and discussing possible future work.
- **Appendix A:** describes the questionnaires and time sheets applied in the experimental study.

2 Software Product Lines: An Overview

Life has no limitations, except the ones you make.
—LES BROWN (Musician)

Software product lines engineering 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. Several reports document the significant achievements and experience gained by introducing software product lines in the software industry (Pohl *et al.*, 2005).

This Chapter introduces the concepts of Software Reuse in Section 2.1 and Software Product Lines in Section 2.2. The SPL Motivations and Benefits are discussed in Section 2.2.1. The SPL Essential Activities in Section 2.2.2 and the SPL Strategies in Section 2.2.3. The Chapter Summary is described in Section 2.3.

2.1 Software Reuse

Software reuse is the process of creating software systems from existing software rather than building software systems from scratch. This simple yet powerful vision was introduced in 1968 by McIlroy (1968). The concept of software reuse has been introduced to overcome the software crisis, i.e., the problem of building large and reliable software system in a cost effective and controlled way.

Many definitions for software reuse can be found in the literature. Krueger (1992) defines software reuse as ”the process whereby an organization define a set of systematic operating procedures to specify, produce, classify, retrieve, and adapt software artefacts for the purpose of using them in its development activities”. Basili and Rombach (1991) defines software reuse as the use of everything associated with a software project, including knowledge. According to Frakes and Isoda (1994), software reuse is defined as the use of engineering knowledge or artefacts from existing systems to build new ones. In a practic way, Jamwal (2010) defines software reuse as a ”solution that avoids a repeated labor in the software development and can make use of the knowledge and experience getting from the past software development and concentrates the especial part of application.”

A good software reuse process facilitates the increase of productivity, quality, and reliability, and the decrease of costs and implementation time. An initial investment is required to start a software reuse process, however, this investment pays for itself in a few reuses (Tracz, 1988).

Besides the issues related to non-technical aspects, a software reuse process must also describe two essential activities: the development for reuse and the development with reuse. In the literature, several research work that study efficient ways to develop reusable software can be found. These work focus on two directions: *domain engineering* and, currently, *product lines*, as can be seen in the next Sections. According to Pohl *et al.* (2005), the aim of the *domain engineering* process is to define and realise the commonality and the variability of the software product line.

Thus, with the use of techniques such as domain engineering or software product lines, a set of assets (requirements, architectures, source code, test cases, etc.) can be reused in an effective way. In this context, the reminder of this Chapter describes the basic concepts and ideas of Software Product Lines.

2.2 Software Product Lines

The way that goods are produced has changed significantly in the course of time. Formerly goods were handcrafted for individual customers. Along the time, the number of people who could afford to buy various kinds of products increased. In the domain of automobiles Ford has led the invention of the production line, which enabled the production for a mass market much more cheaply than individual product creation on a handcrafted basis. However, the production line reduced the possibilities for diversification. Thus, industry was confronted with a rising demand for individualised products. This was the beginning of mass customisation, which meant taking into account the customers’ requirements and giving them what they wanted (Pohl *et al.*, 2005).

Mass customisation is the large-scale production of goods tailored to individual customers’ needs (Davis, 1987). For the customer, mass customisation means the ability to have an individualised product. For the company, mass customisation means higher technological investments which leads to higher prices for the individualised products and/or to lower profit margins for the company. Both effects are undesirable. Thus many companies, especially in the car industry, started to introduce common platforms for their different types of cars by planning beforehand which parts will be used in different car types. The parts comprising the platform were usually the most expensive subsystem in terms of design and manufacturing preparation costs. The use of the platform for different car types typically led to a reduction in the production cost for a particular car type. The platform approach enabled car manufactures to offer a large variety of products and to reduce costs at the same time (Pohl *et al.*, 2005).

R. Cooper and Kleinschmidt (2001) consider a platform when we can derive multiple products. According to TechTarget (2004), a platform is any base of technologies on which other technologies or processes are built, however, a common definition of platform does no exist. This definition encompasses all kinds of reusable artefacts as well as all kinds of technological capabilities.

The combination of mass customisation and a common platform allows us to reuse a common base of technology and, at the same time, to bring out products in close accordance with customers' wishes. The systematic combination of mass customisation and the use of a common platform for the development of software-intensive systems and software products is the key focus of software product line engineering.

The strategy commonly adopted is, first, to focus on what is common to all products, and next, to focus on what is different, designing the commonality first and differences later. In the first step, artefacts are provided that can be reused for all products. These artefacts may be built from scratch or derived from another platform or earlier systems (Pohl *et al.*, 2005).

This flexibility is a precondition for mass customisation; it also means that we can predefine what possible realisations shall be developed. The flexibility described here is called variability in the software product lines context. This variability is the basis for mass customization. For Frank J. van der Linder and Schmid (2007), a key distinction of software product line engineering from other reuse approaches is that the various assets themselves contain explicit variability.

Variability management is a concern in any software product lines engineering approach, covering the whole life-cycle, since the early steps of scoping, covering the implementation, testing and finally the evolution. Thus, variability is relevant to all assets throughout software development (Frank J. van der Linder and Schmid, 2007).

Consequently, it became necessary to manage carefully the trace information from a platform to the products derived from it. Without such trace information, it is barely possible to find out which parts of the platform have been used in which product.

Next, are described the different reasons and benefits that motivate the companies to adopt a software product lines engineering approach.

2.2.1 SPL Motivations and Benefits

Many different reasons lead companies to adopt a software product lines approach. These range for more process-oriented aspects such as cost and time over product qualities (e.g. reliability to end-user aspects such as user interface consistency) (Frank J. van der Linder and Schmid, 2007).

According to Clements and Northrop (2001), the companies adopt software product lines in order to achieve not only benefits that they desired, but also benefits that they absolutely needed to ensure their organizational health and in some cases their very existence.

The following are some reasons for developing software based on the product line engineering approach.

Reduction of Development Costs

An essential reason for introducing product line engineering is the reduction of costs. When artefacts from the platform are reused in several different kinds of systems, this implies a cost reduction for each system. Before the artefacts can be reused, investments are necessary for creating them. In addition, the way in which they shall be reused has to be planned beforehand to provide managed reuse. This means that the company has to make an up-front investment to create the platform before it can reduce the costs per product by reusing platform artefacts (Pohl *et al.*, 2005).

Figure 2.1 shows the accumulated costs needed to develop *n* different systems. The solid line sketches the costs of developing the systems independently, while the dashed line shows the costs for product line engineering. In the case of a few systems, the costs for product line engineering are relatively high, whereas they are significantly lower for larger quantities. The location at which both curves intersect marks the break-even point. At this point, the costs are the same for developing the systems separately as for developing them by product line engineering. The precise location of the break-even point depends on various characteristics of the organisation and the market it has envisaged, such as the customer base, the expertise, and the range and kinds of products (Pohl *et al.*, 2005).

According to McGregor *et al.* (2002), the strategy that is used to iniciate a product line also influences the break-even point significantly. The strategies behind the product line engineering paradigm are outlined in Section 2.2.3.

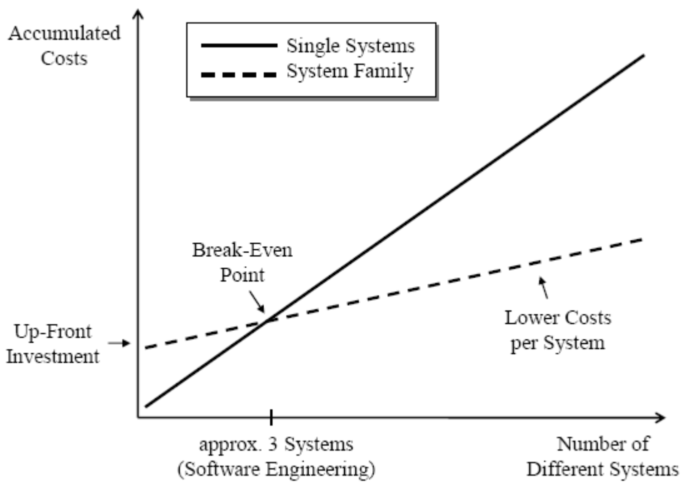


Figure 2.1 SPL Development Cost (Pohl *et al.*, 2005)

Moreover, a complete study achieved by Poulin (1997) shows that the development costs are recovered after two or three reuses.

Enhancement of Quality

Software product lines engineering also has a strong impact on the quality of the resulting software. A new application consists, to a large extent, of matured and proven components. This implies that the defect density of such products can be expected to be drastically lower than products that are developed all anew (Frank J. van der Linder and Schmid, 2007).

Thus, the artefacts in the platform are reviewed and tested in many products. They have to prove their proper functioning in more than one kind of product. The extensive quality assurance implies a significantly higher chance of detecting faults and correcting them, thereby increasing the quality of all products (Pohl *et al.*, 2005).

Reduction of Time to Market

According to Pohl *et al.* (2005), a very critical success factor for a product is the time to market. For single-product development, it is assumed that is roughly constant, mostly comprising the time to develop the product. Figure 2.2 shows that for product line engineering, the time to market indeed is initially higher, as the common artefacts have to be built first. Yet, after having passed this hurdle, the time to market is considerably shortened as many artefacts can be reused for each new product.

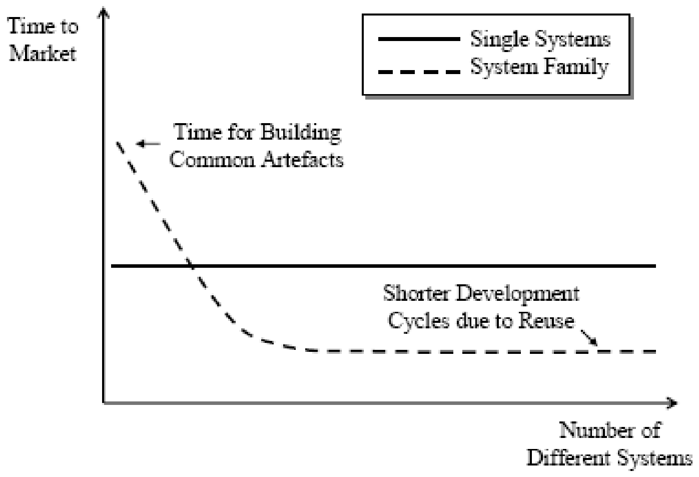


Figure 2.2 SPL Time to Market (Pohl *et al.*, 2005)

The improvements of costs and time to market are strongly correlated in software product line engineering: the approach supports large-scale reuse during software development (Frank J. van der Linder and Schmid, 2007). As opposed to traditional reuse approaches (Poulin, 1997), this can be as much as 90% of the overall software. Reuse is more cost-effetive than development by orders of magnitude. Thus, both development costs and time do market can be dramatically reduced by product line engineering.

Reduction of Maintenance Effort

Whenever an artefact from the platform is changed, e.g., for the purpose of error correction, the changes can be propagated to all products in which the artefact is being used. This may be exploited to reduce maintenance effort. At best, maintenance staff does not need to know all specific products and their parts, thus also reducing learning effort. However, given the fact that platform artefacts are changed, testing the products is still unavoidable. Yet, the reuse of test procedures is within the focus of product line engineering as well and helps reduce maintenance effort (Pohl *et al.*, 2005).

Usually, along with the reduction of development costs, a reduction of maintenance costs is also achieved. According by Frank J. van der Linder and Schmid (2007), several aspects contribute to this reduction; the most notably is the fact that the overall amount of code and documentation that must be maintained is dramatically reduced. As the overall size of the application development projects is strongly reduced, the accompanying project risk is reduced as well.

Coping with Evolution and Improving Cost Estimation

According to Pohl *et al.* (2005), the introduction of a new artefact into the platform or the change of an existing one gives the opportunity for the evolution of all kinds of products derived from the platform. Thus, the company has gains with the evolution and propagations of new or modified artefacts.

The activity of software estimation becomes straightforward and does not include much risk, once the core assets base used in SPL is already built. Consequently, the platform provides a sound basis for cost estimation.

Benefits for the Customer

Customers have the guarantee of getting products adapted to their real needs and wishes. Besides the advantages of customized products, users do not have to adapt their own way of working to the software anymore. In the past, it often happened that customers had to get used to a different user interface and a different installation procedure with each new product. This annoyed them, in particular as it even happened when replacing one version of a product by the next version. So, customers began to ask for improved software ergonomics. Accordingly, software packages were developed to support common user interfaces and common installation procedures. The use of such packages contributed to the proliferation of the idea of platforms. Moreover, customers can purchase these products at a reasonable price as product line engineering helps to reduce the production costs (Pohl *et al.*, 2005).

The benefits, on the oher hand, accrue with each new product release. Once the approach is established, the organization's productivity accelerates rapidly and the benefits far outweigh the cost. However, an organization that attempts to institute a product line without being aware of the cost is likely to abandon the product line concept before seeing it through. It takes a certainn degree of maturity in the developing organization to field a product line sucessfully. Technology change is not the only barrier to sucessful product line adoption. For instance, according to Clements and Northrop (2001), traditional organizational structures that simply have one business unit per product are generally not appropriate for product lines.

The following section presents the essential Software Product Lines Activities.

2.2.2 SPL Essential Activities

Each organization differs in a variety of terms, such as, the nature of this products, their bussiness goals, culture and policies and even their software process discipline. However, this diversity between all companies do not represents an impediment to the successfull of a software product lines adoption.

Nevertheless, Clements and Northrop (2001) distilled universal and essental software product line activities and practises that apply in every situation. At the highest level of generality are three essential activities illustrated in Figure 2.3.

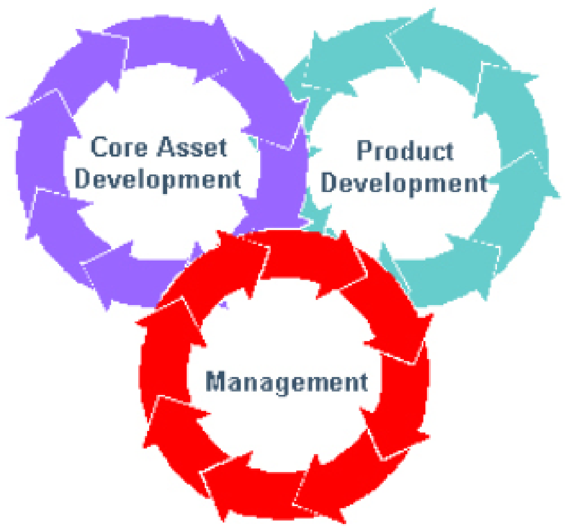


Figure 2.3 SPL Essential Activities (Clements and Northrop, 2001)

According to Clements and Northrop (2001), core asset development and product development from the core assets can occur in either order: new products are built from core assets, or core assets are extracted from existing products.

Each rotating circle described in Figure 2.3 represents one of the essential activities. All three are linked together and in perpetual motion, showing that they are all essential, linked, and highly iterative, and can occur in any order. Moreover, the rotating arrows indicates that revisions of existing core assets or even new core assets might, and most often do, evolve out of product development. Thus, it is given a neutral idea in regard to which part of the effort is launched first (Clements and Northrop, 2001).

These three essential activities are outlined in more detail next.

Core Asset Development

Core Asset Development has also been called domain engineering (Pohl *et al.*, 2005). The goal of the core asset development activity is to establish a production capability for products. Figure 2.4 illustrates the core asset development activity along with its outputs and influential contextual factors (Clements and Northrop, 2001).

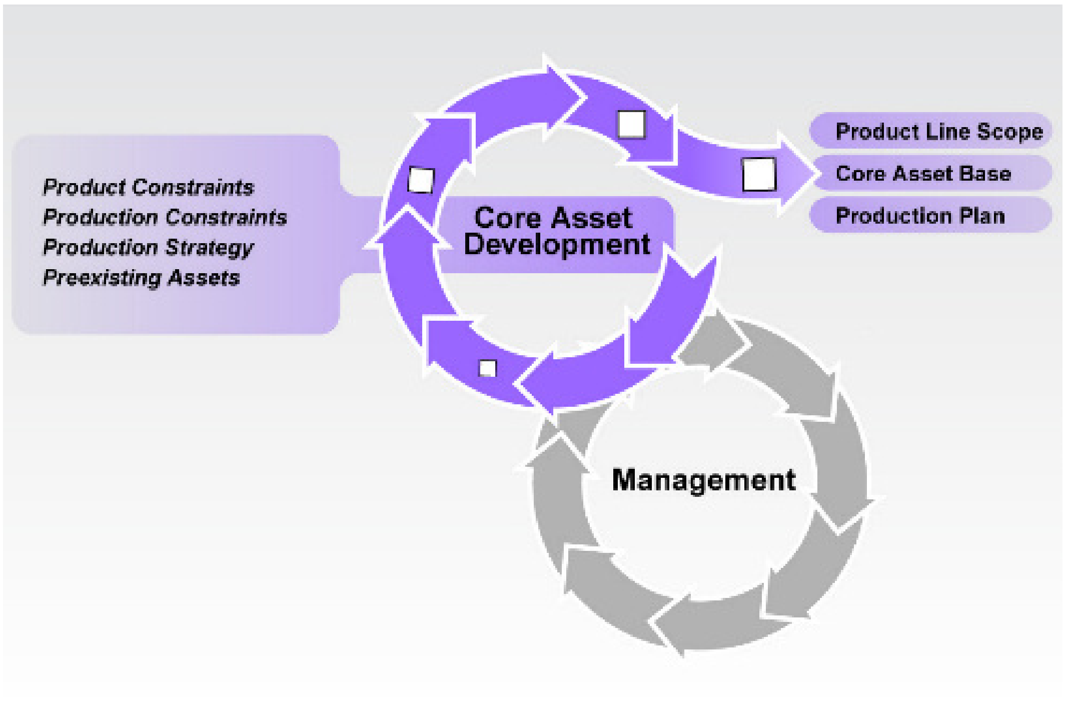


Figure 2.4 Core Asset Development (Clements and Northrop, 2001)

The necessary inputs required by a production capability to develop products are the outputs of the core asset development activity. They are Product Line Scope, Core Asset Base and Production Plan.

Product Line Scope is a description of the products that will constitute the product line or that the product line is capable of including. At its simplest, scope may consist of an enumerated list of product names. For a product line to be successful, its scope must be defined carefully. The scope of product line envolves as market conditions change, as the organization's plans change, as new opportunities arise, or as the organization quite simply becomes more adept at software product lines. Envolving the scope is the starting point for evolution the product line to keep it current (Clements and Northrop, 2001). Thus, the integration of technical and marketing-oriented product line planning is key to successful product line adoption (Frank J. van der Linder and Schmid, 2007).

According to Helferich *et al.* (2006), the advantages an organization can reap from product line engineering strongly depend on how well the product line infrastructure and the actual products that the organisation os going to develop are aligned.

Besides the traditional forms of define product line scope, Balbino (2010) realized a new and challenger study that presents an agile scoping process for product lines joining the benefits of the two approaches.

Another output of the core asset development activity are the *Core Assets* that almost certainly include an architecture that the products in the product line will share, as well as software components that are developed for systematic reuse across the product line.

A product line architecture is shared across many different products. Thus, it must capture the commonality of these products and deal with their diferences in an effective manner (Frank J. van der Linder and Schmid, 2007).

Finally, the last output of core asset development activity is the *Production Plan*, that describes how the products are produced from the core assets. The Product Plan describes the overall scheme for how these individual process can be fitted together to build a product. It is, in fact, the reuser's guide to product development within the product line (Clements and Northrop, 2001).

Therefore, these three outputs (the product line scope, core asset base, and production plan) are necessary ingredients for feeding the product development activity, which turns out products that serve a particular customer or market niche (Clements and Northrop, 2001).

Product Development

Product Development is also known as Application Engineering. The goal of the product development is to derive specific applications by exploting the variability of the softwre product line (Pohl *et al.*, 2005).

In general way, mature product line organizations priorize the health of the overall product line over that of individual products, but in the end, the activity of turning out the products is the ultimate product line goal (Clements and Northrop, 2001).

The product development activity depends on the three outputs outlined in the *Core Asset Development* Section: the product line scope, the core assets, and the production plan. Besides these outputs, the product development activity also uses the requirements for individual products as input. Figure 2.5 shows these relationships.

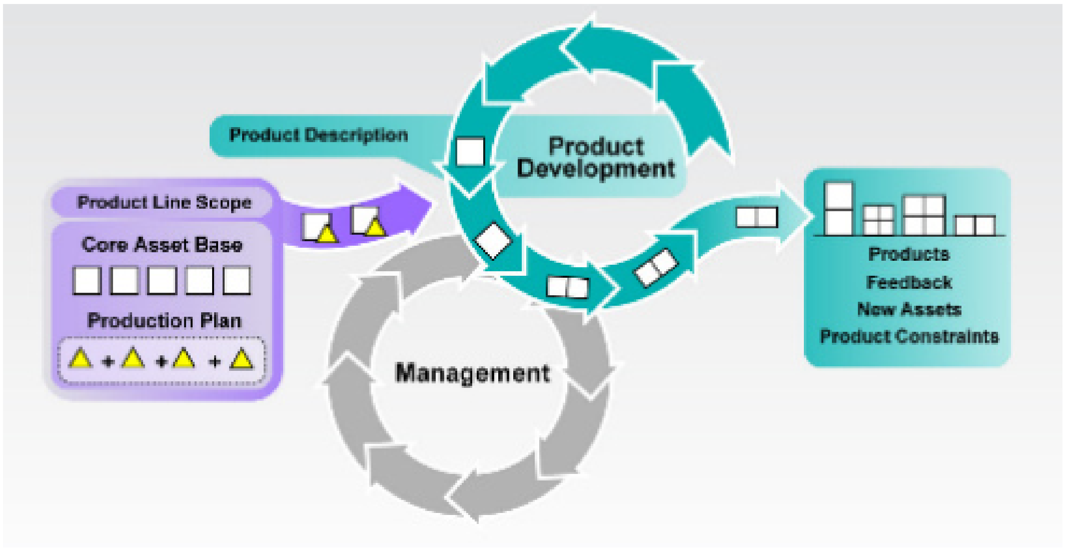


Figure 2.5 Product Development (Clements and Northrop, 2001)

As well in the *Core Asset Development* section, the rotating arrows indicate interation and intricate relationships. An appropriate example for this representation, building a product that has previously unrecognized commonality with another product already in the product line will create pressure to update the core assets and provide a basis for exploiting that commonality for future products (Clements and Northrop, 2001).

In a wider way, the main goal of the application engineering is to achieve an as high as possible reuse of the domain assets. Some specific goals are: to exploit the commonality and the variability of the software product lines during the development of a product line application; to document the application artefacts, i.e. application requirements, architecture, components, tests, and relate them to the domain artefacts. Last but not least, the activity of estimate the impacts of the differences between application and domain requirements on architecture, components, and tests (Pohl *et al.*, 2005).

Management

Management in the context of Core Assets and Product Development has a critial role of maintain the sucessful fielding of a product line (Pohl *et al.*, 2005). Thus, both technical and organizational levels must be strongly committed to the software product lines effort. That commitment manifests itself in a number of ways that feed the product line effort and keep it healthy and vital (Clements and Northrop, 2001).

Commonly the practice of management identifies production constraints and ultimately determines the production strategy. This activity should ensure that these operations and the communication paths of the product line effort are documented in an operational concept.

According Clements and Northrop (2001), some individual or group should be designated to either fill the product line management role and act as a product line champion or find and empower one. That champion must be a strong, visionary leader who can keep the organization squarely pointed toward the product line goals, especially when the going gets rough in the early stages.

2.2.3 SPL Strategies

The Software Product Lines strategy describes how the product line practices should be employed so that the product line organization will achieve its production goals. Thus, the production strategy for a software product lines is the high-level description of how the production system performs both the core assets and products. The production strategy is derived from the organization business strategy and is intended to coordinate the actions of the core asset and product developers (Chastek *et al.*, 2009).

There are some strategies for developing SPLs (Pohl *et al.*, 2005): proactive, reactive, and extractive.

With the *proactive* approach, the organization analyzes, designs and implements a fresh Software Product Lines to support the full scope of products needed on the foreseeable horizon. Consequently, from the analysis and design, a complete set of common and varying source code, feature declarations, product definitions, and automata are implemented.

In contrast, in the *reactive* approach, the organization incrementally grows an existing Software Product Lines when the demand arises for new products or new requirements on existing products. With this strategy, the common and varying source code, along with the feature declarations, product definitions, and automata, are incrementally extended in reaction to new requirements (Krueger, 2002).

Finally, in the the *extractive* approach, the organization extracts existing products into a Software product Lines. Thus, this high level of software reuse enables an organization to very quickly adopt software mass customization (Krueger, 2002).

Although these approaches have different strategies to work during the process of SPL adoption, they aren not mutually exclusive. For example, a common approach is to bootstrap a software mass customization effort using the extractive approach and then move on to a reactive approach to incrementally evolve the production line over time (Krueger, 2002).

2.3 Chapter Summary

Software Product Lines is a very successful approach to achieve software reuse. A good software reuse adoption facilitates the increase of productivity, quality, and reliability, and the decrease of costs and implementation time. In this context, this chapter summarized the basic concepts about software product lines and their aspects, such as: SPL Motivations, the related benefits to use and adoption of SPL and the commonly strategies used in the SPL adoption.

Next Chapter presents an overview about Traceability in order to identify the concepts and associated techniques. Moreover, it outlines the specific aspects about the Traceability in the context of Software Product Lines and the issues of Impact Analysis area.

3 An Overview on Traceability and Impact Analysis

E não diga que a vitória está perdida
Se é de batalhas que se vive a vida
Tente outra vez!

Don't say that the victory is lost
Because life is a succession of battles
Try one more time!
—RAUL SEIXAS (Musician)

Traceability refers to the ability to link different software artefacts, therefore, traceability of software artefacts has been recognized as an important approach for supporting various activities in the software system development process. In general, the objective of traceability is to improve the quality of software systems. More specifically, the traceable software artefacts can be used to support the analysis of implications and integration of changes that occur in the software systems (Bennett and Rajlich, 2000); the maintenance and evolution of software systems; the reuse of software system components by identifying and comparing requirements of new and existing systems; the testing of software system components; and system inspection, by indicating alternatives and compromises made during development (von Knethen and Paech, 2002). More succinctly, Kotonya and Sommerville (1998) emphasize that traceability is mainly concerned to maintaining links between different core assets.

According to the various interrelationships of artefacts within a project, software traceability has been recognized by both researchers and practitioners as a key factor for improving software development. Potential benefits of traceability include better impact analysis, lower maintenance costs, and better assessment of product quality (Asuncion, 2008). Besides these benefits, traceability also enables system acceptance by allowing users to better understand the system and contributes to a clear and consistent system documentation.

In the context of Software Product Lines Engineering, the traceability of software artefacts is an important factor when it comes to effective development and maintenance of software system. Traceability management facilitates the SPL artefacts to remain in synchronous state and ensure the consistency of derived products (Abid, 2004).

Thus, the remainder of this Chapter is organized as follows. In Section 3.1, are introduced the concepts of Software Traceability. Next, the main topics of the Traceability area in the context of Software Product Lines are discussed in Section 3.2. The concepts of Impact Analysis are depicted in Section 3.3 and finally, the Chapter Summary is described in Section 3.4.

3.1 Traceability

During the software development lifecycle, a lot of software artefacts are generated at each phases and there are huge amounts of complex traceability links among them (Zhou *et al.*, 2008). According to Lindvall and Sandahl (1996), the establishment of traceability relations has makes the documentation of a system clear and consistent, and makes the process of maintaining the system less dependent on individual experts.

von Knethen and Paech (2002) describe traceability as the ability to determine which documentation entities of a software system are related to which other ones, according to specific relationships. Another accepted definition given by IEEE (1990) is that ”Traceability is the degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another; for example, the degree to which the requirements and design of a given software component match.”

According to Spanoudakis and Zisman (2004), software traceability is the ability to relate artefacts created during the development of a software system to describe the system from different perspectives and levels of abstraction with each other, the stakeholders that have contributed to the creation of the artefacts, and the rationale that explains the form of the artefacts - has been recognized as a significant factor for any phase of a software system development and maintenance process, thus contributing to the quality of the final product. Its importance has been widely recognized in software process standards and in government regulations (IEEE, 1998) (Ramesh and Jarke, 2001).

Consistently, von Knethen and Paech (2002) describe that besides the tracing approach captures and manages relationships between documentation entities, a trace should also capture the human cooperation in the design process, that is, how stakeholders contribute to the development (e.g., maintainer, project manager, or tester) in performing their tasks (e.g., changing, controlling, or testing). In this way, tracing approach is used to establish traceability.

Over the last few years, many work addressing various aspects of traceability proposed by system engineering communities have been done to enhance traceability between developed artefacts throughout the development lifecycle, such as following.

In the context of vizualization area, Cleland-Huang and Habrat (2007) proposed an approach to display the candidate links to the user in a relatively bland textual format. This work described several visualization techniques for helping analysts to evaluate sets of candidate links.

There are also other work in the same direction conducted by Cleland-Huang and Habrat (2007), that proposed a special visualization technique - ENVISION, which is intended to facilitate major software traceability understanding tasks with viewing, navigating, focusing, searching and filtering. In Addition, ENVISION provides other useful functions like dual visualization mode, historical navigation path, round trip visualization. ENVISION is implemented in Eclipse for a traceability visualizing (Zhou *et al.*, 2008).

In the context of automated traceability approaches, Alexander (2002) described the experience with a toolkit that helped to reduce the burden of installing traces and allowed readers to view traces with familiar tools. Three traceability tools were used: an analyser that can automatically link Use Case references to Use Cases; a dictionary builder that links and if need be creates definitions from marked-up terms; and an exporter that translates a database of Use Cases into a fully-navigable and fully-indexed hypertext.

In the Information Retrieval (IR) area, Hayes *et al.* (2003) presented an approach for improving requirements tracing based on framing it as an IR problem. Specifically, they focus on improving recall and precision in order to reduce the number of missed traceability links as well as to reduce the number of irrelevant potential links that an analyst has to examine when performing requirements tracing.

Overall, research into software traceability has been mainly concerned with the study and definition of different types of traceability relations; support for the generation of traceability relations, which is the focus of this work in the context of SPL; development of architectures, tools, stakeholders' purpose and environments for the representation and maintenance of traceability relations; and empirical investigations into organizational practices regarding the establishment and deployment of traceability relations in the software development life cycle (Spanoudakis and Zisman, 2004).

3.1.1 Purpose of Stakeholders

Although traceability has been studied in practice for over two decades, there has yet to be a consensus on what information should be captured and used as a part of a traceability scheme. There are many different definitions of traceability, each changing with a stakeholder's view of the system (Ramesh *et al.*, 1995). Different stakeholders are interested in different types of relations. For Kotonya and Sommerville (1998), system stakeholders are people or organisations who will be affected by the system and who have a direct or indirect influence on the system requirements. For example, end users may be interested in relations between requirements and design objects as a way of identifying design components generated by or satisfying requirements; designers may be interested in the same type of relations but as a way of identifying the constraints represented as requirements associated with a certain design object (Spanoudakis and Zisman, 2004). In the context of single software, stakeholders could be the customer, the project manager, the system analyst, the system designer, the test engineer, system maintenance personnel, or the end user of the system.

Traceability provides stakeholders with a means of showing compliance with requirements, maintaining system design rationale, showing when the system is complete, and establishing change control and maintenance mechanisms.

In a general way, the purpose of a tracing approach depends on the stakeholder who is interested in the traceability information and the task of the stakeholder that should be supported by the traceability. In this context, purpose characterizes different tracing approached according to the stakeholders' view of traceability (von Knethen and Paech, 2002). These purposes are discussed following (von Knethen and Paech, 2002).

- **Project Manager:** Project managers use the traceability information to provide a control project progress. The traceability matrix provides the manager a means of tracking staff progress on the project. Consequently, some project managers believe that proper use on traceability provides a means of showing full control of the project. The project manager uses traceability to track project status similar to the way upper management does, only in more detail. For instance, a GANTT chart generated from the traceability information is used by the project manager in developing weekly project status reports (Ramesh *et al.*, 1995). Also, traceability helps the project manager in detecting project delays. Finally, the project manager plans to use traceability to verify and prove to the customer that the system meets the stated requirements and the job is complete. By using traceability to acceptance test plans for every validated requirement, including derived requirements, the project manager can prove to the customer that the system ”completely” meets their needs.

- **Customer:** Traceability ensures the customer’s satisfaction by proving that all of the stated requirements are met and that the job is completed. In addition, the effect of a required change can be demonstrated. These activities are executed by the **Project manager**.
- **Project planner:** Project planners are responsible for planning, organizing, securing and managing activities. Thus, a project planner uses a tracing approach to perform impact analysis. Requirements can be tracked to determine the impact of a required change.
- **Tester:** The system testers plan on using traceability in writing the acceptance test plan. Being a tester is about improving the quality of the product before it is complete. Thus, making use of the traceability tool, the testers will verify that the Acceptance Test Plan tests all of the system requirements, thus ensuring completeness and that it operates as it is designed to, while meeting all of the customer’s requirements.
- **System Engineering:** Although the project is not at the stage of maintenance, the system designer foresees using requirements traceability extensively in tracing changes to code modules and documentation due to the complexity of the engineering projects and management over the life of the project.
- **System Designer:** Designers use the traceability information to understand dependencies between the requirements and to check whether all requirements are considered by the design. The system designer also plans on using traceability to determine which test plans and documentation are effected by a change so that they

could be updated and rewritten. The software designer use traceability as a ”fit and function” verification tool.

- **Requirements engineer:** Requirements engineers is responsible for manage changes to the system requirements (Kotonya and Sommerville, 1998), thus the traceability information is used to check correctness and consistency of the requirements.

- **Validator:** Validators use traceability relationships between requirements and test plans to prove that the system ”completely” meets the needs of the customer. In addition, test procedures can be identified that should be rerun to validate an implemented change. This saves test resources and allows the schedule to be streamlined.

- **Maintainer:** Maintainers use the traceability information to decide how a required and accepted change will affect a system. Maintainers could identify which modules are directly affected and which other modules will experience residual effects. Documenting an engineer’s design rationale helps the maintainer to understand the system. If a required change is implemented, understanding the existing solution structure helps to prevent the system from degrading.

In a case study conducted by Ramesh *et al.* (1995), are identified that from the upper level management to the system maintenance personnel, every person believes that traceability needed for the successful completion of a project and that without it, their organization’s success would be in jeopardy. However, there is no agreement in the literature as to which conceptual trace models are necessary to support which stakeholder and task.

In general way, stakeholders with different perspectives, goals and interests who are involved in software development may contribute to the capture and use of traceability information. Depending on their perceptions and needs, they may influence the selection of different types of traceability relations which are used in software development projects, and can establish project specific conventions for interpreting the meaning of such relations (Spanoudakis and Zisman, 2004). An overview of the main types of traceability relations that have been proposed in the literature is outlined in the following session.

3.1.2 Types of Traceability Relations

Existing approaches and tools for traceability support the representation of different types of relations between system artefacts but the interpretation of the semantics associated with a traceability relation depends on the stakeholders, as described in the preview Section 3.1.1.

The semantic associated with the traceability relations leads to a series of related studies. Dick (2002) shows that in industrial settings, the traceability relations are very shallow and it is necessary to represent deeper and richer semantic traceability relations. He proposes the use of textual rationale and propositional logic in the construction of traceability arguments. According to Pinheiro and Goguen (1996), traceability relations should have precise semantic definition to avoid the problem of culture-based interpretations. On the other hand, Bayer and Widen (2002) suggested that in order to increase the use of traceability and, therefore, compensate for its cost, traceability relations should have a rich semantic meaning instead of being bi-directional referential relations.

In order to overcome the lack of standard semantic interpretation of traceability relations and establish meaningful forms of semantics for traceability relations, Spanoudakis and Zisman (2004) organized various types of traceability relations proposed in the literature into eight main groups namely: dependency, generalisation, evolution, satisfaction, overlap, conflicting, rationalisation, and contribution relations.

In a general way, traceability relations denote overlap, satisfiability, dependency, evolution, generalization, conflict, contribution and rationalisation associations between various software artefacts and the stakeholders that have contributed to their construction. These groups are described next in details (Spanoudakis and Zisman, 2004) . In this explanation about these types of traceability, is used the term element in a general way to represent the different parts, entities, and objects in software artefacts that are traceable, such as: stakeholders, requirements statements, classes and code statements. Thus, two elements *e1* and *e2*, in different or in the same software artefact can be related by more than one type of relations.

- **Dependency relations:** This type of relations, an element *e1* depends on an element *e2*, if the existence of *e1* relies on the existence of *e2*, or if changes in *e2* have to be reflected in *e1*. For instance, Ramesh and Jarke (2001) propose the use of dependency relations between different requirements, and between requirements and design elements.

- **Generalisation relations** This type of relationship is used to identify how complex elements of a system can be broken down into components, how elements of a system can be combined to form other elements, focusing on specific concepts and how an element can be refined by another element (Spanoudakis and Zisman, 2004).

- **Evolution relations:** Relations of this type means the evolution of elements of software artefacts. In this case, an element *e1* "evolves to" an element *e2*, if *e1* has been replaced by *e2* during the development, maintenance, or evolution of the system.
- **Satisfiability relations:** In this type of relations, an element *e1* satisfies an element *e2*, if *e1* meets the expectation, needs, and desires of *e2*; or if *e1* complies with a condition represented by *e2*. This type of relations is classified within the condition link group, which associates restrictions to requirements and contains constraints and pre-condition links (Pohl, 1996b).
- **Overlap relations:** In this type of relations, an element *e1* overlaps with an element *e2*, if *e1* and *e2* refer to common features of a system or its domain. In (Jirapanthong and Zisman, 2009), an example of this relation exists between operations in the sequence diagram and the descriptions in the use case, since this description contains the name of the operation and the name of the class of the object of this operations is the sequence diagram. While in (Spanoudakis and Zisman, 2004), overlap relations are used between requirement statements, use cases, and analysis object model.
- **Contribution relations:** This type of relations are used to represent associations between requirement artefacts and stakeholders that have contributed to the generation of the requirements. Gotel and Finkelstein (1995) present an approach, based on modelling the dynamic contribution structures underlying requirements artefacts, which addresses this issue. More specifically, this approach supports requirements pre-traceability, that is the ability to relate a requirement with the stakeholders that expressed it and contributed to its specification.
- **Conflict relations:** This type of relations means conflicts between two elements *e1* and *e2*. In (Ramesh and Jarke, 2001), conflict relations are used to identify conflicts when two requirements conflit with each other. This type of conflict relations also are used to components, and design elements, to define issues related to these elements, and to provide information that can help in resolving the conflicts and issues.
- **Rationalisation relations:** Relations of this type are used to represent and maintain the rationale behind the creation and evolution of elements, and decisions about

the system at different levels of detail. Ramesh and Jarke (2001) capture the rationalisation relations based on the history of actions of how elements are created.

According to Ramesh and Jarke (2001), most of the existing approaches in the literature have proposed different types of traceability relations that relate requirements specifications, and requirements with design specifications and requirements and design artefacts. Ramesh and Jarke (2001) attributed this to the fact that traceability was initially proposed to describe and follow the life of a requirement, such as requirement traceability. In additional, Ramesh and Jarke (2001) describe in details the establishment of traceability relations involving code specifications and other software artefacts is not an easy task.

Another classification identified in the literature is related with the dependency of condition whether traceability relations associate elements of the same artefacts or elements of different artefacts, they can be distinguished into vertical and horizontal relations, respectively.

As described in this Section, much reference models have also been proposed to support semantic interpretation of traceability relations in the literature, but they are not an easy task, besides being error prone on the maintanence activities. In the Chapter 4, we describe in details an metamodel that proposes a representation of traceability information including the different traceable elements and possible traceability scenarios of recommendation.

In the following Section is outlined the different options of traceability generation described in the literature.

3.1.3 Generation of Traceability

As described in the beginning of this Chapter, potential benefits to traceability include better impact analysis, lower maintenance costs, and better assessment of product quality (Asuncion, 2008). However, despite these advantages, traceability is difficult to achieve in practice, because users still face many problems when employing traceability tools and techniques.

The majority of contemporary requirements engineering and traceability tools offer only limited support for traceability as they require users to create traceability relations manually. Accordingly to Asuncion *et al.* (2007), these approaches are generally infeasible, because most of the manual approaches require high overhead and are viewed by software engineers as imposed work.

In order to mitigate the problem, some approaches which support automatic or semi-automatic generation of traceability relations have been proposed. Soon thereafter, are described the different approaches for traceability generation based on the level of automation that they often which ranges from manual, to semi-automatic, and fully automatic generation.

Manual Generation

Manual generation of traceability is normally supported by visualisation and display tool components, in which the documents to be traced are displayed and the users can identify the elements in the documents to be related in an easier way (Spanoudakis and Zisman, 2004).

An example of this approaches was presented in (Pohl, 1996a) that presents a requirements engineering environment, called PRO-ART, which enables requirements pre-traceability. PRO-ART is based on some main contributions: a three-dimensional framework for requirements engineering which defines the kind of information to be recorded; a trace-repository for structuring the trace information and enabling selective trace retrieval.

Despite the numerous advantages, the manual creation of traceability relations is difficult, error prone, time consuming and complex. Thus, an effective traceability is rarely established manually in industry.

Semi-automatic Generation

Spanoudakis and Zisman (2004) classify the semi-automatic traceability generation approaches into two groups: **pre-defined link** group, that is concerned with the approaches in which traceability relations are generated based on some previous user-defined links, and **process-driven** group, that is concerned with the approaches in which traceability relations are generated as a result of the software development process.

The semi-automated approach appeared in order order to overcome the issues associated with the manual generation of traceability relations. Some approaches have been proposed in which traceability relations are generated in a semi-automatic way as follows.

In the context of a pre-defined group approach, Egyed (2003) presented an semi-automated approach to generating and validate trace dependencies. This work addressed the severe problem that the lack of trace information or the uncertainty of its correctness limits the usefulness of software models during software development. It also automates what is normally a time consuming and costly activity due to the quadratic explosion of potential trace dependencies between development artefacts.

Although the above approaches may be considered an improvement when compared with the manual approaches, the identification of the initial user-defined links required by some of the approaches may still cause traceability to be error prone, time consuming, and expensive (Spanoudakis and Zisman, 2004).

Automatic Generation of Traceability Relations

In order to overcome the issues associated with the manual and semi-automated generation of traceability relations, some recently approaches have been proposed in which traceability relations are generated in a automatic way. Spanoudakis and Zisman (2004) identify three proposals of approaches to support automatic generation of traceability relations. Some of these approaches use Information Retrieval - IR techniques, others use traceability rules, special integrators and inference axioms.

In the context of Information Retrieval IR, Antoniol *et al.* (2002) proposed a method based on IR to recover traceability links between source code and free text documents. A premise of this is that programmers use meaningful names for program items, such as functions, variables, types, classes, and methods. They believes that the application-domain knowledge that programmers process when writing the code is often captured by the mnemonics for identifiers; therefore, the analysis of these mnemonics can help to associate high-level concepts with program concepts and vice-versa. This work has been reported to produce traceability relations at low levels of precision and reasonable levels of recall. It should be noted, however, that the relations produced by this approach could only represent overlap relations between elements in different system artefacts that refer to common features of a system. With the same technique, Hayes *et al.* (2003) present an approach for improving requirements tracing based on framing it as an information retrieval IR problem. Specifically, this work focus on improving recall and precision in order to reduce the number of missed traceability links as well as to reduce the number of irrelevant potential links that an analyst has to examine when performing requirements tracing.

In the context of rule-based technique, Zisman *et al.* (2003) presents an approach for automatic generation and maintenance of bi-directional traceability relations between commercial and functional requirements expressed in natural language, and requirement object models. The generation of traceability relations is based on two types of traceability rules: requirements-to-object-model rules and inter-requirements rules.

Sherba *et al.* (2003) describe a framework for automating the management of traceability relationships chaining, called TraceM. According to Sherba *et al.* (2003), a key contribution of TraceM is its ability to transform implicit relationships into explicit relationships by processing chains of traceability relationships. This approach uses special integrators, which can discover and create traceability relations between software artefacts and other previously defined relations.

In a tool called TOOR (Traceability of Object-Oriented Requirements), traceability is defined and derived in terms of axioms. Based on these axioms, the tool allows automatic identification of traceability relations between requirements, design, and code specifications (Pinheiro and Goguen, 1996).

Although none of the above approaches can fully automate the generation of traceability relations, they have taken significant steps toward this direction.

According to Ramesh *et al.* (1995), the costs associated with implementing a comprehensive traceability scheme can be justified in terms of better quality of the product and the systems development and maintenance process with potentially lower life cycle costs.

Some of the points described in this Section are strongly present in software product lines. The Mapping Study in the Traceabilify for SPL, and the relation between software product lines and traceability are outlined in the following Section.

3.2 Traceability for SPL

The Software Product Lines methods and techniques aim at producing software system families with high level of quality and productivity (Pohl *et al.*, 2005). As discussed in the preview Chapter, the development of a SPL is typically organized in domain and application engineering. The focus of domain engineering is the production of a set of core assets to scoping, specification and modeling of the common and variable features of a Software Product Lines; the definition of a flexible architecture that comprise the SPL common and variable features; and the implementation of the SPL architecture. In an application engineering process, a feature model configuration can be used to compose and integrate the core assets produced during the domain engineering stage in order to generate and instance of the SPL architecture.

Several authors believe that traceability is a fundamental condition for the SPLs sucess. According to Streitferdt (2001), traceability throughout model elements is a necessary precondition for preserving the consistency of the complete family model during development. Likewise, Sousa *et al.* (2009) show that traceability is fundamental to guarantee and validade the quality of SPL development and to allow a better management of SPL variabilities. For Jirapanthong and Zisman (2005), traceability relations can improve the quality of the product being developed, and reduce the development time and cost in the SPL. Researchers have widely acknowledged the potential of traceability relations in identifying reusable artefacts in the software development life-cycle. According to Spanoudakis and Zisman (2004), these artefacts may be at different levels of abstraction (e.g. source code, design or requirement artefacts) and can be identified and reused through different scenarios, which require the existence of different types of traceability relations.

Next, is outlined the Research Strategy conducted to provide an overview of Traceability in the context of the Software product Lines area.

3.2.1 Research Strategy - SPL Traceability Literature

Compared to single systems development, the traceability for Software Product Lines becomes more complex and challenging, since it involves more artefacts and more different stakeholders. In order to address part of these challenges, some approaches have been proposed by industry and academy. Many of these approaches are reported on the scientific literature, thus it is necessary a strategy to map the studies that report those approaches.

The research strategy adopted to relate the main available approaches of the Traceability area in the context of Software Product Lines was carried out based on some practices of the Systematic Mapping Study (Petersen *et al.*, 2007). The Systematic mapping is a methodology that is frequently used in medical research, but that have largely been neglected in Software Engineering (SE). The main practices of the systematic mapping defined by Petersen *et al.* (2007) are: definition of the research questions, conducting the search for relevant papers, screening of papers, keywording of abstracts and data extraction and mapping.

The remain of this Section presents the good practices of Systematic Mapping Study adopted in our Research Strategy, but without using all the activities proposed by Petersen *et al.* (2007).

Research Questions

The research questions guide the design of the research strategy to provide an overview of a research area, and identify the main and relevant related work. Thus, this study is intended to answer the following research questions:

Q1.Which techniques and areas are addressed by the Traceability in the context of SPL? This question aims identifing the Traceability approaches in the SPL context, such as: rule based, information retrieval, metamodel.

Q2. Which type of traceability approaches are adopted in the context of SPL? This question aims identifing the the types of existing traceability approaches in the context of SPL, such as: manual generation, semi-automatic generation and automatic generation.

Following are described how the research was conducted. If involves the steps adapted from (Petersen *et al.*, 2007), such as: Search Strategy and Data sources.

Search Strategy

The first strategy to conduct our research was to create the search string because the primary studies are identified by using search strings on scientific databases or browsing manually through relevant conference proceedings or journal publications (Petersen *et al.*, 2007). According to Kitchenham and Charters (2007), a good way to create the search string is to structure them in terms of population, intervention, comparison, and outcome. The structure should be driven by the research questions and are constructed using boolean ANDs and ORs to combine keywords.

We used various combinations of search items to achieve a more adequate set of keywords, they are: software product lines, 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). Thus, this research strategy defines the two following search strings:

- **Search With Software Product Line criterias:** *(”Software Product Line” OR ”product line” OR ”product lines” OR ”software family” OR ”software families” OR ”spl” OR ”ple”) AND (”traceability” OR ”tracing” OR ”trace”) AND (”automate” OR ”automated” OR ”automatic” OR ”semi-automatic” OR ”semi-automated” OR ”rule-based” OR ”visualization” OR ”information retrieval” OR ”ontology” OR ”ontological” OR ”semantic index” OR ”latent semantic indexing” OR ”semantic indexing” OR ”generation”)*

- **Search without Software Product Line criterias:** *(”Software”) AND (”automate” OR ”automated” OR ”automatic” OR ”semi-automatic” OR ”semi-automated” OR ”rule-based” OR ”visualization” OR ”information retrieval” OR ”ontology” OR ”ontological” OR ”semantic index” OR ”latent semantic indexing” OR ”semantic indexing” OR ”generation”)*

Data Sources

The choice of databases was performed in search engines and digital libraries of the most popular publishers and organizations in software engineering. Beside the analysis of the journals, conferences, proceedings, books and technical reports, we also analyzed the references of the gray literature. The research was done through last ten years of conferences and journals. The used strategy is explained in three steps:

1. Since usually the most important work are published on journals (from IEEE, ACM and Springer), they were the first place search.

2. After that, the papers were searched in important conferences in the Software Engineering. The searched conferences were:

    - International Conference of Software Engineering (ICSE);
    - International Conference on Software Reuse (ICSR);
    - International Conference on Software Maintenance (ICSM);
    - International Requirements Engineering Conference (IREC);
    - International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA);
    - Software Product Line Conference (SPLC);
    - International Conference on Enterprise Information Systems (ICEIS);
    - International Conference on Software Process (ICSP);
    - International Conference/Workshop on Program Comprehension (ICPC);
    - International Conference on Information Reuse and Integration (IRI);
    - European Conference for Object-Oriented Programming (ECOOP);
    - Empirical Software Engineering and Measurement (ESEM);

- European Software Engineering Conference (ESEC);
- European Conference on Software Architecture (ECSA);
- International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS);
- Asia Pacific Software Engineering Conference (APSEC);
- International Conference on Automated Software Engineering (ASE);
- International Computer Software and Applications Conference (COMPSAC);
- International Conference on Advanced Information Systems Engineering (CAiSE);
- European Conference on Software Maintenance and Reengineering (CSMR);
- Technology of Object-Oriented Languages and Systems Conference (TOOLS);
- Working IEEE/IFIP Conference on Software Architecture (WICSA);
- Working Conference on Reverse Engineering (WCRE);
- Fundamental Approaches to Software Engineering (FASE);
- International Conference on Quality Software (QSIC);
- Euromicro Conference on Software Engineering and Advanced Applications(SEAA);
- International Conference on Generative Programming and Component Engineering (GPCE);
- International Conference on Software Engineering and Knowledge Engineering (SEKE);
- International Conference on Composition-Based Software Systems (ICCBSS);
- International Conference on the Quality of Software Architectures (QoSA);
- International Symposium on Component-based Software Engineering (CBSE);
- International Conference on Model Driven Engineering Languages and Systems (MODELS).

3. As next step, the search string defined just above was used to filter potential work that are not found in the second step. The main databases used to find these papers were: Scopus, IEEEXPlore, ACM and SpringLink.

4. Finally, papers referenced by the authors of the found papers were also analyzed.

Studies Selection

As soon as the potentially relevant primary studies have obtained, according to the match of the search items and research questions, the studies found needed to be assessed for their actual relevance. In order to obtain these results, are shown shortly after the inclusion and exclusion of the objects. According to Petersen *et al.* (2007), inclusion/exclusion criteria are used to exclude studies that are not relevant to answer the research questions, thus it useful to exclude papers which only mentioned our main focus, *traceability* and *Software Product Lines* in the abstract. This was important since *traceability* and *Software Product Lines* are the central concepts in the area, thus is frequently used in abstracts without papers really addressing it any further. The criteria are next detailed:

Inclusion Criteria: Studies that present at least one of the following topics:

- Traceability for Software Product Lines.
- SPL Meta-models. It is directly related with traceability for SPL.
- Software Product Lines Variability. Variability is a key concept for SPL modeling and the traceability aspects are important points that must be analyzed.

The inclusion criteria establish the reasons for each study found in the previous step. The abstract explicitly mentions these following topics, because from the abstract, the researcher is able to deduce that focus of the paper contributes to traceability in the context of software product lines engineering.

Exclusion Criteria: the criteria that describes the reasons to discart studies found in the previous step are:

- The paper lies outside the software engineering domain.
- A work where traceability and software product lines are not part of the contributions of the paper or with insufficient information in traceability.
- A work that is not an approach. It involves only concepts or issues related with traceability for software product lines.
- A work that the term are only mentioned in the general introductory sentences of the abstract.
- Finally, the duplicate studies are ignored when has been published in more than one publication, the most complete version will be used.

In general, for each selected study was applied the reading of the following elements: titles, abstract, keywords, conclusion and references. These studies were analyzed with focus on the inclusion and exclusion criteria. The results of this research strategy is vizualized in the Section 3.2.2 (related works), which are described the main available and relevant works in the area.

Data Analysis

The first stage resulted in a set of 66 studies raised from the web search, through the use of search strategy described previously. However, titles are not always clear indicators of what a study is about. The possibility of finding not so relevant studies was considerated, therefore in a next stage inclusion and exclusion criteria described previously were applied to them, which resulted in the reduction of 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 (Kitchenham and Charters, 2007). In the next stage, the whole text was visited in order to have a critical viewpoint on the topic addressed by the study. Finally, in the end of this stage, many studies that lies out of the inclusion criteria was excluded.

Finally, these good practices addressed the research strategy to the process of identification the main related and available work in the gray literature. A description about some selected approaches in the area are outlined in detail in the next Section.

3.2.2 Research Results

According to the research approach based on some good practices of Systemactic Mapping Studies proposed by Petersen *et al.* (2007) and 28 studies were selected.. The following works bring differents traceability approaches in the context of Software Product Lines.

In order to apply SPL practically and to automate SPL process, elements of artefacts and their relationships should be precisely specified. In this context, Streitferdt (2001) proposed a requirements metamodel for system family development. As mentioned in the beginning of the Chapter, Streitferdt (2001) puts traceability throughout model element as a necessary precondition for preserving the consistency of the complete family model during development. According to Streitferdt (2001), current methods such as FODA (Kang *et al.*, 1990a) and FeatuRSEB (Griss *et al.*, 1998) address the requirements engineering phase by high level constructs such as use-cases. This is insufficient, since not all systems are workflow oriented being needed a lower level approach. Thus, the proposed solution consists of a metamodel upon which a low level model of requirements can be built for developing system families. In addition, this work identifid the necessity of a tool to perform consistency checking and analyses. The prototype was used in academic projects and a cooperation project with industry partners, to evaluate, refine and validate the proposed metamodel in practice.

Likewise, Kim *et al.* (2005) proposed a generic Product Line Engineering (PLE) process based on a survey to identify the representative PLE approaches. Then, this work defined a meta-model of PLE artefacts, to show how each artefact can be represented at conceptual and physical levels. Finally, were specified an overall traceability map, trace links and mapping relationships. For Kim *et al.* (2005), through the foundation of PLE traceability, the PLE artefacts such as core assets and instantiated products can be more efficiently and correctly developed. According to this proposal, family development based on the metamodel guarantees traceability by the inclusion of all development artefacts in a single and consistent model.

Another work suggested a metamodeling approach to support the tracing of variability in requirements and architecture in a product line (Moon *et al.*, 2007). Two metamodels representing the domain requirements and domain architecture with variability were proposed. In a general way, the variations in artefact constituents are determined based on several trace matrices. It must be demonstrated that decisions as to whether properties of the requirements are common or optional are rational, because these decisions affect the properties of subsequent core assets.

The work proposed by Moon *et al.* (2007) realized a study and verified that many features modeling approaches are used to represent variability in the problem space, i.e., at the requirements level, or in the solution space, i.e., at the architecture or source code level. However, Moon *et al.* (2007) emphasized that while these approaches appropriately address variability issues at each level of abstractions in a product line development, they do not address issues with regard to tracing between the requirements and the architecture. In other words, they only describe the means of mapping features to the architecture or design elements and they do not describe traceability with respect to variation points in the requirements and architecture, which are considered crucial for consistent and efficient development of product line based applications. Although Berg *et al.* (2005) describe the importance of tracing variability in a product line and proposes a conceptual model for traceability, it does not provide a detailed description of traceability dependencies between requirement and architecture, such as requirement and architecture models and specific dependencies with respect to the variability between these.

In summary, this work presents a metamodel, the basis for tracing the variabilities with a clear description between two phases in a product line.

Another work conducted by Sousa *et al.* (2009) proposed a traceability framework for implementing forward and backward trace links among different artefacts from SPL development, using model-driven engineering techniques. This framework requires the definition of a variability model to allow the tracing of SPL common and variable features along the domain and application engineering stages. The variability model is used in this approach as the main reference to trace the SPL artefacts. In addition, it is implemented as a flexible framework in order to allow its customization to different SPL traceability scenarios.

This work also delineates a survey of existing traceability tools development. This survey was conducted in the context of the AMPLE project (AMPLE, 2011). According to Sousa *et al.* (2009), the objectives of this survey were to investigate the current features provided by existing tools in order to assess their strengths and weaknesses and their suitability to address SPL development. The tools were evaluated in terms of the following criteria: management of traceability links; traceability queries; traceability views; extensibility; and support for Software Product Lines, Model Driven Engineering and Aspect-Oriented Software Development (AOSD).

In this context, Sousa *et al.* (2009) concluded that none of the investigated tools had built-in support for SPL development, and a vast majority of them are closed, so they cannot be adapted to deal with the issues raised by SPL. In additional, this work also identifies that many tools are still in an experimental state and does not cover traceability up to a degree that is desirable. Another relevat aspect detected in this survey is that many tools do not support advanced and specific support to deal with change impact analysis or requirement/feature covering in the context of SPL development. This is a specific gap that will be covered in our approach in Chapter 4.

Finally, the conclusions draws from this survey were that existing traceability tools do not provide a sucient support to address the traceability problem in SPL development.

In the context of automatic generation of traceability relations, Jirapanthong and Zisman (2009) define a traceability reference model with nine different types of traceability relations for eight types of documents. The traceability rules used in this work are classified into two groups namely: *direct rules*, which support the creation of traceability relations that do not depend on the existence of other relations, and *indirect rules*, which require the existence of previously generated relations. This rule-based approach allows automatic generation of traceability relations between elements of documents created during the development of product line systems.

In general, rules assist and automate decision making, allow for standard ways of representing knowledge that can be used to infer data, facilitate the construction of traceability generators for large data sets, and support representation of dependencies between elements in the documents. In addition, the use of rules in this approach allows for the generation of new relations based on the existence of other relations, supports the heterogeneity of documents being compared, and supports data inference in similar applications (Jirapanthong and Zisman, 2009).

This work is an extension of the work of (Jirapanthong and Zisman, 2005). It presented some initial ideas about the approach, an initial version of the traceability reference model, and few examples of traceability rules.

A prototype tool called XTraQue was implemented. This tool allows the generation of traceability relations by interpreting traceability rules. It also offers support for creating new traceability rules and translating documents into XML format.

In order to illustrate and evaluate this work, the researchers used a case study from a mobile phone product line system. According to Jirapanthong and Zisman (2009), the results of these experiments are encouraging and comparable with other approaches that support automatic generation of traceability relations. The results of these experiments have shown an average precision of 85.3% and an average recall of 83.5%. These results are comparable to other approaches that support automatic generation of traceability relations between requirements specifications and source code, requirements, use cases, and object models, and requirements and design models (Jirapanthong and Zisman, 2009).

In the context of Information Retrieval - IR approaches, Lucia *et al.* (2007) have improved an artefact management system with a traceability recovery tool based on Latent Semantic Indexing (LSI), an information retrieval technique. Although several research and commercial tools are available to support traceability between artefacts, the main drawback of these tools is the lack of automatic or semi-automatic traceability link generation and maintenance (Alexander, 2002). According to Lucia *et al.* (2007), several researches have recently applied Information Retrieval techniques to the problem of recovering traceability links between artefacts of different types. IR-based methods recover traceability links on the basis of the similarity between the text contained in the software artefacts. The rationale behind them is the fact that most of the software documentation is text based or contains textual descriptions and that programmers use meaninful domain terms to define source code identifier.

This work has assessed LSI in relation to strengths and limitations of using information retrieval techniques for traceability recovery and devised the need for an incremental approach. LSI is an advanced IR method that is able to achieve the same performances without as the classical probabilistic or VSM without requiring preliminary morphological analysis of the document words.

Another contribution of this work are the definition and implementation of a tool that helps the software engineer to discover emerging traceability links during the software evolution, as well as to monitor previously traced links.

Finally, Lucia *et al.* (2007) concluded that although tools based on information retrieval provide a useful support for the identification of traceability links during software development, they are still far to support a complete semi-automatic recovery of all links. The results they experience have also shown that such tools can help to identify quality problems in the textual description of traced artefacts.

Following are outlined the main risks and challenges of the traceability adoption in the context of Software Product Lines Engineering.

3.2.3 Risks and Challenges

The large number and heterogeneity of documents generated is considerated as the biggest challenge to the adoption and maintenance of traceability in the development of product lines. Systems may cause difficulties to identify common and variable aspects among applications, and to reuse core assets that are available under the product line (Jirapanthong and Zisman, 2009). Moreover, according to Kim *et al.* (2005), a precise mapping and relationship among the artefacts are yet to define. In order to apply PLE practically and to automate PLE process, elements of artefacts and their relationships should be precisely specified.

In general, most of authors agree that despite the advantages and benefits of current traceability methods and techniques, most of them do not provide automatic mechanisms or tools to address the traceability between the produced artefacts in both domain and application engineering process. Sousa *et al.* (2009) argue that this is fundamental to guarantee and validate the quality of SPLs development and to allow a better management of SPL variabilities. On the other hand, current traceability tools do not provide support to address the new artefacts, such as variability model or processes of SPL development.

As a result, they do not allow the explicit modeling and management of SPL features. For Lucia *et al.* (2007), the support for traceability in contemporary software engineering environments and tools is not satisfatory. Lucia *et al.* (2007) also complete that this inadequate traceability is one of the main factor that contributes to project overruns and failures. According to (Alexander, 2002), the main drawback of these tools is the lack of automatic or semi-automatic traceability links generation and maintenance.

Beside all these topics discussed previously, the traceability area in the context of SPL has some limitation, because the majority of the approaches concerning traceability for product line systems focus on traceability metamodel and do not provide ways of generating traceability relations automatically. This is one of the main concerns of this work.

Finally, Moon *et al.* (2007) show that traceability issues are even more important in the context of product line because the impact of changes in a product line can involve all the product line-based applications, thus the impact analysis is considered as another challenge in this area. The main topics of Impact Analysis are outlined in next session.

3.3 Impact Analysis

Since the process proposed by Royce (1987), popularly known as waterfall process, the maintenance was considered a post-production activity. According to the waterfall process, maintenance is the last phase of the software life-cycle, after the delivery and deployment of the software. This classical view on maintenance has governed the industrial practice in software development and is still in use today by several companies (Mens and Demeyer, 2008).

As analyzed by Oliveira (2009), it took a while before the software engineers have realized the limitations of this waterfall model, given the fact that the separation in phases were too strict and inflexible. Commonly, it is unrealistic to assume that all requirements are know before starting the design phase of the software. One example is that in many cases, the software requirements continues to change until the end of the software life-cycle.

Software process more evolutionary have been proposed, such as the change mini-cycle, which introduced new important activities such as change *impact analysis* and change propagation (Yau *et al.*, 1993).

According to Abma (2009), when a system is designed with evolvability in mind, a good change process can improve the implementation of changes. Such process is called change management and provides a common communication channel between maintenance staff, users, project managers and operations staff, and it provides a directory of changes to the system, for status reporting, project management, auditing and quality control.

According to Moreton (1996), change management needs at least the following steps: *impact analysis*, system release planning, change design, implementation, testing and system release and integration. One of the interests of this work are in the first stage: *impact analysis*. In general, *impact analysis* is the activity of identifing what to modify to accomplish a change, or of identifying the potential consequences of a change. For Arnold and Bohner (1996), software change impact analysis is the process of identifying the potential consequences of a proposed change. It can be used to estimate what has to be modified in an implementation to accomplish a change. Another definition is providen by Pfleeger (2001), as the evaluation of the many risks associated with the change, including estimates of effects on resources, effort, and schedule.

Arnold and Bohner (1996) summarize that an important aspect that all definitions have in common is that *impact analysis* normally does not change anything but for their understanding of the system and the effect of the proposed change. Arnold and Bohner (1996) still identified two types of *impact analysis* being supported in tools. The first type works based on analyzing the source code using techniques like program slicing, cross-referencing and control flow analyzing. The second type is more life-cycle-document oriented and is based on creating relations between artefacts from the beginning of the project, instead of the end.

According to Lientz and Swanson (1980), software evolution can be categorized by the type of changes that are being peformed into four different ”dimensions”:

- **Adaptive:** related to maintenance performed in response to changes in data and processing environments;
- **Perfective:** maintenance performed to eliminate processing inefficiencies, enhance performance, improve maintainability, or which is response to new user requirements;
- **Corrective:** maintenance performed in response to failures; and
- **Preventive:** where the objective is to prevent problems in the future.

The adaptive, perfective and corrective activities are reactive evolution management activities, since it concerns with managing the arrived changes and controlling them in order to maintain the stability and integrity of the systems. On the other hand, preventive activities can be seen as a proactive evolution management activity, since the efforts are on improving the evolvability, and preventing future problems.

3.3.1 Impact Analysis for SPL

In the context of Software Product Lines, the evolution mechanisms still appling in general. However, acoording to Galvão *et al.* (2008), Software Product Lines pose additional challenges regarding their evolution. The evolution in product line is complicated by the fact that an asset is shared among products, and any change in this asset may affect on several products (McGregor, 2003). Thus, this makes evolution management more challenging than in traditional single software development (Pussinen, 2002).

A recent work developed in the RiSE Labs by Oliveira (2009) presented the RiPLE-EM process to evolution management. This process is a systematic way to guide and manage the evolution of every asset and product in a product line context, handling change management, build management and release management activities.

Following are described the main *Impact Analysis* Challenges in the context of Software Product Lines.

Challenges

The main challenges identified by Galvão *et al.* (2008) are described in order to preserve the benefits of the Software Product Lines approach:

- **Declining quality:** As with single systems development, the product line is subject to decline quality unless measures are taken to preserve the quality of the product line. Therefor changes to software artefacts need to be kept track of avoid the deviation of core assets from the general architecture, or even disconnection between assets and products.

- **Corrective maintenance:** Differently than single systems development, corrective maintenance on SPL may impact several application products. We can discern between maintenance to core assets and custom assets, where the maintenance on core assets potentially impacts a higher number of products, while maintenance to custom assets only affects a smaller number. In any case, corrective changes need to be evaluated carefully in order to ascertain the applicability of the performed changes to all involved products (Galvão *et al.*, 2008).

3.4 Chapter Summary

This Chapter presented the main concepts about the Traceability area, including traceability in the context of Software Product Lines and Impact Analysis.

Traceability refers to the ability to link different information, therefore, traceability of software artefacts has been recognized as an important approach for supporting various activities in the software system development process.

Existing approaches and tools for traceability support the representation of different types of relations (dependency, generalization, evolution, satisfiability, overlap, contribution, conflit and rationalisation) between system artefacts but the interpretation of the semantics associated with a traceability relation depends on the stakeholders, as described in the Section 3.1.1. Traceability provides stakeholders with a means of showing compliance with requirements, maintaining system design rationale, showing when the system is complete, and establishing change control and maintenance mechanisms. The different options of traceability generation (manual, semi-automatic and automatic) described in the literature also were described in Section 3.1.3.

In the context of Software Product Lines Engineering, software artefact traceabily is an important factor when it comes to effective development and maintenance of software system. Traceability management facilitates the SPL artefacts to remain in synchronous state and ensure the consistency of derived products (Abid, 2004).

A preliminary Mapping Study (Petersen *et al.*, 2007) on Traceability for SPL was performed and several related and important works were selected.

Finally, software change *impact analysis* was discussed in the context of Software Product Lines. Impact Analysis is the process of identifying the potential consequences of a proposed change (Arnold and Bohner, 1996). Impact analysis in the context of SPL is complicated by the fact that an asset is shared among products, and any change in this asset may affect on several products.

Next Chapter presents in details the proposed approach to traceability recommendation in the context of Software Product Lines. Still in the following Chapter are outlined the set of requirements, functionalities and architecture of the TIRT.

4 TIRT: Traceability Information Retrieval Tool

Whether you think you can or can't, you're right.

—HENRY FORD (Engineer)

Traceability of software artefacts has been recognized as an important approach for supporting various activities in the software development process. Potential benefits of traceability include better impact analysis, lower maintenance costs, and better assessment of product quality (Bennett and Rajlich, 2000). Besides these benefits, traceability also enables system acceptance by allowing users to better understand the system and contributes to a clear and consistent system documentation. In this way, traceability improves the quality of software system (Asuncion, 2008).

However, as discussed in the previous Chapter, the adoption and maintenance of traceability in the context of product lines is considered a difficult task, due to the large number and heterogeneity of assets developed during product lines engineering. Futhermore, the manual creation and management of traceability relations is difficult, error-prone, time consuming and complex.

In this Chapter, a metamodel is described, which was developed by the RiSE Labs which proposes a representation of traceability information, including the different traceable elements (Features, Requirements, Use Case and Test). This metamodel is the basis for the traceability recommendation proposal and the tool called TIRT (Traceability Information Retrieval Tool).

TIRT goal is to facilitate the creation and mainly the maintenance activities regarding to the traceability relationship among the different core assets in a Software Product Line through its recommendation system. In addition, the tool was also built in order to help in the process of impact analysis. Thus, the time spent in these activities can be reduced and less error prone.

This Chapter is organized as follows: Section 4.1 presents set of Functional and Non-Functional requirements proposed for the tool and Section 4.2 presents the the proposal for traceability recommendation. Section 4.3 describes the architecture, the set of frameworks and technologies used during the TIRT implementation, while Section 4.4 shows the operation of the tool and, finally, Section 4.5 summarizes this Chapter.

4.1 The Set of Requirements

According to Kotonya and Sommerville (1998), the requirements define the services that the system should provide and the set out constraints on the system's operation.

4.1.1 Functional Requirements

Functional Requirements (FR) define the functions of a system, which can be, for example, manipulation of data, implementation of data, implementation of algorithms, and the technical details on the system implementation (Kotonya and Sommerville, 1998). In the TIRT specification, the following Functional Requirements were defined:

- **FR1. Keyword-based Search in Core Assets base:** The tool must provide features for core assets search in the database, as in web search engines. This was used since that people are familiar with web search engines.

- **FR2. Core Asset Traceability Recomendation:** After the inclusion or update of some core asset, the system has to provide possible recommendation of the core assets traceability, showing the link between them. The recommendations are accomplished based on the recommendation scenarios defined in Section 4.2.3. This semi-automatic approach aims to mitigate the inconsistencies of traceability and maintainability problems, since every update realized to a core asset, some possible suggestions of traceability are proposed.

- **FR3. Extract useful information from the traceability's recommendations:** After making a search in the tool of possible core assets recommendations, the submitters can view each possible core assets from the recommendation list. Thus, when a core asset is being view, the tool must extract relevant information from it, such as *descriptions*, *related core assets*, restrictions, and so on; Such supplementary information may be useful in the analysis made by the domain analyst to ascertain whether the relationship among these artefacts is consistent.

- **FR4. Impact Analysis:** When a core asset is updated, its traceability matrix is shown with the purpose of identifying the impact analysis (forward and backward) of the evolution or correction, in the relation to the SPL assets. With this feature, the domain analyst can analyze the impacts that could occur, before the update be performed.

- **FR5. Metamodel Implementation:** The TIRT tool should implement the entities and relationships defined in the metamodel described in Section 4.2.1. The metamodel is in accordance with the proposal of this work, since this metamodel can easily understand the relationships among the SPL assets, communicate to the stakeholders, facilitate the evolution and maintenance of the SPL.

- **FR6. Core Asset Addition:** The tool should allow the user to add new core assets (e.g. *features*, *requirements*, and *use cases*), according to metamodel.

- **FR7. Rank search results based on core assets similarity rate:** The search result must be ranked according to the similarity of textual description of core assets according to recommendations scenarios on Section 4.2.3.

- **FR8. Core Assets Status Report:** Project managers have interest in reports such as: relatioship between core assets, list of *features*, *requirements*, or results based on core assets similarity rate.

- **FR8. Core Assets Status Report:** Project managers have interest in reports such as: relationship between core assets, list of *features*, *use cases*. All these information can be helpfull to the management of the SPL, since the manager can use this information to inspect their product line.

4.1.2 Non-Functional Requirements

For Kotonya and Sommerville (1998), Non-Functional Requirements (NFR) are requirements which are not specifically concerned with the functionality of a system. They place restrictions on the product being developed and the development process, and they specify constraints that the product must meet. Non-functional requirements include safety, security, usability, reliability, and peformance requirements, among others. In the TIRT specification, the following Non-Functional Requirements were defined:

- **NFR1. Web-based interface:** Engineers and Domain Analysts can be located remotely, thus the tool must run on a web-server which enables them to access the system independently of their location.

- **NFR2. Usability:** The traceability maintenance can be an exausting activity. Thus, the tool must optimize the usability experience, providing a aesthetic and minimalist design, according to the usability heuristic proposed by Nielsen (2011).

- **NFR3. Access Control:** Since the tool is web-based, it is necessary to keep the access control in the application in order to guarantee the core assets integrity.

- **NFR4. Log users actions:** In order to perform the evolution the tool, some indicators should be collected regarding to the tool usage. Thus, the tool must log all engineers actions. With the log it is possible to analyze these indicators in order to provide some customizations or corrections in the tool.

- **NFR5. Maintainability and Reusability:** The tool must be developed under the properties of maintainability and reusability. As mentioned in Chapter 1, TIRT project is part of the RiSE Labs. Thus, software reuse researchers (Krueger, 1992) advocate that the systems should be developed in form of buildable components, in order to provide good maintainability and reusability. The demoiselle framework outlined in Section 4.3.2 allows this kind of development.

- **NFR7. Help for Vocabulary and Scenario Recommendation:** The tool should provide a detailed explanation about the *vocabulary* characterization. Its *vocabulary* proposes some suggestions to the fields registration. Its aims the improvement of traceability recommendation among the core asset artefacts. This *vocabulary* characterization is outlined in Section 4.2.4. In additional, the system should also provide details regarding to the *recommendation scenarios* that is outlined in Section 4.2.3.

Next Section outlines the Proposal for Traceability Recommendation. These ideas presented in the following Section formed the basis for implementation and aplicability of the Functional and Non-Functional requirements described previously.

4.2 Traceability Recommendation Proposal

In order to apply PLE systematically and to automate its process, artefacts and their relationships need to be precisely specified (Kim *et al.*, 2005).

In this context, the RiSE Labs have been focused on the development of the Rise Product Line Engineering Process (RiPLE), an effort to build a framework for SPL development, that encompasses a set of disciplines that comprise SPL life-cycle, such as *risk management*, *scoping*, *requirements*, *design*, *testing*, *evolution* and *product derivation*. In addition, they have working in a metamodel that is the basis for maintaining the traceability among different assets.

4.2.1 The Metamodel

The Product Line success depends on some factors that should be considered, such as: mature software engineering, planning and reuse, adequate practices of management and development, as well as the ability to deal with organizational issues and architectural complexity. Thus, the development must be supported by auxiliary methods and tools which helps the development process (Birk and Heller, 2007).

In this context, a recent work (Cavalcanti *et al.*, 2011) presented a metamodel which aims to coordinate SPL activities, by managing different SPL phases and to maintain the traceability and variability among different artefacts. The metamodel was built for a SPL project in a private company working on the medical information management domain, which includes four products.

In this work, the metamodel is used as basis for the traceability recommendations described in Section 4.2.3 and development of the TIRT tool, outlined in Section 4.4. This metamodel serves as basis to understand the relationship among the SPL assets, communicate them to the stakeholders, and facilitate the evolution and maintenance of the SPL. Thus, the basis of constructing a SPL metamodel is a strong linkage among all elements/assets involved in a SPL development (Cavalcanti *et al.*, 2011).

The metamodel is composed of five sub-models: *scoping*, *requirements*, *testing*, *project* and *risk management*, each of them, representing a SPL phase (Cavalcanti *et al.*, 2011). It was described using UML notation (Booch *et al.*, 2005), according to Figure 4.1 described next. Figure 4.1 shows the complete view of the metamodel, where the dashed boxes specify the phase of the metamodel.

- **Scoping Module:** In the scoping module, the *Feature* objects are grouped into

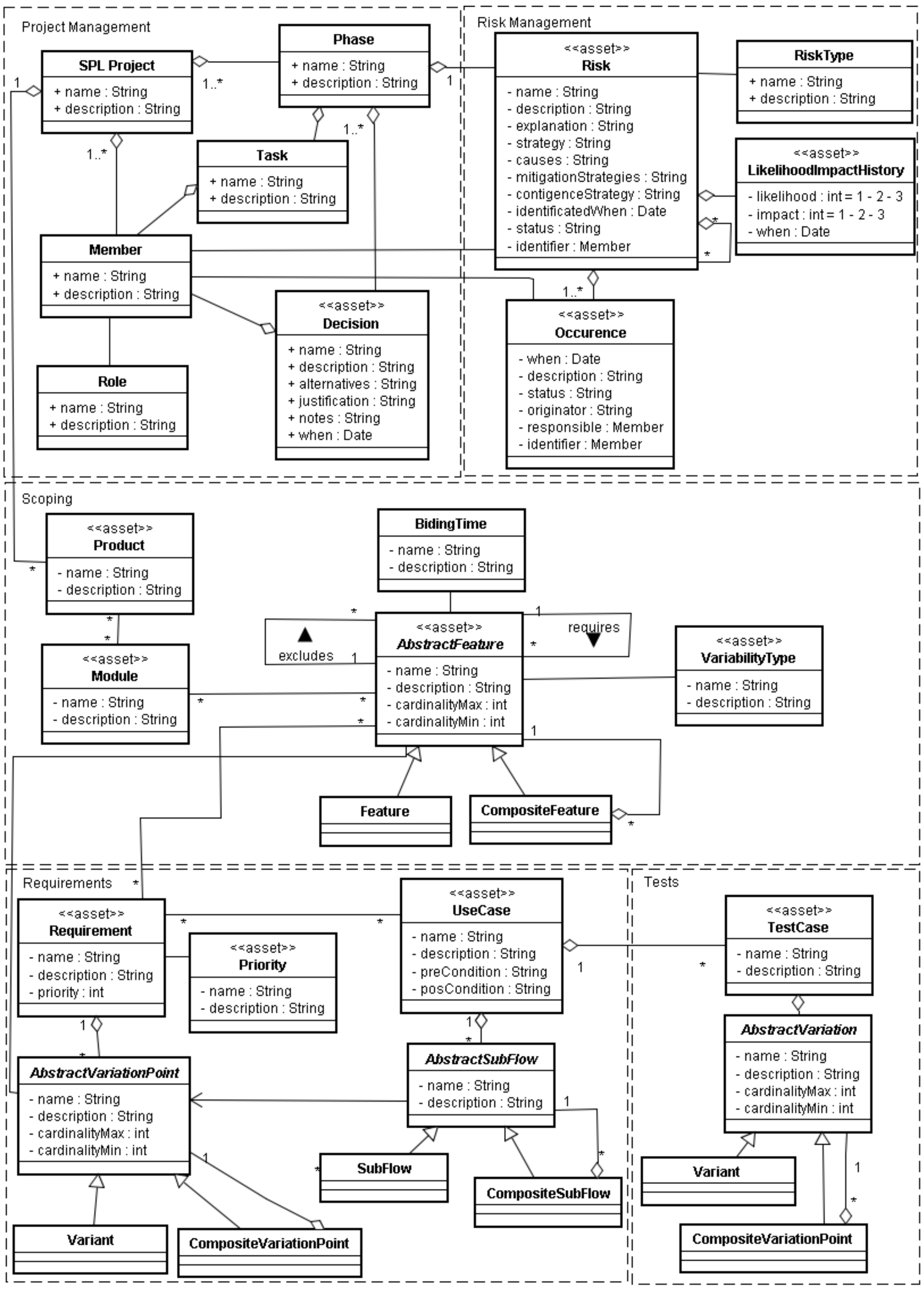


Figure 4.1 Software Product Line Metamodel (Cavalcanti *et al.*, 2011)

Module objects, which are then grouped into *Product* objects. The *Module* objects can be viewed also as the sub-domains of the *Product* objects, since it better represents the SPL project. During *scoping* phase, the feature model is assembled based on the method to identify those *features* assets (Kang *et al.*, 1990a). The tree is built containing all the *features types*, such as *mandatory*, *alternative* and *optional* features. In the scoping model, a *Feature* object also has *BindingTime* and *VariabilityType* associated with it. Kang *et al.* (1990a) describe examples of biding time, such as *before compilation*, *compile time*, *link time*, *load time*, *run-time*; and examples of variability as *Mandatory*, *Alternative*, *Optional*, and *Or*.

In addition, the feature model proposed in the scoping model also enables other relationship between features, such as: *required* and *excluded* features (Cavalcanti *et al.*, 2011). In this sence, the composite design pattern (Gamma *et al.*, 1995) is a good representation using UML diagrams for the feature model proposed by Kang *et al.* (1990a), hence it was used to dependencies to be represented in a form of tree, where features can represent the feature model. This pattern enables the features and their subfeatures recursively.

- **Requirements Module:** The two main assets from this SPL phase are the *requirements* and *use cases*. This phase also involves the requirement engineering traceability and interactions issues, considering the variability and commonality in the SPL products. The Requirement object is composed by *name*, *description*, *priority*. During its elicitation, it should envisage the variations over the foreseeable SPL life-cycle, considering the products requirements and the SPL variations (Cavalcanti *et al.*, 2011).

  Some scoping outputs serve as source of information in this phase. For instance, during the requirements elicitation, the feature model is one of the primary artefacts. The relationship between feature and requirements is many-to-many in both ways, which means that one feature can encompass many requirements and different requirements can conglomerate many features (Neiva, 2008).

  Cavalcanti *et al.* (2011) highlight three scenarios where a requirement may be described: (i) the requirement is a variation point; (ii) the requirement is a variant of a variation point; and (iii) the requirement has variation points. The same scenarios are used when eliciting use cases, it also can be composed of variation points and variants, described in the model by flow and sub-flow, respectively. In addition, the same many-to-many association is used between *requirements* and *use cases*, in which one *requirement* could encompass many *use cases*, and an *use case* could be related by many *requirements*. The Use Case model is composed of *name*, *description*, *preCondition* and *posCondition*. The alternative flows are represented by the *flows* and *sub-flows*.

- **Testing Module:** The metamodel developed also encompasses testing issues in terms of how system test cases interact with other artefacts. The *Test Cases* are derived from *Use Cases*, as can be seen in Figure 4.1. This model expands on the abstract use case definition, in which variability is represented in the use cases. According to Cavalcanti *et al.* (2011), a use case is herein composed of the entity *AbstractFlow*, that comprises the subentity Flow, which actually represent the use case steps. Every step can be associated with a subflow, which can represent a variation point. Considering that a use case can generate several test cases, the strategy to represent each step in a use case and enable it to be linked to a variation point, supporting the building of test cases which also consider the variation points. This way, several test cases can be instantiated from a use case, which describes the variation points. Variability is preserved in the core asset test artefacts to facilitate reuse.

  It allows that every change in the use case and/or variation points and their variants are properly propagated to the test cases and their steps (variable or not), due the strong traceability represented in the metamodel (Cavalcanti *et al.*, 2011).

- **Project Management Module:** The main objective of this module is to coordinate the SPL activities. It supports the SPL management by keeping track of SPL phases and the staff responsible for that. Thus, through this model it is possible to define information about the SPL project, as well as details, such as: the SPL phases, which are the tasks supported, the members responsible for these tasks and the roles of each one. In addition, the decisions about the SPL project can be documented using the *Decision* entity, by describing the *decisions*, *alternatives*, *justification* and *notes*, also registering their occurence and the involved staff (Cavalcanti *et al.*, 2011).

- **Risk Management Module:** According to Sommerville (2007), Risk Management (RM) is particularly important for software projects since of the uncertainties that most projects face. In this context, this RM involves activities that encompass all SPL phases. These activities performed by this metamodel are: *risk identification*, *documentation*, *analysis*, *planning* and *risk monitoring*. These activities are based on the definition proposed by Sommerville (2007), however it was adapted to this context, based on the needs with feedback in risk identification stage. (Cavalcanti *et al.*, 2011).

The risks detected according to these activities are identified and then their main characteristics are documented. The characteristics are: the *risk description*, *type*, *status*, *mitigation strategy*, and *contingency plans*. In addition, as the RM process is a continuous activity, it is necessary to keep track of the history about the management of these risks. Thus, the risks likelihood and impact are documented according to their occurrence, which happens in different moments throughout the project development (Cavalcanti *et al.*, 2011).

4.2.2 Metamodel Instantiation

The TIRT tool implements the traceability scenarios of recommendation proposed in the next Section. These scenarios are based on part of the metamodel outlined in the previous Section. Figure 4.2 describes the subset of the Software Product Lines Metamodel highlighting the *Project Management*, *Scoping* and *Requirements* modules, focus of this work.

As mentioned in Chapter 1, the creation of individual products by reusing the artefacts is an important issue in a SPL. However, this aspect can be as complex as core assets development, thus, this work is restricted to *features*, *requirements*, *use cases* and the existing traceability among them.

4.2.3 Scenarios Recommendation

Recommendation Scenarios are likely occurrences of updates, which map their impacts and correlations. The recommendation scenarios of traceability are described in order to facilitate the maintenance activities of SPL. This recommendation approach aids the decision making, supporting representation of dependencies between different core assets.

The *engineer* or *domain analyst* interact with the collection of core assets in order to examine existing similarity relationship after the updating of these artefacts. In order to verify whether a core asset is related, the engineer should perform searches in the system and interact with this artefact in order to verify the consistency of this relationship.

In this way, the main motivation of traceability recommendations of core assets is to support the involved stakeholder during the maintenance activities. The maintenance process are influenced by the different change dimensions (Lientz and Swanson, 1980)

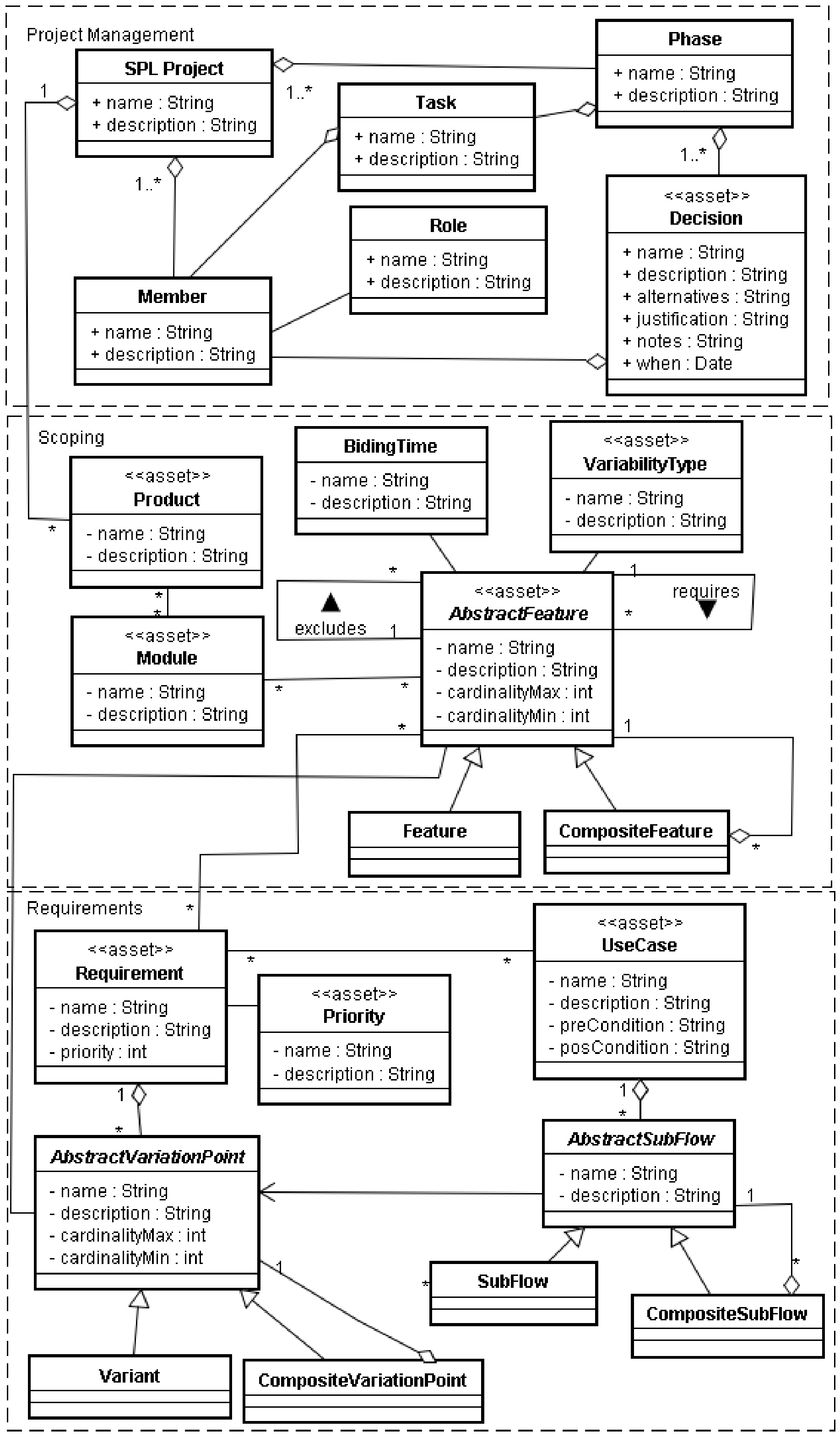


Figure 4.2 Subset of the Software Product Line Metamodel

which occurs during the software evolution, such as *adaptative*, *perfective*, *corrective* and *preventive*. According to Lientz and Swanson (1980), these dimensions of changes are intrinsic in the life-cycle of the system, because the adaptive and perfective changes are driven by the improvements or adaptations. On the other hand, corrective is peformed in response to changes in data and documents. Since the preventive changes are concerned to prevent problems in the future.

We identified and defined seven different scenarios in which the core assets can be modified. Figure 4.3 depict the three core assets (*Feature*, *Requirement* and *Use Case*) analyzed in the context of this work and the relationship among them. After that, all seven modification scenarios are detailed.

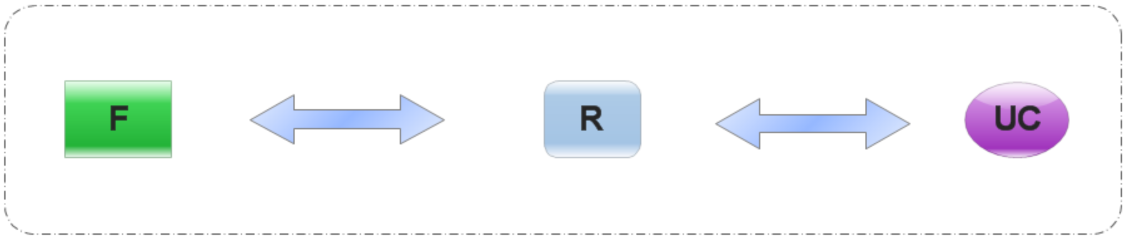


Figure 4.3 Core Assets: Features, Requirements and Use Cases

- **Scenario 1:** Given a reference scenario that represents all modifications between a *Feature* and *Parent-Feature* in Figure 4.4 which describes the relationship between these two core assets, where the *Feature* **F5** is initially related with the *Parent-Feature* **F2**. Imagine that a *Feature* **F5** should be modified to **F5'** in order to reflect a change, due to a evolution or corrective action in the SPL. In this moment, according to the recommendation system, another *Parent-Feature* is recommended to be related to *Feature* **F5'**. Considering these two versions, the original *Feature* **F5** is related to *Parent-Feature* **F2**, and the new one is related to *Parent-Feature* **F3** that the engineer considers as a relevant and consistent relationship.

- **Scenario 2:** Given a reference scenario with *Feature* and *Required-Feature* in Figure 4.5 that describes the relationship between these two core assets, where the *Feature* **F5** is initially related only to some *Required-Features* **F1** and **F2**. Imagine that a *Feature* **F5** should be modified to **F5'** in order to reflect a change, due to a evolution or corrective action in the SPL. In this moment, according to the recommendation system, a new *Related-Feature* is recommended to be related to *Feature* **F5'**. Considering these two versions, the original *Feature* **F5** related to

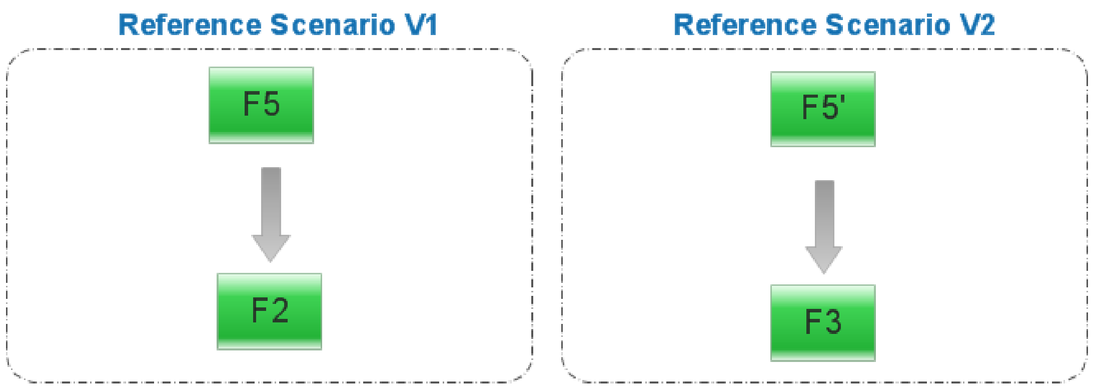


Figure 4.4 Scenario 1 - Feature and Parent-Feature

Related-Features **F1** and **F2**, and the new one is related to *Related-Features* **F1**, **F2** and **F3** that the engineer considers as a relevant and consistent relationship.

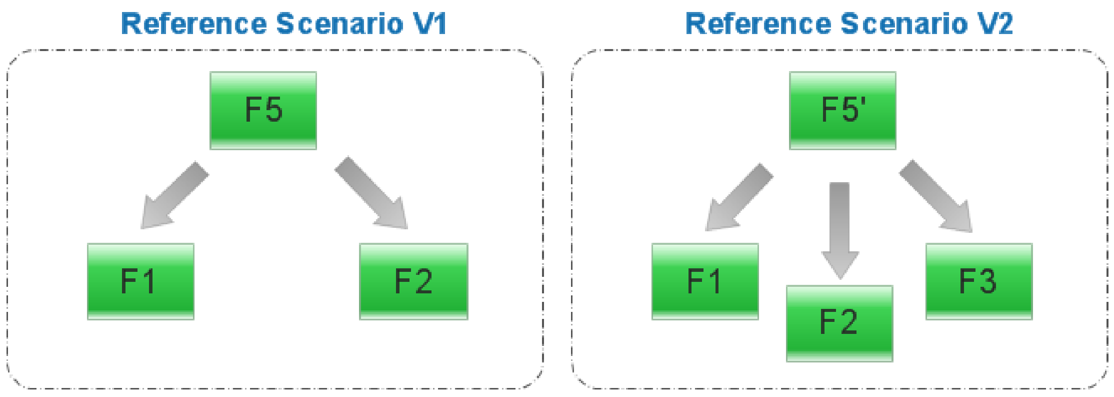


Figure 4.5 Scenario 2 - Feature and Required-Features

- **Scenario 3:** Given a reference scenario with *Feature* and *Excluded-Feature* in Figure 4.5 that describes the relationship between these two core assets, where the *Feature* **F5** is initially related only to some *Excluded-Features* **F1** and **F2**. Supose that a *Feature* **F5** should be modified to **F5'** in order to reflect a change, due to a evolution or corrective action in the SPL. In this moment, according to the recommendation system, a new *Excluded-Feature* is recommended to be related to *Feature* **F5'**. Considering these two versions, the original *Feature* **F5** related to *Excluded-Features* **F1** and **F2**, and the new one is related to *Excluded-Features* **F1**, **F2** and **F3** that the engineer considers as a relevant and consistent relationship.

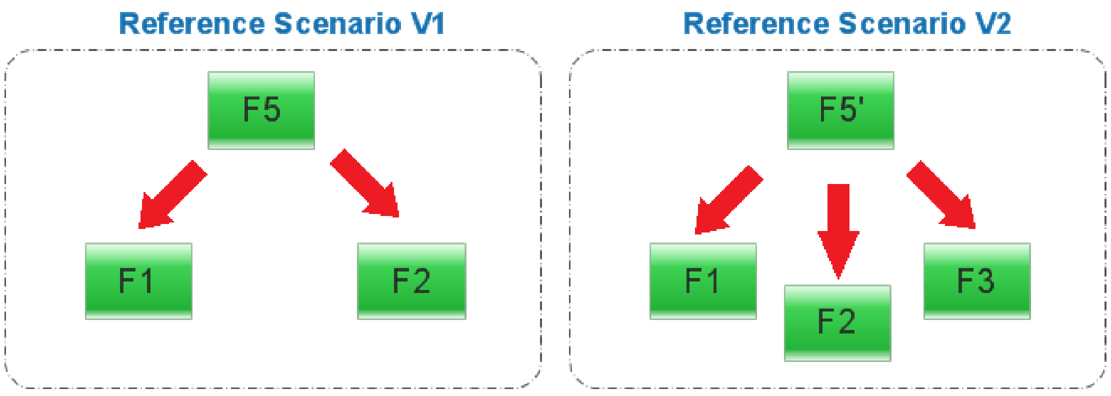


Figure 4.6 Scenario 3 - Feature and Excluded-Features

- **Scenario 4:** Given a reference scenario with *Feature* and *Requirement* in Figure 4.7 that describes the relationship between these two core assets, where the *Feature* **F1** is initially related only to some *Requirements* **R1** and **R2**. Imagine that a *Feature* **F1** should be modified to **F1'** in order to reflect a change, due to a evolution or corrective action in the SPL. In this moment, according to the recommendation system, a new *Requirement* is recommended to be related to *Feature* **F1'**. Considering these two versions, the original *Feature* **F1** related to *Requirements* **R1** and **R2**, and the new one is related to *Requirements* **R1**, **R2** and **R3** that the engineer considers as a relevant and consistent relationship.

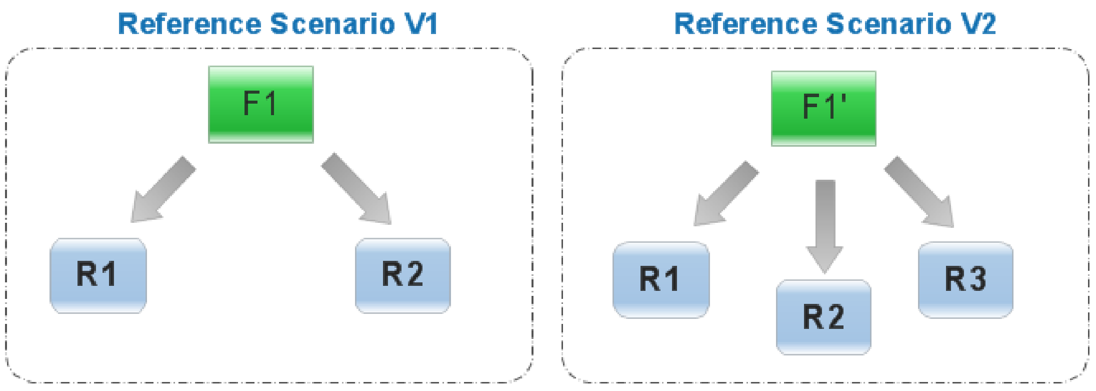


Figure 4.7 Scenario 4 - Feature and Requirements

- **Scenario 5:** Given a reference scenario with *Requirement* and *Feature* in Figure 4.8 that describes the relationship between these two core assets, where the

Requirement **R1** is initially related only to some *Features* **F1** and **F2**. Imagine that a *Requirement* **R1** should be modified to **R1'** in order to reflect a change, due to a evolution or corrective action in the SPL. In this moment, according to the recommendation system, a new *Feature* is recommended to be related to *Requirement* **R1'**. Considering these two versions, the original *Requirement* **R1** related to *Features* **F1** and **F2**, and the new one is related to *Features* **F1**, **F2** and **F3** that the engineer considers as a relevant and consistent relationship.

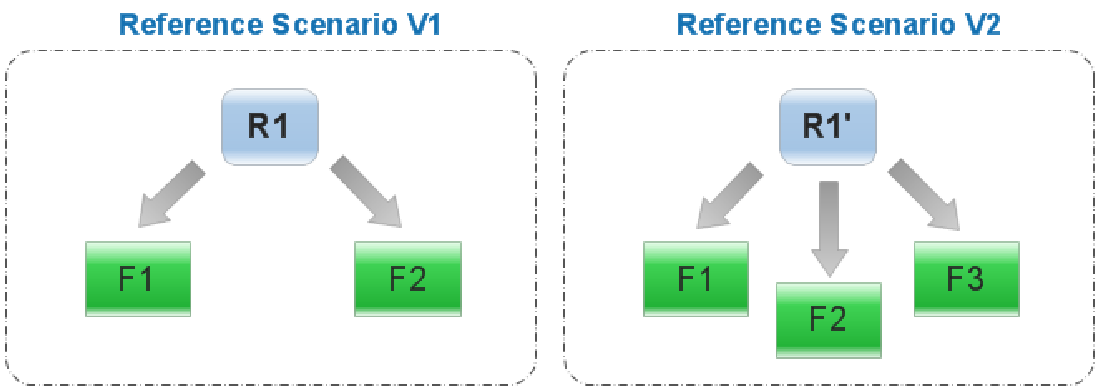


Figure 4.8 Scenario 5 - Requirement and Features

- **Scenario 6:** Given a reference scenario with *Requirement* and *Use Case* in Figure 4.9 that describes the relationship between these two core assets, where the *Requirement* **R1** is initially related only to some *Use Cases* **UC1** and **UC2**. Imagine that a *Requirement* **R1** should be modified to **R1'** in order to reflect a change, due to a evolution or corrective action in the SPL. In this moment, according to the recommendation system, a new *Use Case* is recommended to be related to *Requirement* **R1'**. Considering these two versions, the original *Requirement* **R1** related to *Use Cases* **UC1** and **UC2**, and the new one is related to *Use Cases* **UC1**, **UC2** and **UC3** that the engineer considers as a relevant and consistent relationship.

- **Scenario 7:** Given a reference scenario with *Use Case* and *Requirement* in Figure 4.10 that describes the relationship between these two core assets, where the *Use Case* **UC1** is initially related only to some *Requirements* **R1** and **R2**. Imagine that a *Use Case* **UC1** should be modified to **UC1'** in order to reflect a change, due to a evolution or corrective action in the SPL. In this moment, according to the recommendation system, a new *Requirement* is recommended to be related to *Use*

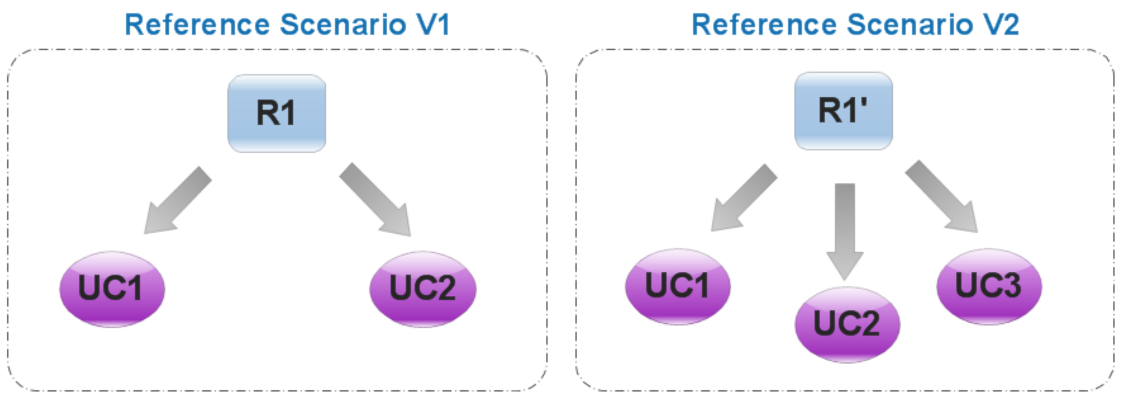


Figure 4.9 Scenario 6 - Requirement and Use Cases

Case **UC1'**. Considering these two versions, the original *Use Case* **UC1** related to *Requirements* **R1** and **R2**, and the new one is related to *Requirements* **R1**, **R2** and **R3** that the engineer considers as a relevant and consistent relationship.

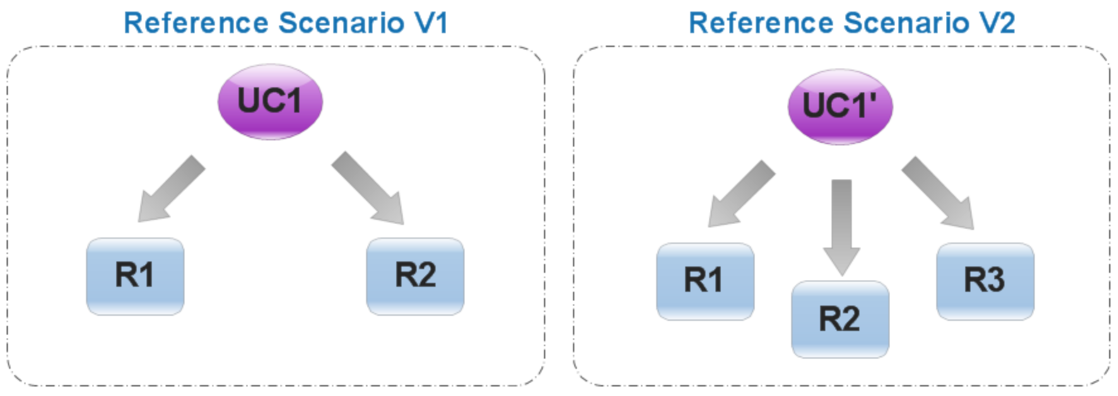


Figure 4.10 Scenario 7 - Use Case and Requirements

In the following Section, are mentioned the vocabulary adopted in order to obtain a better and consistent recommendation of core assets according to the scenarios depicted previously.

4.2.4 Vocabulary Standardization

Some experiments performed by Lucia *et al.* (2007) showed that the quality of textual description of the fields registred in the systems influency in the process and quality of information extraction. According to Lancaster (1986), a common and well defined vocabulary for registration the fields could help to finding the related core assets in the recommendation process.

In this context, we propose some suggestions to the fields registration. It aims the improvement of traceability recommendation between the core asset artefacts. The TIRT tool helps the software engineer to discover potential core assets relations, according to the descriptions expressed in natural language. However, these core assets might apply some suggestions of registration described following, aiming to minimize quality problem, in terms of poor textual description.

A recent work developed in the RiSE Labs by Neiva (2008) defined a requirement engineering process by providing activities for correct requirements development and management for Software Product Lines. Basically, this process called RiPLE-RE consists of a guideline to implement three activities: *Model Scope*, *Define Requirements* and *Define Use Case*. Although the process details some guidelines for requirements engineering, some specific aspects of the registration form are left out.

In face the problem of the different forms of core assets specification, we were encouraged to propose some suggestions of core assets registration, because the real language are not perfectly punctuated, and this problem influence negatively in the recommendation process. There are many difficult problems to consider in the context of core assets vocabulary, such as: *abbreviations*, *multi-component words*, *token-boundaries*, *contractions* and *sence-initial*. According to Kaplan (2005), many applications that work with text-processing ignore the treatment of these kind of limitations and difficulties. The text-processing is one of the main functionalities of the TIRT tool, as will be discussed in Section 4.3.

Most core assets express the text in natural language, based on textual description, written in narrative manner as the *name* and *description* properties. Based on some performed analyses in the tokenization context (Kaplan, 2005), we detailed and proposed some suggestions for registration these core assets. There is not a grammatical formalism that are closed under composition with regular relation, but instead, we propose some desirable properties, such as:

- **Abreviation:** The english string *chap.* can be taken as either an abbreviation for the word Chapter or as the word chap appearing at the end of the sentence, and Jan. can be regarded neither as an abbreviation for January or as a sentence final proper name. Thus, we recommend avoiding the registration of abbreviations.

- **Multi-Component Words:** One of the token boundaries used in the TIRT tool is the white-space, but there are some multi-component words in english that include white spaces as internal characters, e.g., *General Motors, a priori*. Thus, we do not recommend the hyphenation between these words as an alternative to represent these multi-component words.

- **Token-boundaries:** Another difficult in the context of text-processing is related to the token-boundaries, such as: *quotes* and *commas*. Thus, we do not recommend the use of *quotes* or *commas* in the core assets registrations.

Finally, the name of an use case have a particular formalization. We recommend that this name start with a verb. According to Larman (2004), a common exception to one use case per goal is to collapse CRUD (create, retrieval, update, delete) separate goals into one CRUD use case, idiomatically called *Manage <X>*. For example, the goals *edit user*, *delete user*, and so forth are all satisfied by the Manage Users use case.

In general, the punctuation is one of the main problem in the context of text-processing, but with these patterns described previously, we can mitigate these problems.

4.2.5 Impact Analysis

The TIRT tool supports the *impact analysis* approach, providing a traceability matrix to trace the impact analysis of some changes in the context of SPL. Moreover, the tool proposes a way of alerting and presenting to user the impact that could occur in the core asset if its was updated.

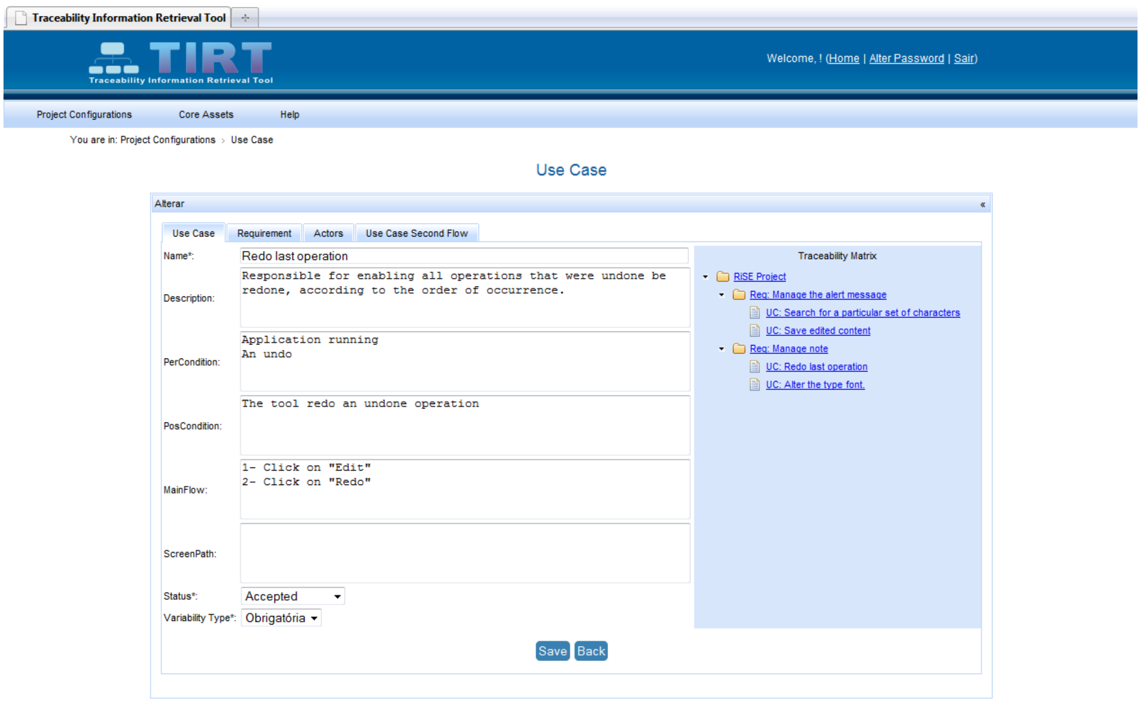
Arnold and Bohner (1996) summarize that an important aspect that all definitions have in common is that *impact analysis* normally does not change anything but for their understanding of the system and the effect of the proposed change. Arnold and Bohner (1996) still identified two types of *impact analysis* being supported in tools. The first type works based on analyzing the source code using techniques like *program slicing*, *cross-referencing* and *control flow analyzing*. The second type, that is the focus of this work is more life-cycle-document oriented and is based on, creating relations between artefacts from the beginning of the project, instead of the end.

The graphical representations details the traces and their direction between two core asserts. Therefore, with this representation it is possible to find the forward and backward traces between them. When an artefact in the SPL has been changed, this matrix view also shows which target artefacts can be affected by this change. In this context, the TIRT tool provides a way to perform an *impact analysis* to determine the affected artefacts. The matrix view is represented as a *tree view component*, or formally called in the literature

(Diehl, 2007) as the *indented component*, organized as a horizontal tree, that is presented before the update of some core asset, ensuring the engineer's analysis of the correlated core assets and the analysis of the change impact that could accur with these changes. In the horizontal tree, vertical lines show the extent of clocks, and vertically stretched oval lines show that of loops. This kind of structure between artefacts has to create a structure among artefacts to make clear which artefacts are predecessors or successors of an artefact.

Accordingly to Abma (2009), normally a change affects artefacts in more the two phases. Therefore, the system, to detect mappings that have to be changed or preserved between two phases, has to be extended so it taked in account the mappings in and between multiple phases.

The details view for a particular artefact can be used to see which other artefacts are directly linked to this one. By doing this again for all these linked artefacts and their links it is possible to analyze the impact (Abma, 2009). Figure 4.11 presents an sample of the *indented component* that represent the traceability matrix. In this picture, the use case artefact willbe updated, thus its relationships forward and backward are detailed in order to enable the analyze of the software engineer.


Traceability Information Retrieval Tool
TIRT
Traceability Information Retrieval Tool
Welcome, ! (Home | Alter Password | Sair)
Project Configurations
Core Assets
Help
You are in: Project Configurations › Use Case
Use Case
Alterar
Use Case
Requirement
Actors
Use Case Second Flow
Save
Back

Name*: Redo last operation

Description: Responsible for enabling all operations that were undone be redone, according to the order of occurrence.

PerCondition: Application running
An undo

PosCondition: The tool redo an undone operation

MainFlow: 1- Click on "Edit"
2- Click on "Redo"

ScreenPath:

Status*: Accepted

Variability Type*: Obrigatória

Traceability Matrix
- RISE Project
  - Req: Manage the alert message
    - UC: Search for a particular set of characters
    - UC: Save edited content
  - Req: Manage note
    - UC: Redo last operation
    - UC: Alter the type font.

Figure 4.11 Tree View of Analysis Impact

4.3 TIRT Architecture and Technologies

As mentioned in the beginning of this Chapter, TIRT goal is to provide a system for core asset registrations and recommendation of possible traceability relationship among them in the Software Product Lines context, thus, the time spent in these activities can be reduced, less error prone and less exhausted. To achieve this goal, the tool meets all the requirements previously defined.

Next section presents the architecture, the set of frameworks and technologies used in TIRT project.

4.3.1 TIRT's Architecture Overview

The recommendation search for core assets is one of the main and most important services that the TIRT tool provides for the software engineer. It is possible through the extraction and mining of core assets candidates. Thus, this type of activities follows the way a generic application of Text Mining.

Feldman and Sanger (2007) define Text Mining as a knowledge-intensive process in which a user interacts with a document collection over time by using a suite of analysis tools. In a manner analogous to data mining, text mining seeks to extract useful information from data sources through the identification and exploration of interesting patterns. In the case of text mining, however, the data sources are document collections, and interesting patterns are found not among formalized database records but in the unstructured textual data in the documents in these collections.

In the context of the TIRT tool, the data sources are the core assets and the user is the domain analyst, software engineer or another stakeholder responsible for maintenance of the core asset traceability. The software interacts with the collection of these core assets in the maintenance process, where one artefact should be modified in order to reflect a change, due to a evolution or corrective action in the SPL. Thus, in the moment of the update, a core asset relation is indicated according the traceability scenarios defined in Section 4.2.3.

Accordingly to Figure 4.12, some classic data mining applications are roughly divisible into four main areas: *(a) preprocessing tasks*, *(b) core mining operations*, *(c) presentation layer components and browsing functionality*, and *(d) refinement techniques* (Feldman and Sanger, 2007). However, the TIRT does not implement all areas of a generic architecture of text mining. Some modifications were performed in order to support the TIRT domain. As can be observed in schematic Figure 4.13, we removed from the original architecture described in Figure 4.12 the module *Text Mining Discovery Algorithms*. We also removed the *Document Fetching/Crawling* module, since the tool peforms the searchs from a single base. The *core assets* are our unique source, so we reduced the scope of the type of documents. In summary, this architecture was refined to cover only the issues to search based on by keywords, indexing, information extraction and visualization of possible recommended core assets.

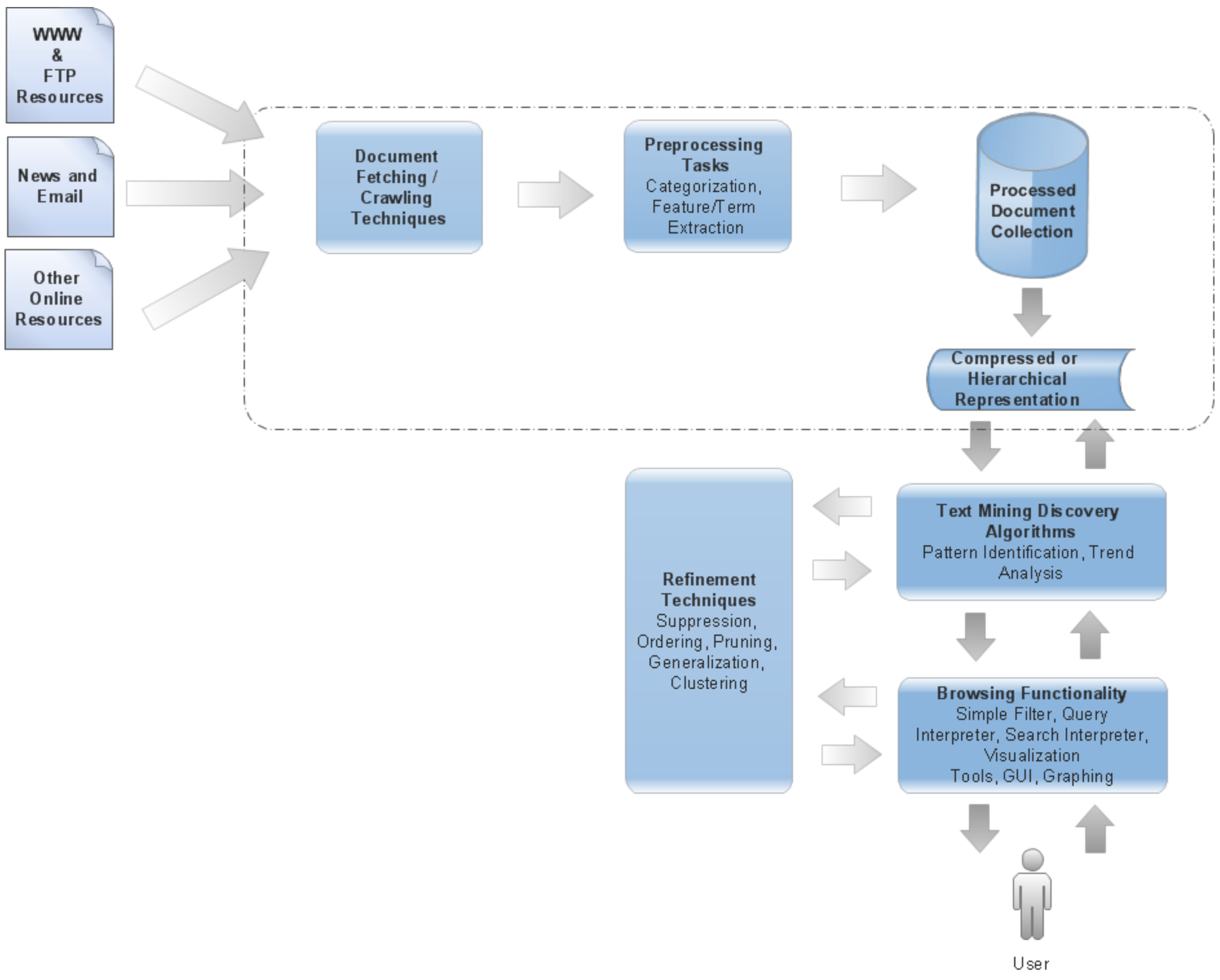

WWW & FTP Resources
News and Email
Other Online Resources
Document Fetching / Crawling Techniques
Preprocessing Tasks Categorization, Feature/Term Extraction
Processed Document Collection
Compressed or Hierarchical Representation
Text Mining Discovery Algorithms Pattern Identification, Trend Analysis
Refinement Techniques Suppression, Ordering, Pruning, Generalization, Clustering
Browsing Functionality Simple Filter, Query Interpreter, Search Interpreter, Visualization Tools, GUI, Graphing
User

Figure 4.12 System Architecture for Generic Text Mining System (Feldman and Sanger, 2007)

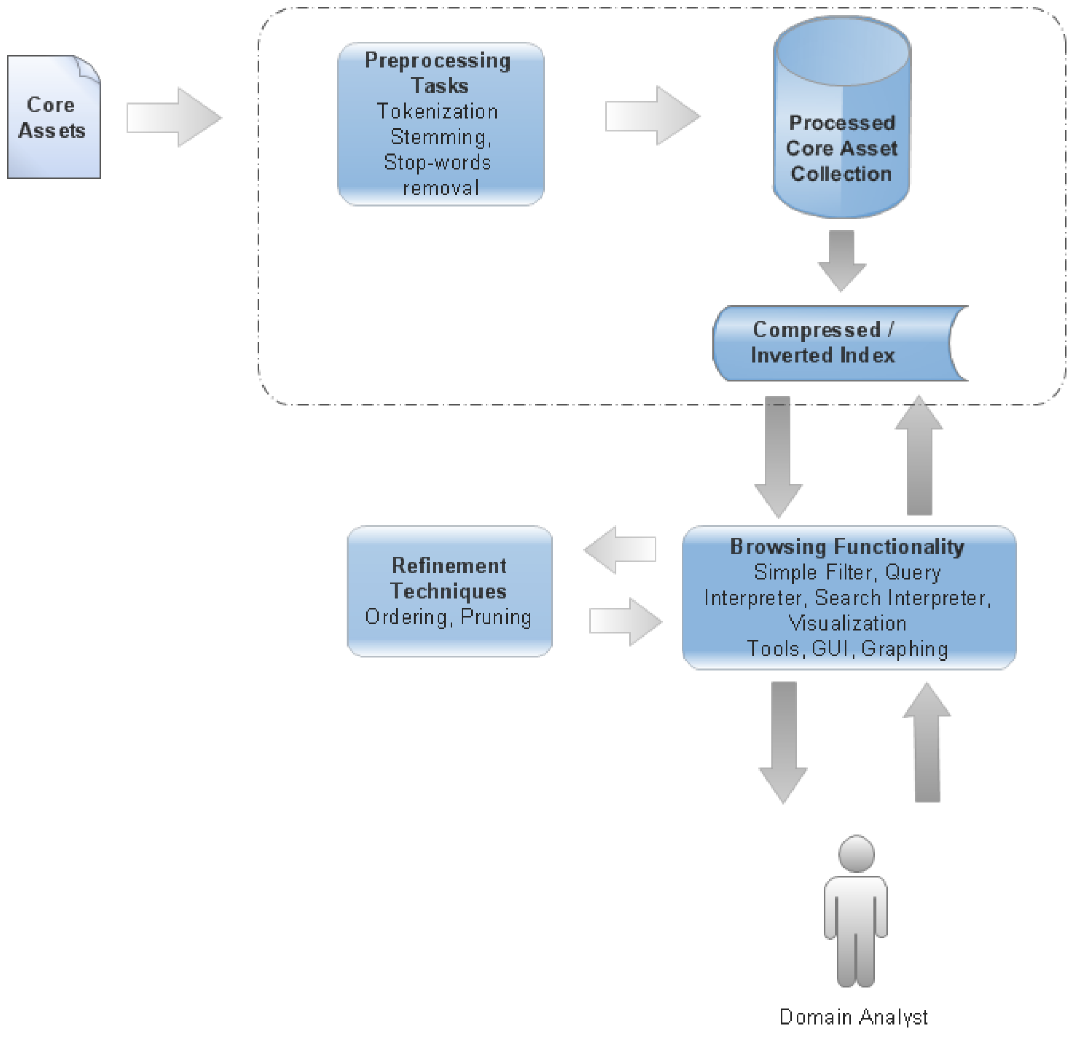

Core Assets
Preprocessing Tasks
Tokenization Stemming, Stop-words removal
Processed Core Asset Collection
Compressed / Inverted Index
Refinement Techniques
Ordering, Pruning
Browsing Functionality
Simple Filter, Query Interpreter, Search Interpreter, Visualization Tools, GUI, Graphing
Domain Analyst

Figure 4.13 Instantiation of Text Mining Architecture for TIRT

4.3.2 Demoiselle Framework Architecture

The Demoiselle Framework project is a platform for Java[1] development that aims the reuse. This platform is a Brazilian system developed by SERPRO[2], a public Brazilian company specialized in data processing sevices for the government. The framework project implements the concepts of integrator framework of technologies. It performs an integration between amount specialized frameworks and garantee the evolution, maintenance and compatibility between each of them (Demoiselle, 2011). According to Tiboni *et al.* (2009), the major framework's contribution is giving the real direction in the use of technologies.

Although the framework has been developed from the federal Brazilian government, the Demoiselle Framework is an open source, available under the LGPL 3[3] and any component that are available or developed for it must be compatible with this licence. Its development is realized in the colaborative way, thus, anyone can colaborate with development or in the documentation development.

The motivation for choosing the Demoiselle Framework was its guidance in the use of technologies, web support, largest community supporting, besides its flexibility with text processing, which is an essential task for TIRT. Futhermore, recent work (Tiboni *et al.*, 2009) has showed that reuse is the greatest contribution of the framework during the development process. The languages used to developed both, Demoiselle and TIRT tool were Java, JavaScript, Java Server Faces (JSF) and Structured Query Language (SQL).

The demoiselle framework consists of a high level layer called *Architectural Framework*, that establishes padronized interfaces to be used for the applications. According to Figure 4.14, the lower layers, consists of the widely frameworks used in the industry. The framework was built on the premises to be extensible, easy to use, stable, configurable, reliable, and have their published documentation.

Futhermore, the Component-Based Development (Component-Based Development (CBD)) with the independency life-cycle of the architectural framework allows the independence of modules, that could be developed in a colaborative way (Tiboni *et al.*, 2009). The components is not part of the architectural framework, considering that they have an independent life-cycle.

Next are described the modules of the framework. One of the main goal of the framework is the flexibility, therefore components developed by third parties, since uses

[1]http://www.sun.com/java/

[2]http://www.serpro.gov.br

[3]http://www.opensource.org/licenses/LGPL-3.0

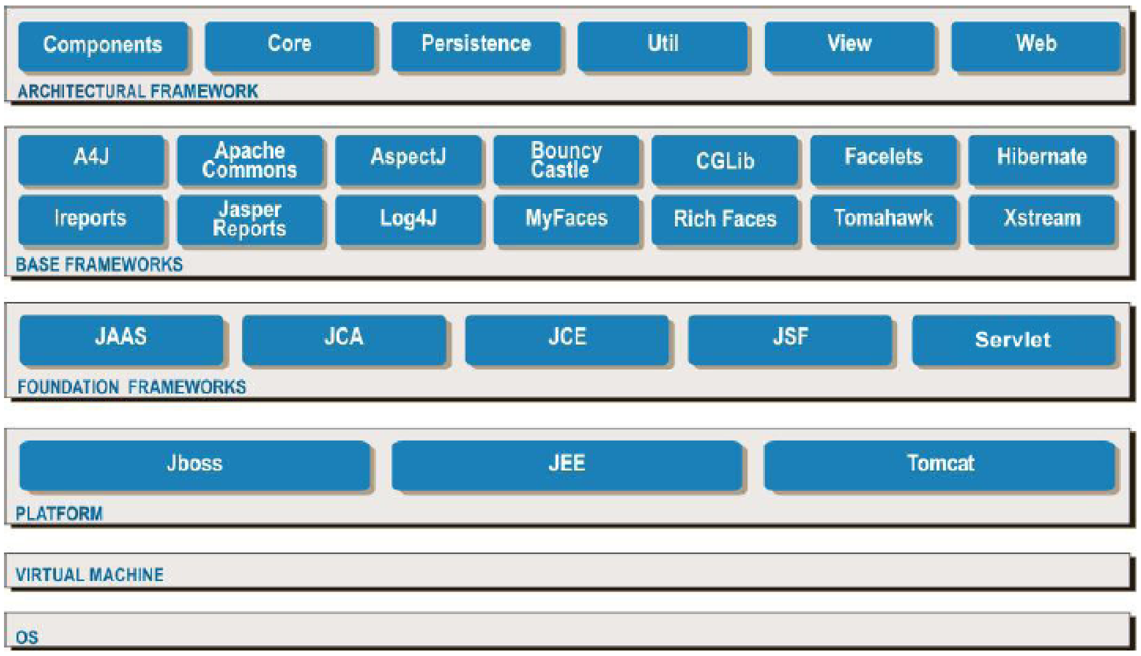


Figure 4.14 Demoiselle Framework Architecture (Demoiselle, 2011)

the interfaces defined by Demoiselle.

- **Core:** set of specifications that are in the basis of the framework, enabling standarlization, extension and integration between the layers of applications based on it.
- **Persistence:** performs system integration with other management systems, ensuring efficiency of data to retrieve, store and process information.
- **Util:** contains utility components that facilitate the work of other features of the framework and its logic modules.
- **View:** contains implementations of specific interface user based on the JSF specification.
- **Web:** implementation of the logic module for Web applications (Java 2 Enterprise Edition (J2EE)), provides common utilities web applications that facilitates treatment for the user's sessions and their requests.

The reference modules proposed for the demoiselle framework is based on layers, as shown in Figure 4.15. However, besides the classical layers defined by the Model-

View-Controller (MVC) pattern, its is distinguished as the *persistence layer*, *transaction*, *security*, *injection dependency* and *message*.

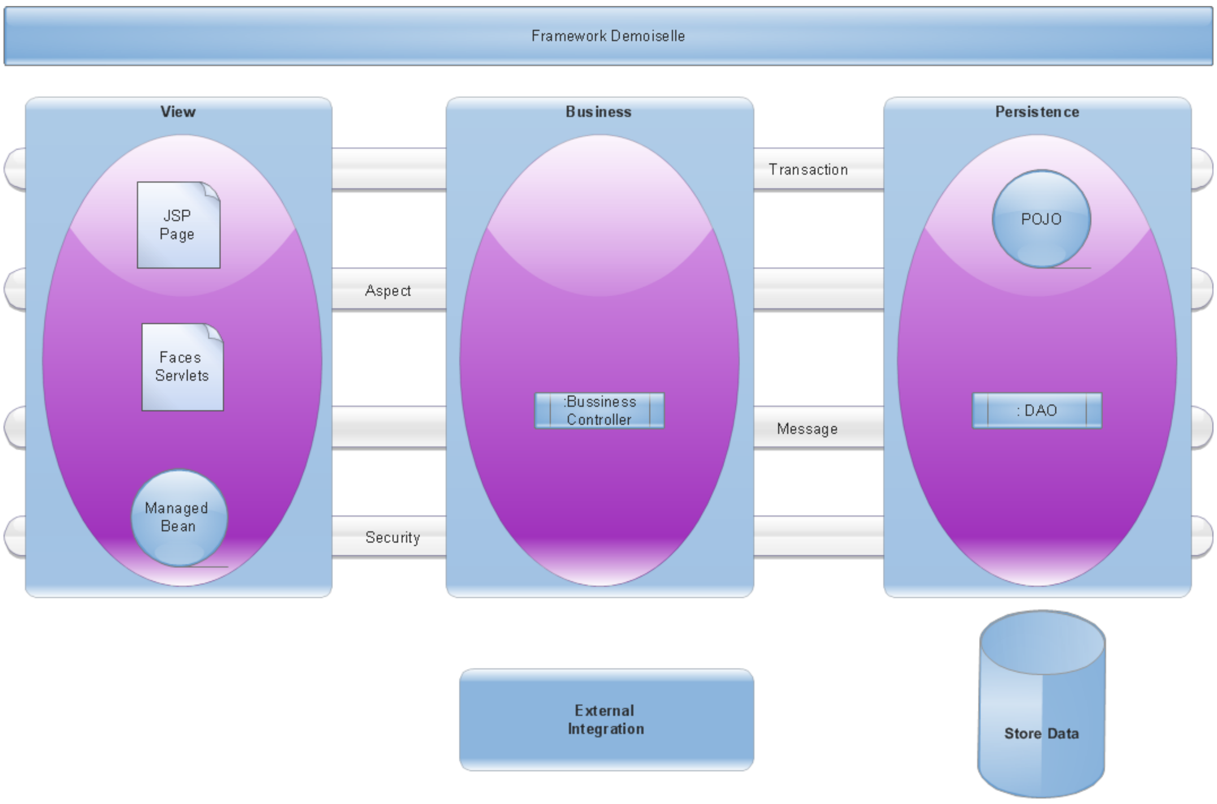


Figure 4.15 Demoiselle Layers (Tiboni *et al.*, 2009)

4.3.3 Demoiselle Framework Architecture Instatiation for TIRT

In order to support the TIRT purpose, the demoiselle framework architecture was instantiated. The main goal of this instantiation was made to accomplish the activities and a way to support the components specified in the Figure 4.13. As can be observed in Figure 4.16, we added the new functionalities in the the layers of the MVC model. In the **Persistence layer**, were added the *Query Parser* and *Indexer*, since the *Text Processor* was incorporated in the **Business** layer, and finally, the **View layer** received the *Visualization*. All these components are following outlined.

Visualization

For Diehl (2007), software visualization is defined as the visualization of artefacts related to software and its development process. Thus, this TIRT module provides a visualization

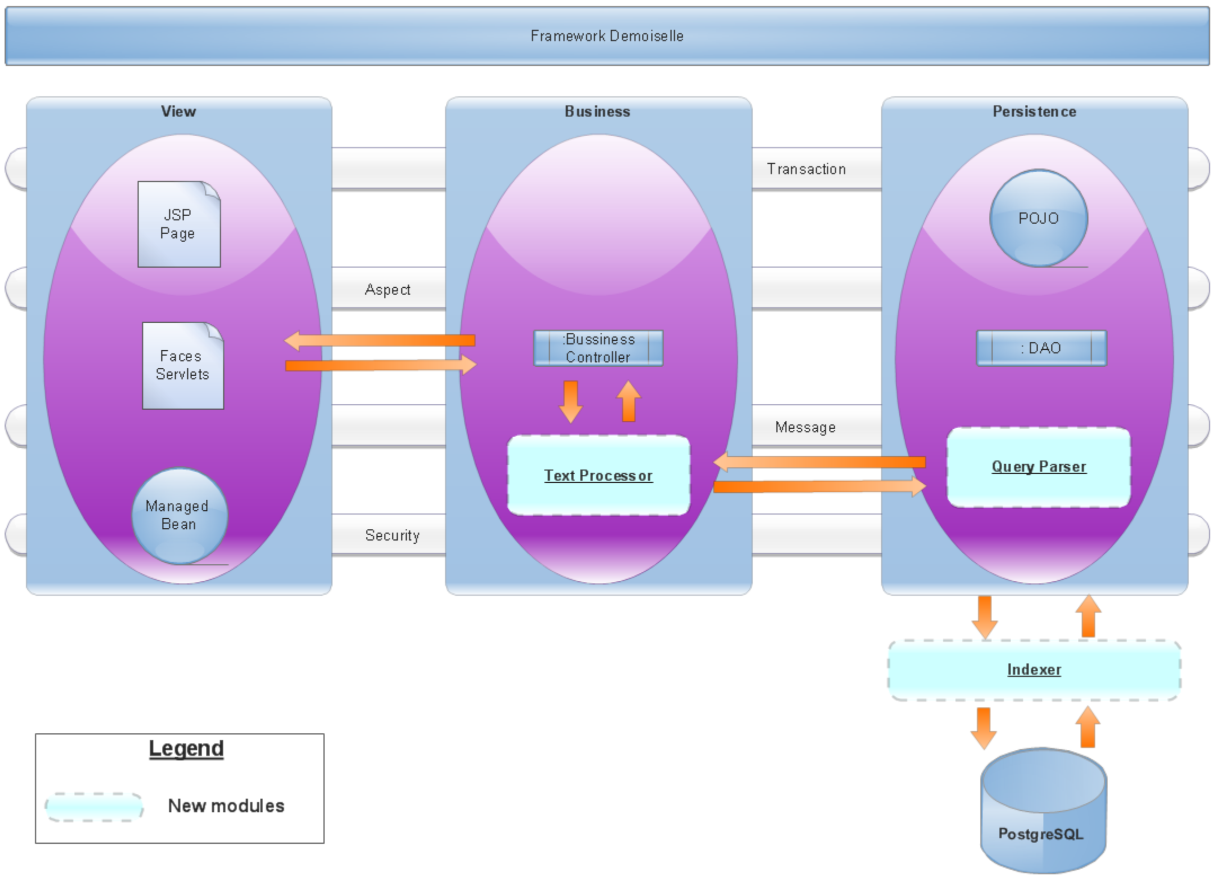


Figure 4.16 Demoiselle Instantiation Layers for TIRT

of the traceability matrix. This matrix is represented with the *indented component*, organized as a horizontal tree that is presented before the update of some core asset, enabling the analysis of the correlated core assets and the analysis of the change impact that can happen with these changes. This analysis can be peformed by the domain analyst, software engineer or another stakeholder responsible for maintenance of the core asset traceability in the SPL.

Text Processor

This module is responsible to process the text from core assets database. Some of the features of this module are: *text tokenization*, *stop-words removal*, and *stemming*. These modules are detailed below:

- **Text Tokenization:** The approach involves breaking the text into sentences and words (Feldman and Sanger, 2007). The stream of characters in a natural language text must be broken up into distinct meaningful tokens before any language processing beyond the character level can be performed. Accordingly to Kaplan (2005), this would be a trivial thing to do, however the language have to be perfectly pontuated, which rarely happens. For this reason, the vocabulary proposed in Section 4.2.4 is essential.

- **Stop-word removal:** This module removes the function words and in general the common words of the language that do not contribute to the semantics of the core assets descriptions and have no real added value. According to Feldman and Sanger (2007), most of these words are irrelevant and can be dropped with no harm to the performance and may even result in improvement owing to noise reduction.

- **Stemming:** Removing suffixes from words by automatic means is an operation which is especially useful in the field of information extraction (Porter, 1980). According to Strzalkowski (1994), word stemming has been an effective way of improving document recall since it reduce words to their common morphological root, thus allowing more successful matches. On the other hand, Strzalkowski (1994) analyzes that stemming tends to decrease retrieval precision, if care is not taken to prevent situations where otherwise unrelated words are reduced to the same stem. In this context, we applied the *Poter algorithm* (Porter, 1980) for reducing words with a common stem to its radical. For example, the words *connect*, *connected*, *connecting* and *connection* are reduced to *connect*.

Query Parser

This module provides the ability to create queries, where the *software engineer* interacts with the system in order to identify possible core assets recommendations. These core assets are stored in the PostgreSQL[4], on open source object-relational database system. It has more than 15 years of active development and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness. This database system was also selected because of its advanced indexing features. In additional, the Hibernate[5] was used in order to facilitate the storage and retrieval of Java domain objects via Object-Relational Mapping (ORM).

Indexer

This module is responsible for processing the contents of each core asset in the database. The index structure is created and maintained by the PostgreSQL engine. As mentioned previously, the PostgreSQL database provides the full text searching or just text search capability to identify natural-language documents that satisfy a query, and optionally to sort them by relevance to the query. In the TIRT context, this type of search is applied in order to identify relevants core assets to recommendation.

4.4 TIRT in Action

This Section presents the TIRT from the user's point view, exposing its main operations and its main screens, showed in Figures 4.17, 4.18 and 4.19.

Beyond the support for recommendation of core assets traceability and support for change analysis, TIRT also focus on usability, following many of the usability heuristics proposed by (Nielsen, 2011), resulting in an clean and intuitive user interface. In additional, Asynchronous Javascript and XML (AJAX) technology were also used in order to improve the user experience with an interative application. The main activities of TIRT are following described:

1. **Menu Bar:** The engineer or domain analyst interact with this area of the tool in order to access all modules of the system. As depicted in Figure 4.17, the menu bar are initially composed by the **Project Configurations** menu, that offers access to the *Project*, *Product*, *Module* and *Actors* management. Therefore, the **Core Assets**

[4]http://www.postgresql.org/
[5]http://www.hibernate.org/

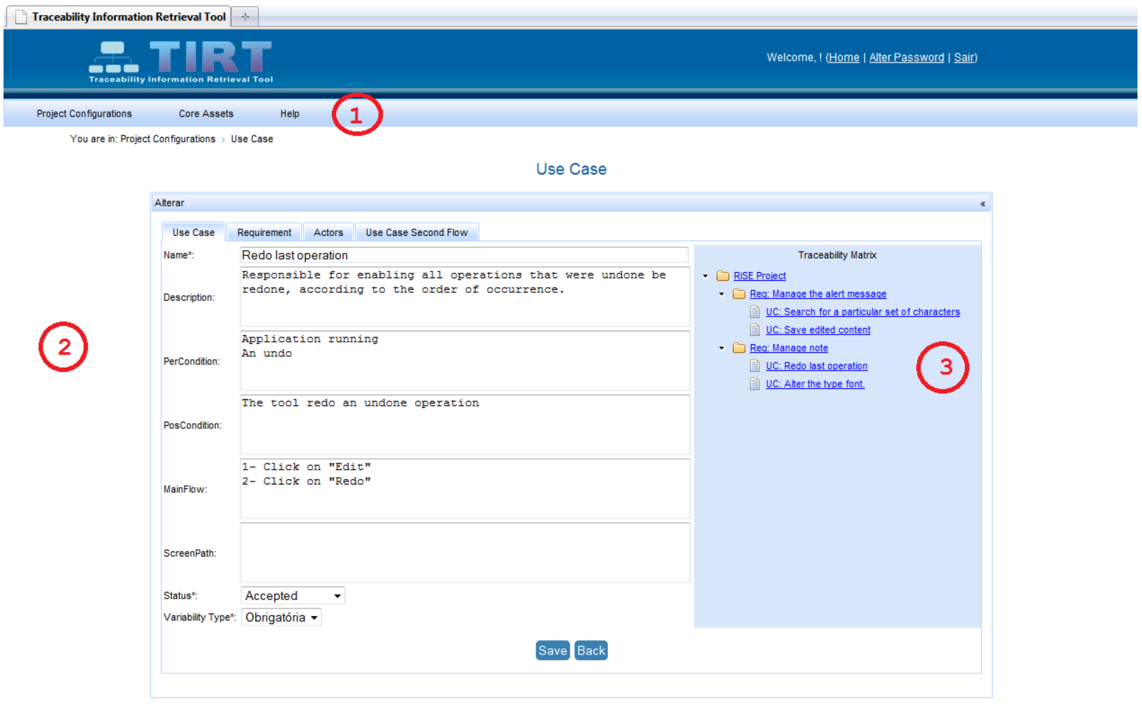
menu provides access to the *Feature*, *Requirement* and *Use Case* management. Finally, the **Help** menu provides instructions and some tips about the *Vocabulary* and *Scenario Recommendation*.

2. **Core Asset Management:** In this part, are provided the basic functionality of the tool, which performs the registration of core assets (e.g. features, requirements and use cases), according to metamodel defined in Section 4.2.1. This activity is also showed in Figure 4.17.

3. **Traceability Matrix:** This area of the tool is responsible for making possible the impact analysis by the *engineer* or *domain analyst*. This information is displayed when a core asset is updated, and the traceability matrix is shown with the purpose of identify the impact analysis (forward and backward) that may accur in the SPL contexting. The matrix view is represented as a *indented component*, organized as a horizontal tree. In this picture, the use case artefact will be updated, thus its relationships (forward and backward) are detailed in order to enable the analyze by the software engineer.

4. **Traceability Recommendation:** This part of the tool presents the search recommendation for core assets, which is one of the main and most important functionalities that TIRT tool provides for the software engineer. It is possible through the extraction and mining of core assets candidates. The software interact with the collection of these core assets in the maintenance process, where one artefact should be modified in order to reflect a change, due to a evolution or corrective action in the SPL. Thus, in the moment of update, a core asset relation is indicated according to the traceability scenarios defined in Section 4.2.3. In addition, when a core asset is being view, the tool must extract relevant information from it, such as descriptions, related core assets, restrictions, and so on;

5. **Help:** TIRT provides a detailed explanation about the characterization of the *vocabulary*, as was outlined in Section 4.2.4. Moreover, the system also must provide details about the *scenarios of recommendation* that are outlined in Section 4.2.3.


Traceability Information Retrieval Tool
TIRT
Traceability Information Retrieval Tool
Welcome, ! (Home | Alter Password | Sair)
Project Configurations
Core Assets
Help
1
You are in: Project Configurations › Use Case
Use Case
Alterar
Use Case
Requirement
Actors
Use Case Second Flow
2
3
Save
Back

Name*: Redo last operation

Description: Responsible for enabling all operations that were undone be redone, according to the order of occurrence.

PerCondition: Application running
An undo

PosCondition: The tool redo an undone operation

MainFlow: 1- Click on "Edit"
2- Click on "Redo"

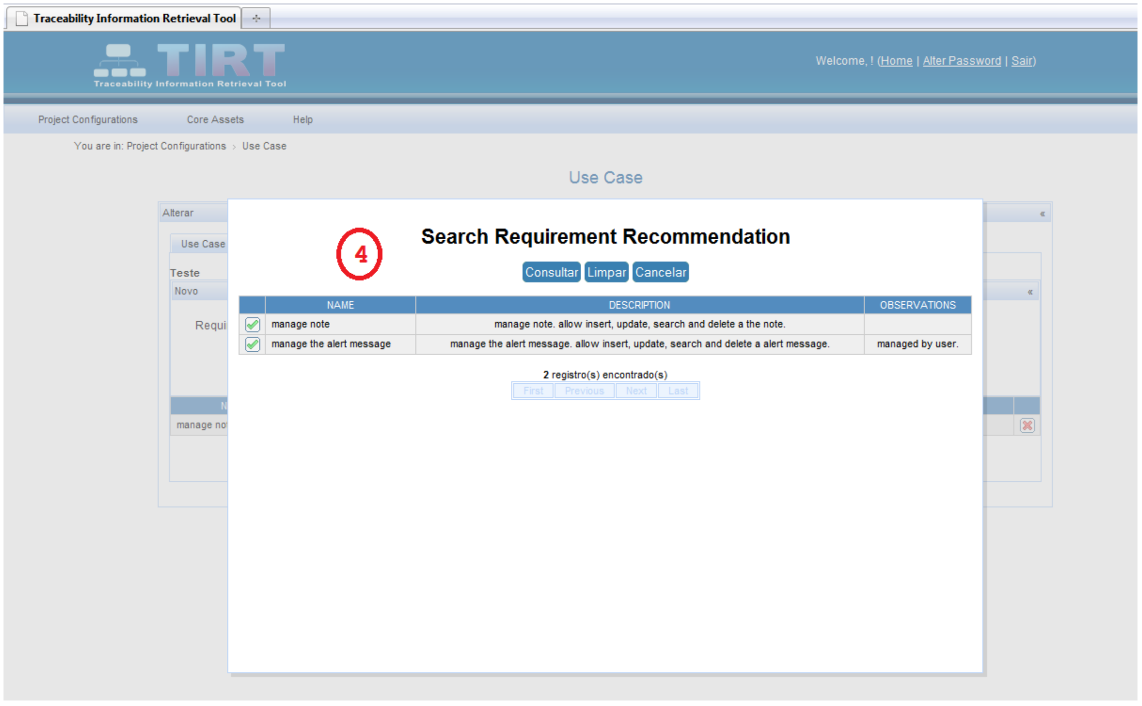
ScreenPath:

Status*: Accepted

Variability Type*: Obrigatória

Traceability Matrix
- RiSE Project
  - Req: Manage the alert message
    - UC: Search for a particular set of characters
    - UC: Save edited content
  - Req: Manage note
    - UC: Redo last operation
    - UC: Alter the type font.

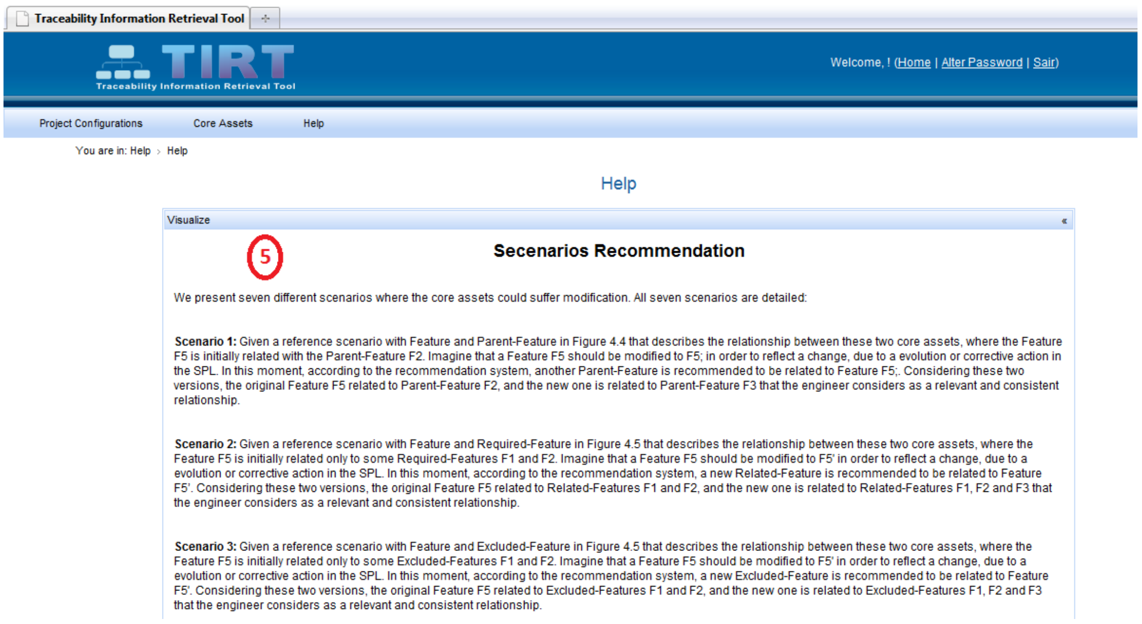
Figure 4.17 Tree View of Analysis Impact


Traceability Information Retrieval Tool
TIRT
Traceability Information Retrieval Tool
Welcome, ! (Home | Alter Password | Sair)
Project Configurations
Core Assets
Help
You are in: Project Configurations › Use Case
Use Case
Alterar
Use Case
Teste
Novo
Requi
manage no
4
Search Requirement Recommendation
Consultar
Limpar
Cancelar
First
Previous
Next
Last

	NAME	DESCRIPTION	OBSERVATIONS
✓	manage note	manage note. allow insert, update, search and delete a the note.	
✓	manage the alert message	manage the alert message. allow insert, update, search and delete a alert message.	managed by user.

2 registro(s) encontrado(s)

Figure 4.18 Requirements Recommendation Screen


Traceability Information Retrieval Tool
TIRT
Traceability Information Retrieval Tool
Welcome, ! (Home | Alter Password | Sair)
Project Configurations
Core Assets
Help
You are in: Help › Help
Visualize
5

Help

Secenarios Recommendation

We present seven different scenarios where the core assets could suffer modification. All seven scenarios are detailed:

Scenario 1: Given a reference scenario with Feature and Parent-Feature in Figure 4.4 that describes the relationship between these two core assets, where the Feature F5 is initially related with the Parent-Feature F2. Imagine that a Feature F5 should be modified to F5; in order to reflect a change, due to a evolution or corrective action in the SPL. In this moment, according to the recommendation system, another Parent-Feature is recommended to be related to Feature F5;. Considering these two versions, the original Feature F5 related to Parent-Feature F2, and the new one is related to Parent-Feature F3 that the engineer considers as a relevant and consistent relationship.

Scenario 2: Given a reference scenario with Feature and Required-Feature in Figure 4.5 that describes the relationship between these two core assets, where the Feature F5 is initially related only to some Required-Features F1 and F2. Imagine that a Feature F5 should be modified to F5' in order to reflect a change, due to a evolution or corrective action in the SPL. In this moment, according to the recommendation system, a new Related-Feature is recommended to be related to Feature F5'. Considering these two versions, the original Feature F5 related to Related-Features F1 and F2, and the new one is related to Related-Features F1, F2 and F3 that the engineer considers as a relevant and consistent relationship.

Scenario 3: Given a reference scenario with Feature and Excluded-Feature in Figure 4.5 that describes the relationship between these two core assets, where the Feature F5 is initially related only to some Excluded-Features F1 and F2. Imagine that a Feature F5 should be modified to F5' in order to reflect a change, due to a evolution or corrective action in the SPL. In this moment, according to the recommendation system, a new Excluded-Feature is recommended to be related to Feature F5'. Considering these two versions, the original Feature F5 related to Excluded-Features F1 and F2, and the new one is related to Excluded-Features F1, F2 and F3 that the engineer considers as a relevant and consistent relationship.

Figure 4.19 Help of Scenarios Recommendation

4.5 Chapter Summary

This Chapter presented the main aspects of TIRT tool, including the set of functional and non-functional requirements, architecture, frameworks and technologies adopted during its construction.

Futhermore, it presented the proposal for traceability recommendation, that outlines the recommendation scenarios of traceability, described in order to facilitate the maintenance activities of SPL. This recommendation approach aids the decision making, supporting representation of dependencies between different core assets. Finally, this Chapter presented the TIRT from the user's point view, exposing its main operations.

Next Chapter presents the evaluations peformed with a group of subjects to verify the tool helpfulness to the maintenance traceability activity.

5 TIRT Evaluation

If you can dream it, you can do it.

—WALT DISNEY (Cartoonist)

New methods, techniques, languages and tools should not just be suggested, published and marked. It is crusial to evaluate these new inventions and proposals. Experimentation provides this opportunity and should be used. In other words, we should use the methods and strategies available when conducting research in software engineering (Wohlin *et al.*, 2000). In this way, this Chapter presents the experimental study in order to to evaluate the proposed tool and the approaches outlined in the previous Chapter.

The remainder of this Chapter is organized as follows. Section 5.1 introduces some definitions to clarify the steps and concepts of the experimental study. Section 5.2 presents the definition of the experiment. Section 5.3 presents the planning of the experiment. The details about the operation of the experiment are presented in Section 5.4. Section 5.5 presents the analysis and interpretation of the results. Section 5.6 presents the conclusion and finally, the Chapter summary is described in Section 5.7.

5.1 Introduction

This dissertation presented *TIRT - Traceability Information Retrieval Tool*, that focus on the maintenance activities regarding to the traceability relationship among the different core assets in a Software Product Lines through its recommendation system. In additional, the tool was also built in order to aid in the process of impact analysis. Thus, we believe that the time spent in these activities can be reduced and less error prone. In order to evaluate the tool, a controlled experiment was performed with 10 subjects.

so that mode, conclusive results can be obtained.

This experimental study is based on the process proposed in (Wohlin *et al.*, 2000), see Figure 5.1. It is divided into the following main activities: **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, analysed and evaluated in the analysis and interpretation. Finally, the results are **presented and packaged** in the **conclusion**.

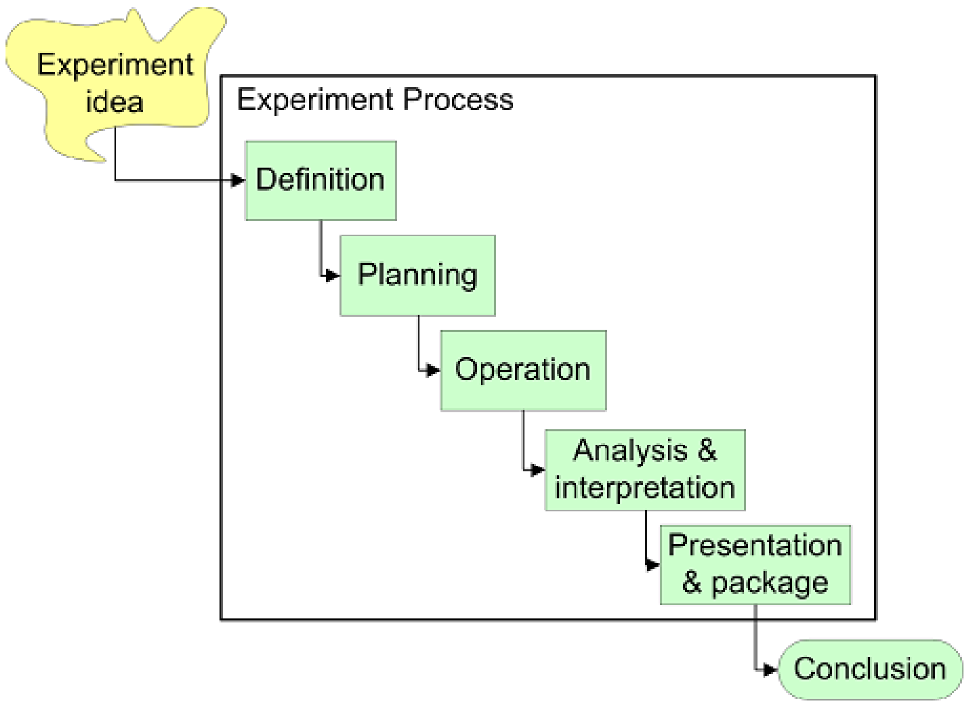


Figure 5.1 Overview of the Experimental Process (Wohlin *et al.*, 2000)

The process detailed in Figure 5.1 is not supposed to be a ”true” waterfall model, because it is an iterative and it may be necessary to go back and refine a previous activity before continuing with the experiment.

The main objetive of an experiment is mostly to evaluate a *hypothesis*. Hypothesis testing normally refers to the former and the latter is foremost a matter of building a relational model based on the data collected. In additional, two kinds of variable are studied in an experiment, *independent* and *dependent*. These variables are studied in order to analyse their outcomes when are varied some of the input variable to a process. The variables that are objects of the study which see the effect of the changes in the independent variables are called *dependent variables*. All variables in a process that are manipulated and controlled are called *independent variables* (Wohlin *et al.*, 2000).

According to Wohlin *et al.* (2000), an experiment studies 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 *traceability matrix* that will be updated using different systems. The people that apply the treatment are called subjects. In general, an experiment consists of a set of tests where each test is a combination of *treatment*, *subject* and *object* (Wohlin *et al.*, 2000).

5.2 The Definition

The first activity is definition. In this phase, the foundations of the experiment are determined, presenting why the experiment will be conducted. In addition, the hypothesis has to be stated clearly. Futhermore, the objective and goals of the experiment must be defined.

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: *Goal*, *Question* and *Metric*. The metrics also are defined later as *independent variables*.

5.2.1 Goal

According to the paradigm, the main objective of this experiment is:

G1: To analyse the tool for the purpose of evaluating it with respect to the *recall* and *precision* of *recommendation of core assets traceability* and *effort (time saving)*, in the view point of software engineers.

Object of Study *(What is studied?)*: The objective of study is the tool used to analyse and perform the *recommendation of core assets traceability* and their ability in term of *effort reduction (save time)*.

Purpose *(What is the intention?)*: The purpose is to evaluate the *recommendation activities of core assets*. We would like to understand if the tool can bring benefits to maintenance of core asset traceability, according to the *scenario recommendation* and *vocabulary* proposed.

Quality Focus *(Which effect is studied?)*: The benefits gains obtained by the use of the tool is the *recommendation of core assets traceability*. Moreover, we also analyse *effort reduction (time saving)* in the core asset maintenance.

Perspective *(Whose view?)*: The perspective is from the software engineer's point of view.

5.2.2 Questions

In order 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 one concerned with the subjects' feedback about the new approach adoption. The questions are described as follow:

- **Q1:** Is there increase of the effectiveness of core assets management with the *recommendation approach* performed in the new tool adoption? In order to understand if the *recommendation of core assets traceability* provides effective recommendations, the *recall* and *precision* are analysed.

- **Q2:** Is there reduction of the effort *(time)* that *software engineers* spend to perform the core assets management? This quantitative issue is observed in order to analyse if TIRT tool provides saving time.

- **Q3:** Did the software engineers have difficult to use the tool? In order to understand the difficult that the submitters will face during the experimentation, they will be asked to describe the difficulties encountered during the experiment.

5.2.3 Metrics

The metrics are a set of data that is associated with every question in order to answer it in a quantitative way.

Two fundamental metrics are used to assess the efficiency of any information retrieval mechanism: *recall* and *precision* (Kowalski and Maybury, 2002). Both *recall* and *precision* take on values between 0 and 1.

M1: *Precision* of core assets relationship that were recommended; The capacity of avoiding irrelevant results is measured by the *precision* metric. The following equation is used to calculate this metric.

$$P = \frac{TP}{TP + FP} \tag{5.1}$$

TP: True-Positive. *FP:* False-Positive.

M2: *Recall* of core assets relationship that were recommended; The capacity of a mechanism of retrieving relevant results is measured by the *recall* metric. The following equation is used to calculate this metric.

$$R = \frac{TP}{TP + FN} \tag{5.2}$$

TP: True-Positive. *FN:* False-Negative.

M3: *Effort Variation (EV)* of core asset management (Basili *et al.*, 1996); The following equation is used to calculate *effort* metric.

$$EV = \frac{ETIRT}{ESPLMT} \tag{5.3}$$

ETIRT: Average Effort to subjects execute the core asset management using the Traceability Information Retrieval Tool (TIRT), a different tool.

ESPLMT: Average Effort to subjects execute the core asset management using Software Product Lines Management Tool (SPLMT), the tool used to initially modeling the *MedicWare's* core assets.

The *Effort Variation* metric has range [0, N], where *EV* 0 indicates the maximun effort *Time Spent* reduction when using the TIRT tool, and *EV* >= 1 indicates that the effort to execute the core asset management using TIRT tool is equals, or even higher than using SPLMT tool.

5.3 The Planning

After the *definition* of the experiment, *the planning* activity takes place. The *definition* determines the foundation for the experiment - *why* the experiment is conducted - while the *planning* prepares for *how* the experiment is conducted. The Planning phase of an experiment can be divided in some steps. Based on the experiment definition, the context of the experiment is determined in details, as well as the hypothesis and an alternative hypothesis. The next step in the planning activity is to determine the independent and dependent variables. The subjects of the study are also identified. Futhermore, the questions of validity of results also are consider. Validity can be divided into three major classes: internal, external, and conclusion validity (Wohlin *et al.*, 2000).

5.3.1 Context Selection

The experiment will be performed as an off-line experiment. It will run using software engineers of *Pitang Software Factory*[1]. In addition, the experiment will be performed distributed, which means that the subjects are free to choose their work environment, such as their home or company laboratories.

Regarding the data used in the experiment, the subjects will use core assets from a private company working in the medical information management domain, called *MedicWare Informatics System LTDA*[2] which has negociated a cooperation project with RiSE Labs.

MedicWare works with software development for the medical domain since 1994 and is located at Salvador, Bahia, Brazil. The company was created in 1994 offering strategic and operational solutions integrated for hospitals, clinics, labs and private doctor's office.

The company has four products presented in the decreasing order according to their size. The first one *(SmartHealth)* is a product of 35 modules or sub-domains, and manages the whole area of a hospital, from financial to patient aspects. The second one *(SmartClin)*, composed of 28 modules, performs the clinical management supporting activities related to medical exams, diagnostics and so on. The third one *(SmartLab)* has 28 modules and integrates a set of features to manage labs of clinical pathology. Finally, the last one *((SmartDoctor)* is the only web product and composed of 11 modules to manage the tasks and routines of a doctor's office. Figure 5.2 shows the correlation among the products.

[1]http://www.pitang.com/ - *Pitang* is the Software Factory of C.E.S.A.R., offering system development, consulting, software factory, testing factory, legacy migration and outsourcing services.

[2]http://www.medicware.com.br/

As it can be seen in the figure, SmartHealth is the biggest product and some of its features compose the other ones.

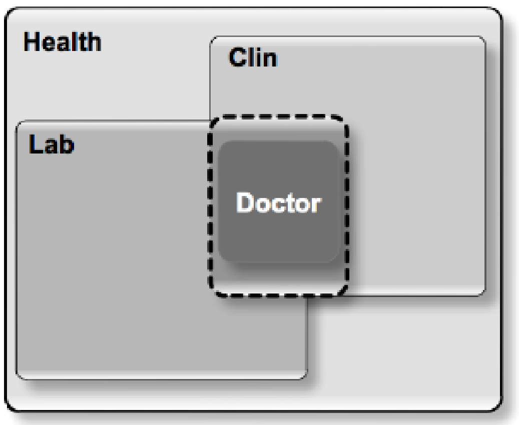


Figure 5.2 MedicWare Projects

This experiment is concerned with the adoption of a tool developed to aid the maintenance of core assets links, focusing on analysis of recommendation of core assets traceability. The *Traceability Information Retrieval Tool (TIRT)* will be compared with the *Software Product Lines Management Tool (SPLMT)*, that was used to modeling the MedicWare core assets. In addition, it is important to highlight that SPLMT tool does not have the recommendation approach, proposed and developed in TIRT. Thus, the subjects will use both tools and it will be analysed the *effort reduction (time saved)* and *recommendation of core assets traceability* performed by the tool. Initially, the subjects will be trained in several aspects of SPL and in usage of the tool.

5.3.2 Hypothesis 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. According to Wohlin *et al.* (2000), in the planning phase, the experiment definition is formalized into two hypotheses: *Null Hypothesis* and *Alternative Hypothesis*.

Null Hypothesis: The null hypothesis determines that there is no benefits of using TIRT to perform the *recommendation of core assets traceability*, besides not have *time saved*. The null hypothesis are:

H0: μ *effort(time saved) with* TIRT > 1

H0: *μ precision* TIRT < 50%
H0: *μ recall* TIRT < 50%

Alternative Hypothesis: The alternative hypothesis determines that TIRT can *save more time* and that there is benefits of using TIRT to perform the *recommendation of core assets traceability*.

H1: *μ effort(time saved) with* TIRT <= 1
H1: *μ precision* TIRT >= 50%
H1: *μ recall* TIRT >= 50%

We perform the *effort (time saved)* measurements between the TIRT and SPLMT tools, but we also perform an analysis of the hypotheses in terms of *precision* and *recall* (Kowalski and Maybury, 2002) measurements, considering the percentage greater than or above of 50% as encouraging results in the context of TIRT tool recommendation.

5.3.3 Variable Selection

Before any design can start, we have to choose the *dependent* and *independent variables*. The independent variables are those variables that we control and changes in the experiment. On the other hand, the effect of the treatments is measured in the dependent variable or variables (Wohlin *et al.*, 2000).

In this experiment we have only one *independent variable*, which is the tool used to perform the *recommendation of core assets traceability*. The *dependent variables* for this experiment are *(a) precision*, *(b) recall* of core assets recommendation and *(b) the effort (time spent)* with maintenance of core assets.

5.3.4 Selection of Subjects

The subjects of this experiment will be selected by the technique of *convenience sampling*, which the nearest and most convenient persons are selected as subjects (Wohlin *et al.*, 2000). The subjects of this study are software engineers of *Pitang Software Factory*. For this evaluation will be selected ten (10) subjects. The questionnaire in Appendix A.3 were used to gather data about subjects education and background.

5.3.5 Experimental Design

An experiment consists of a serie of tests of the treatments. A design of an experiment describes how the tests are organized and run. When designing an experiment, many aspects must be considered, however, the general design principles are *randomization*, *blocking* and *balancing*.

- **Randomization:** The randomization applies on the allocation of the objects, subjects and in which order the tests are performed. Randomization is also used to select subjects that is representative of the population of interest (Wohlin *et al.*, 2000). Since all subjects will participate in both treatments, no randomization is required.

- **Blocking:** is used to systematically eliminate the indesired effect in the comparison among the treatments (Wohlin *et al.*, 2000). We believe the division of subjects are not necessary, since all work in the same company with similar knowledge degree. The questionnaire in Appendix A.3 were used to gather data about education and background, which determined that blocking design were needed.

- **Balancing:** Whe have a balanced design, which means that there is the same number of persons in each group.

The experimental design used to perform the experiment is *one factor with two treatments design*. With this design, we would like to compare the two treatments against each other (Wohlin *et al.*, 2000). In such case, the factor is TIRT. Thus, there is a treatment with such tool and another with the SPLMT tool. It is important to note that the data used in the experiment was the result of cooperation between the RiSE and *MedicWare*, in order to identify the core assets of the company. The two treatments are described as follows and depicted in Figure 5.3:

- **Treatment 1:** In the first treatment, the subjects will use the SPLMT tool in order to perform the core assets update, according to the *firt list* and *traceability matrix*. In addition, the subjects have to register the time spent in this activity *timesheet* in Appendix A.2.

- **Treatment 2:** In the second treatment, the subjects will receive the *second list* and its corresponding *traceability matrix*. The subjects also have to register the time spent in this activity in a *timesheet*. After the update, the TIRT tool will

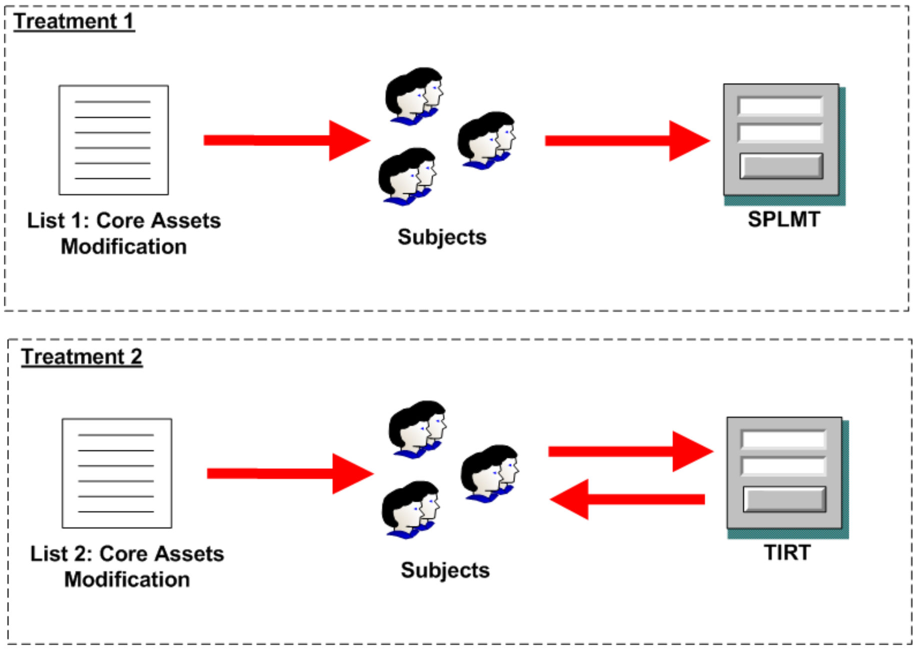

Treatment 1
List 1: Core Assets Modification
Subjects
SPLMT
Treatment 2
List 2: Core Assets Modification
Subjects
TIRT

Figure 5.3 Experiment Design: One factor with two treatment design

recommend possible *core asset traceabilities*. Thus, the subjects also have to quantify the recommendations results, such as: *correct results*, *incorrect results* and *unlisted recommendations*. It is important to highlight that we had to translate the original data to the English language, because the *stop-word removal* and *stemming* algorithms detailed in Chapter 4 and implemented in TIRT tool only considers the English language. It is a limitation to be solved in the future.

In both of treatments, the experiment was applied in the context of one traceability scenario *(Use Case and Requirement)*. The remainder of traceability scenarios proposed also could be analysed. It did not happen, because the time constraint to conduct the experiment, however, in in order to obtain more conclusive results, other experiments can be performed, as detailed in the following Chapter.

5.3.6 Instrumentation

In order to guide the subjects in the experiment, we provided the experiment description and guidelines with all support material, the list of core asset modifications, and questionnaires.

The results of the experiments, such as *recall*, *precision* and *effort (time saved)* will be collected. In order to guarantee more precision for the data collected, the participants will be oriented to use timesheet to register the time, while performing the *core assets management*. In addition, questionnaires will be elaborated to gather qualitative data from subjects. The instruments are presented in Appendix A.

5.3.7 Pilot Project

Before performing the study, a pilot project was conducted with the same structure of the *Definition* and *Planning*. The pilot project was performed by two subjects, who were trained in SPL and tool usage. During training, we highlight some aspects related to the core assets and manipulation of tool. The training was performed for approximately 2 hours. These subjects are not part of the subject identified in the *Selection of Participants*. In this way, the pilot project provided a feedback about the *questions*, *metrics* and *design* adopted in this experiment.

5.3.8 Validity Evaluation

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

Conclusion Validation: This validation is concerned with the relationship between the treatment and the outcome (Wohlin *et al.*, 2000). In order to evaluate and interpret the results of the experiment, the descriptive statistics and statistical hypothesis testing will be collected during the experiment.

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). This study is evaluated with 10 (ten) subjects with similar background on core assets management, providing a good internal validity.

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). Thus, the following external validity were identified:

- **Generalization of subjects:** The study will be conducted with Software engineers who have similar background. Thus, the subjects will not be selected from a general population. In this case, if these professionals succeed using the tool and approach, we cannot conclude that tool would use it succesfuly with another groups too. On the other hand, negative conclusions have external validity, because if the professionals fail in using the tool, then this is strong evidence that another practice would fail too;

- **Time constraint:** We identified that due to the time constraints, the scope of experiment was recuced, which could affect the experiment results. Thus, it can be considered as a external validity.

Validity Threats: There is a conflit between some of the types of validity threats (Wohlin *et al.*, 2000). Thus, the following validity treatments were identified:

- **Internet connection contraints:** The tool will run in the web, thus the time to *manage the core assets* might be comprised by the quality of Internet connection from the subject environment. In this way, the results can be imprecise.
- **Environment:** The subjects will be free to do the task for the experiment in a place that suits them best. Thus, the different environments can have positive or negative influences for the correct execution of the experiment.
- **Boredom:** The subjects may be disappointed or even upset during the experiment, since the experiment will be conducted for a long period of time, but in order to reduce the boredom of the experiment, we recommended short interval during the experiment execution.

5.4 The Operation

When an experiment has been designed and planned it must be carried out in order to collect the data that should be analysed. This is what we mean with the operation of an experiment. In the operational phase of an experiment, the treatments, depicted in Section 5.3.5 are applied to the subjects. Thus, this means that this part of an experiment is the part where the experimenter meets the subjects (Wohlin *et al.*, 2000).

The operational phase of this experiment 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 (Wohlin *et al.*, 2000).

5.4.1 Preparation

The subjects of this experiment were selected by probability technique of *convenience sampling*, which the nearest and most convenient persons are selected as subjects (Wohlin *et al.*, 2000). The subjects of this study are software engineers of *Pitang Software Factory*. For this evaluation were selected 10 (ten) subjects.

Before the experiment can be executed, all experiment instruments defined in Section 5.3.6 must be prepared. After that, a list of core assets with its traceability matrix is given to each subject. This *traceability matrix* contains the relationship between two core assets of MedicWare project. In this sence, the subjects have to update each core asset in order to relate with its correct core asset relationship. Nevertheless in the TIRT situation, the tool will recommend possible *core asset traceabilities*. Thus, the subjects also have to quantify the recommendations results in order to understand and analyse its results, such as: *correct results*, *incorrect results* and *unlisted recommendations*.

As mentioned in the Planning Section 5.3, we had to translate the original data to the English language, because the *stop-word removal* and *stemming* algorithms detailed in Chapter 4 and implemented in TIRT tool only considers the English language.

5.4.2 Execution

The experimental study was conducted in July of 2011 with the subjects described previously. The participants were free to do the experimentation in a place that suits them best. In addition, information about the experiment execution were detailed, via email and skype, to the subjects. The questionnaires and timesheet used in this study are presented in Appendix A.

The experiment focus on analysis of recommendation of core assets traceability. The *Traceability Information Retrieval Tool (TIRT)* will be compared with the *Software Product Lines Management Tool (SPLMT)*, because the SPLMT does not have the recommendation approach, developed in the TIRT tool.

The treatmens and design detailed in Section 5.3.5 were aplied in this phase. Initially a SPL training was performed, because most subjects have no expertise in the reuse area, since all work in the same company with similar knowledge degree. As mentioned before, the questionnaire in Appendix A.3 was used to gather data about education and background.

Table 5.1 details the profile of each subject. The *Subject IDs* are represented in the first column, the *Graduation* column represents the quantifier of years since graduation, the amount of projects that subjects participated are described in the third column, the *Project Roles* are represented in the fourth column, and finally, *Software Reuse* and *SPL* columns represent the expertise in these areas respectively.

ID	*Graduation*	*Projects*	*Roles*	*Software Reuse*	*SPL*
1	0.5	5	*Developer*	*Low*	*None*
2	0.5	8	*Developer*	*Medium*	*None*
3	1.5	6	*Developer*	*Medium*	*None*
4	3	8	*Developer*	*Medium*	*Low*
5	1	4	*Tester*	*None*	*None*
6	3.5	7	*Developer/Tester*	*Low*	*None*
7	3.5	7	*Developer/Architect*	*High*	*Low*
8	3	8	*Tester/Developer*	*Low*	*None*
9	0.5	3	*Tester/Developer*	*None*	*None*
10	1	4	*Developer*	*Low*	*None*

Table 5.1 Subject Profile

5.4.3 Data Validation

Data was collected from ten (10) subject, however, two of them *(Subjects ID 2 and 7)* did not participate of treatments and qualitative questionnaire, affecting the data validation. In this way, we had eight (8) subjects for statistical and quality analysis.

5.5 Analysis and Interpretation

After collecting experimental data in the operation phase, we would like to be able to draw conclusions based on it. Nevertheless, to be able to draw valid conclusions, we must interpret the experiment data (Wohlin *et al.*, 2000). In this phase, the *quantitative* and *qualitative* data were analysed.

5.5.1 Quantitative Analysis

Table 5.2 details the collected data during the experiment. The column *IDs* represents the subjects identifications. It is important to highlight that the subjects *#2* and *#7* were excluded of this analysis, because they did not participate of treatments and qualitative questionnaire, affecting the data validation. The next columns represent the time spent in minutes on analysis of both tools *SPLMT* and *TIRT* respectively. Finally, the table depicts the *Precision* and *Recall* metrics obtained with the TIRT tool.

ID	*SPLMT (min)*	*TIRT (min)*	*TIRT Precision (%)*	*TIRT Recall (%)*
1	14.41	23	64.5	40.5
3	40	44.30	64.5	40.5
4	32	42.10	64.5	40.5
5	34.30	35	58.2	30.3
6	22.30	35	55.2	28.7
8	35	37.3	64.5	40.5
9	20	36.40	64.5	40.5
10	23	33	64.5	40.5

Table 5.2 Collected data during the experiment

From Table 5.2 we can indicate that subjects using SPLMT to analysis the core assets traceability spent less time than using TIRT. However, it is important to highlight that during the analysis of recommendations in TIRT, participants also had to quantify the quality of the recommendations, enumerating the *total*, *correct*, *incorrect* and *unlisted* recommendations. Thus, these activities can be impacted and justify the excess time of TIRT.

Table 5.3 details descriptive statistics about the experiment results. According to Wohlin *et al.* (2000), descriptive statistics can be used to describe and presenting interesting aspects of the data set. Regarding the time spent on analysis, SPLMT had *mean* value of 27.43 minutes, *minimum* of 14.41 and *maximum* of 40 minutes, while TIRT had *mean* value of 35.51 minutes, *minimum* of 23 and *maximum* of 44.30 minutes. In addition, Table 5.3 also brings descriptive statitics about the *precision* and *recall* metrics of TIRT tool. Regarding the *precision* metric, the *mean* value is 62.55 percent, the *minimum* 55.2 and *maximum* value 64.5, while the *recall* metric has *mean* value of 37.75, *minimum* of 28.7 and *maximum* value of 40.5.

	SPLMT (min)	*TIRT (min)*	*TIRTPrecision (%)*	*TIRTRecall (%)*
Mean	27.43	35.51	62.55	37.75
Minimum	14.41	23	55.2	28.7
Maximum	40	44.30	64.5	40.5

Table 5.3 Descriptive Statistics

From Table 5.3, it indicates that although TIRT spent more time in the analysis of core assets traceability, its bring the *recommendation feature* with acceptable data of *precision*.

Hypothesis Testing

The objective of hypothesis testing is to verify if it is possible to reject a certain null hypothesis *(H0)* (Wohlin *et al.*, 2000), defined in the Subsection 5.3.2.

The results detailed in Table 5.4 shows that it spent more time using TIRT in the core assets analysis. Thus, it does not reject the null hypothesis *(H01)*: μ *effort(time saved) with* TIRT > 1. The metric used to evaluate the *precision* of the traceability recommendation presents the value of *62.55%*. Thus, it rejects the null hypothesis *(H02)*: μ *precision* TIRT < 50%. Finally, the metric used to evaluate the *recall* of the traceability recommendation presents the value of *37.75%*. Thus, it does not reject the null hypothesis *(H03)*: μ *recall* TIRT < 50%.

	Null Hypothesis	*Results*	*Rejected*
H01	μ *effort(time saved) with TIRT* > 1	1.29	*No*
H02	μ *precision TIRT* $< 50\%$	62.55%	*Yes*
H03	μ *recall TIRT* $< 50\%$	37.75%	*No*

Table 5.4 Hypothesis Analysis

As mentioned previously, it is important to highlight that during the analysis of recommendations in TIRT, participants also had to quantify the quality of the recommendations, enumerating the *total*, *correct*, *incorrect* and *unlisted* recommendations. Thus, these activities can be impacted and justify the excess time of TIRT.

5.5.2 Qualitative Analysis

The questionnaire presented in Appendix A.4 were elaborated to gather qualitative information about the TIRT *usability* and *functionality*. These qualitative information are summarized as follows:

The eight (8) subject were unanimous in declaring that TIRT *traceability recommendation feature is useful for the core asset management activity*. Six (6) subjects reported that the *tree view component is really useful in order to verify the impact analysis*, on the other hand, two participants reported that using this type of component, a serious performance problem can arise as the database grows, because it loads all the related data field, forward and backward.

The subjects also reported that the the *details of traceability recommendations* presented were helpful to perform the analysis, however, two (2) subjects of them also justified that adding more fields, the reading of these data cold be impaired. Moreover, these participants also informed that the *user interface is clear and intuitive*.

5.6 Conclusion

The analysis performed in this experimet showed that TIRT had a lower performance *(spent time)* compared to SPLMT. The results also indicated that the evaluation about the *recall* metric did not provide good results. However the *precision* metric presents a significant result, helping the subjects in the traceability management. In this way, the descriptive analysis do not provide a concrete conclusion. Moreover, the experiment was applied in the context of one traceability scenario *(Use Case and Requirement)*. The remainder of traceability scenarios proposed also could be analysed. It did not happen because the time constraint to conduct the experiment, however, in in order to obtain more conclusive results, other experiments can be performed.

As analysed in the 5.5.1, only one of three null hypothesis was rejected. However, the qualitative analysis showed that TIRT has many favorable aspects to be taken into consideration when selecting which tool should be used. The subjects reported the main following contributions and improvements: *(a)* the TIRT can contribute to impact analysis with the tree view component; *(b)* the TIRT recommendation of core asset traceability can contribute to the maintenance of core assets in a SPL project.

5.7 Chapter Summary

This Chapter presented the experimental studies conducted in order to evaluate the TIRT tool. This experimental study was based on the proposed process in (Wohlin *et al.*, 2000), divided into the following main activities: **the definition**, **planning**, **operation** and **Analysis and Interpretation**.

The experiment was conducted with 10 subjects, however, two of them *(Subjects ID #2 and #7)* did not participated of treatments and qualitative questionnaire, affecting the data validation. In this way, we had 8 subjects for statistical and quality analysis. The results of the experiment pointed out that the null hypothesis of *effort* has not been rejected, as well as the *recall* metric. On the other hand, the *precision* metric presented favorable results.

Although the quantitative analysis showed that only one hypothesis of three was rejected, the subjects reported in the qualitative analysis that *traceability recommendation feature is useful for the core asset management activity*. In addition, subjects reported that the *tree view component is really useful in order to verify the impact analysis*.

Next Chapter will present the conclusions of this work, its main contributions and directions for future work.

6 Conclusion

Sonho que se sonha só
É só um sonho que se sonha só
Mas sonho que se sonha junto é realidade.

Dream you dream alone
It is just a dream you dream alone
But dream you dream together is reality.
—RAUL SEIXAS (Musician)

Software reuse is an important aspect to minimize costs and time-to-market, and maximize quality and productivity (Clements and Northrop, 2001). In this sence, Software Product Lines Engineering 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. Many reports document the significant achievements and experience gained by introducing software product lines in the software industry (Pohl *et al.*, 2005). Chapter 2 summarized the basic concepts about software product lines and their aspects, such as: SPL motivations, the related benefits from its use and some strategies used in the SPL adoption.

Regarding to Traceability, there are difficults to be addressed by the SPL approaches. Chapter 3 presented the main concepts about the Traceability area, Traceability in Software Product Lines and Impact Analysis. In the context of Software Product Lines Engineering, software artefact traceabily is an important factor when it comes to effective development and maintenance of software system. Traceability management facilitates the SPL artefacts to remain in synchronous state and ensure the consistency of derived products (Abid, 2004).

From this scenario, Chapter 4 described the tool and the proposal for traceability recommendation, that outline the recommendation scenarios of traceability, described in order to facilitate the maintenance activities of SPL. This recommendation approach assist support the decision making, supporting representation of dependencies between different core assets.

6.1 Research Contribution

The main contributions of this work can be summarized according to the following aspects: *(a) state of the art* of Traceability in Software Product Lines, emphasizing the traceability approaches and key challenges; *(b)* a *tool*, called TIRT, was proposed in order to mitigate the maintenance traceability problem; *(c)* the standardization of the vocabulary; *(d)* finally, it was performed an *experimental study* with 10 subject to evaluate the proposed tool.

- **State of the art and Prelimiray Mappy Study on Traceability for SPL:** I presented an study and identification of the main work about Traceability in the context of Software Product Lines, emphasizing the key challenges. This study was conducted based on some good practices of Systematic Mapping Studies proposed by Petersen *et al.* (2007), several related and important work were selected.

- **TIRT:** A prototype tool to support the traceability scenarios was proposed. It must be involved, including other techniques, features, and core assets. TIRT's goal is to facilitate the creation and mainly the maintenance activities regarding to the traceability relationship among the different core assets in a Software Product Lines projects. The prototype tool supports the traceability recommendation proposal in the context of Software Product Lines. Recommendation scenarios are likely occurrences of updates, which were mapped their impacts and correlations. The recommendation scenarios of traceability are described in order to facilitate the maintenance activities of SPL. This recommendation approach can aid the decision making, supporting representation of dependencies between different core assets.

- **Vocabulary:** The proposal of some suggestions to the fields registration. It aims the improvement of traceability recommendation between the core asset artefacts.

- **Experimental Study:** An experiment conducted in order to evaluate the TIRT tool was based on the proposed process defined in (Wohlin *et al.*, 2000), which is composed of the following main activities: *the definition*, *planning*, *operation* and *conclusion*. However, the quantitative analysis showed that only one hypothesis of three was rejected, users reported satisfactory results using the tool.

6.2 Future Work

Due to the time constraints imposed on the master degree, this work can be seen as an initial climbing towards the efficient, usable and effective traceability management in the context of Software Product Lines. Thus, there are interesting topics to improve what was started, and new paths to explore. In this way, the following issues should be investigated as future work:

- **Systematic Mapping Study:** A complete Systematic Mapping Studies about Traceability in Software Product Lines in order to identify more relevant work and build a classification scheme and structure this area.

- **Improvements of the prototype:** In order to optimize the TIRT, enhancements and new features must be implemented, as well as some problems must be fixed. These improvements and some defects reported by users are detailed as follows: *(a)* the recommendation system of tool must *support the Portuguese language*, because in the current version, the *stop-word removal* and *streamming* algorithms implemented in TIRT only considers the English language. *(b)* the tool should enabling the option of *enable or not the impact analysis component*, because some users related that performance problems can arise as the database grows, because it loads all the related data field, foward and backward. *(c)* the users also reported the implementation of new differents *visualization components*, in order to facilitate the software traceability and impact analysis. In addition to the improvements reported by users, it is also important to note that the tool will need to be adapted, according to the evolution and adaptation of the metamodel.

- **Automation of Traceability:** Another area is the automation and optimization of the generation of traceability relations in Software Product Lines, because we believe that this automation can bring considerable gains related to time and maintainability.

- **Analysis of vocabulary:** A tool to analyze the consistency of adoption of recommended vocabulary.

- **Extension of the Experiment:** Since the experiment performed in this dissertation was applied in the context of one traceability scenario *(Use Case and Requirement)*, thus it is necessary to perform a more elaborated experimental study, applying the other proposed scenarios. It did not happen in this current version, since the limitation of time, but we believe that the extension of the experiment could provide more conclusive results. In this way, the experiment should involve more subjects.

Bibliography

Abid, S. B. (2004). Resolving Traceability Issues In Product Derivation For Software Product Lines. *4th International Conference for Software and Data Technologies*, pages 99–104.

Abma, B. (2009). *Evaluation of requirements management tools with support for traceability-based change impact analysis*. Master's thesis, University of Twente, Enschede, The Netherlands.

Alexander, I. (2002). Towards Automatic Traceability in Industrial Practice. In *In Proc. of the 1st Int. Workshop on Traceability*, pages 26–31.

Almeida, E. S., Alvaro, A., Lucrédio, D., Garcia, V. C., and Meira, S. R. L. (2004). Rise Project: Towards a Robust Framework for Software Reuse. In *IEEE International Conference on Information Reuse and Integration (IRI)*, pages 48–53, Las Vegas, NV, USA.

Almeida, E. S., Alvaro, A., Garcia, V. C., Jorge, Burégio, V. A., Nascimento, L. M., Lucrédio, D., and Silvio (2007). *C.R.U.I.S.E: Component Reuse in Software Engineering*. C.E.S.A.R e-book, Recife, 1st edition.

Alvaro, A. (2009). *A Software Component Quality Framework*. Ph.d. thesis, CIn - Informatics Center, UFPE - Federal University of Pernambuco, Recife-PE, Brazil.

AMPLE (2011). Project AMPLE: Aspect-Oriented, Model-Driven Product Line Engineering. http://ample.holos.pt/.

Anquetil, N., Grammel, B., Galvao Lourenco da Silva, I., Noppen, J. A. R., Shakil Khan, S., Arboleda, H., Rashid, A., and Garcia, A. (2008). Traceability for Model Driven, Software Product Line Engineering. In *ECMDA Traceability Workshop Proceedings, Berlin, Germany*, pages 77–86, Norway. SINTEF.

Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., and Merlo, E. (2002). Recovering Traceability Links between Code and Documentation. *IEEE Transactions on Software Engineering*, **28**, 970–983.

Arnold, R. S. and Bohner, S. (1996). *Software Change Impact Analysis*. IEEE Computer Society Press, Los Alamitos, CA, USA.

Asuncion, H. U. (2008). Towards Practical Software Traceability. In *Companion of the 30th international conference on Software engineering*, ICSE Companion '08, pages 1023–1026, New York, NY, USA. ACM.

Asuncion, H. U., François, F., and Taylor, R. N. (2007). An end-to-end industrial software traceability tool. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ESEC-FSE '07, pages 115–124, New York, NY, USA. ACM.

Balbino, M. (2010). *RiPLE-SC: An Agile Scoping Process for Software Product Lines*. Master's thesis, Federal University of Pernambuco, Recife, Pernambuco, Brazil.

Basili, V., Caldiera, G., and Rombach, D. H. (1994). The Goal Question Metric Approach. In J. Marciniak, editor, *Encyclopedia of Software Engineering*. Wiley.

Basili, V. R. and Rombach, H. D. (1991). Support for Comprehensive Reuse. *Software Engineering Journal, Special issue on software process and its support*, **6**(5), 303–316.

Basili, V. R., Briand, L. C., and Melo, W. L. (1996). How Reuse Influences Productivity in Object-Oriented Systems. *Communications of the ACM*, **39**(10), 104–116.

Bayer, J. and Widen, T. (2002). Introducing Traceability to Product Lines. In *Revised Papers from the 4th International Workshop on Software Product-Family Engineering*, PFE '01, pages 409–416, London, UK. Springer-Verlag.

Bennett, K. H. and Rajlich, V. T. (2000). Software maintenance and evolution: a Roadmap. In *Proceedings of the Conference on The Future of Software Engineering (ICSE'00)*, pages 73–87, New York, NY, USA. ACM Press.

Berg, K., Bishop, J., and Muthig, D. (2005). Tracing Software Product Line Variability: from Problem to SSpace. In *Proceedings of the 2005 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*, SAICSIT '05, pages 182–191, Republic of South Africa. South African Institute for Computer Scientists and Information Technologists.

Birk, A. and Heller, G. (2007). Challenges for requirements engineering and management in software product line development. In *Proceedings of the 13th international working conference on Requirements engineering: foundation for software quality*, REFSQ'07, pages 300–305, Berlin, Heidelberg. Springer-Verlag.

Booch, G., Rumbaugh, J., and Jacobson, I. (2005). *The Unified Modeling Language*. Addison-Wesley, 2nd edition.

Brito, K. (2007). *LIFT: A Legacy InFormation retrieval Tool*. Master's thesis, CIn - Informatics Center, UFPE - Federal University of Pernambuco, Recife, Pernambuco, Brazil.

Cavalcanti, R. (2010). *Extending the RiPLE-Design Process with Quality Attribute Variability Realization*. M.sc. dissertation, CIn - Informatics Center, UFPE - Federal University of Pernambuco, Recife-PE, Brazil.

Cavalcanti, Y. a. C., do Carmo Machado, I., da Mota, P. A., Neto, S., Lobato, L. L., de Almeida, E. S., and de Lemos Meira, S. R. (2011). Towards metamodel support for variability and traceability in software product lines. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '11, pages 49–57, New York, NY, USA. ACM.

Cavalcanti, Y. C. (2009). *A Bug Report Analysis and Search Tool*. M.sc. dissertation, CIn - Informatics Center, UFPE - Federal University of Pernambuco, Recife-PE, Brazil.

Chastek, G. J., Donohoe, P., and McGregor, J. D. (2009). Formulation of a Production Strategy for a Software Product Line. Technical report, Software Engineering Institute.

Cleland-Huang, J. (2006). Just Enough Requirements Traceability. In *Proceedings of the 30th Annual International Computer Software and Applications Conference - Volume 01*, pages 41–42, Washington, DC, USA. IEEE Computer Society.

Cleland-Huang, J. and Habrat, R. (2007). Visual Support In Automated Tracing. In *Proceedings of the Second International Workshop on Requirements Engineering Visualization*, REV 2007, pages 4–9, Washington, DC, USA. IEEE Computer Society.

Clements, P. and Northrop, L. (2001). *Software Product Lines: Practices and Patterns*, volume 0201703327. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Cunha, C. (2009). *A Visual Bug Report Analysis and Search Tool*. M.sc. dissertation, CIn - Informatics Center, UFPE - Federal University of Pernambuco, Recife-PE, Brazil.

Davis, S. M. (1987). *Future Perfect*. Addison-Wesley, Boston, Massachusetts.

Demoiselle (2011). Demoiselle Framework. http://www.frameworkdemoiselle.gov.br/.

Dick, J. (2002). Rich Traceability. In *Proceedings of the 1st International Workshop on Traceability for Emerging forms of Software Engineering*, page 2005, Edinburgh, UK.

Diehl, S. (2007). *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Springer, Berlin.

Durao, F. (2008). *Semantic Layer Applied to a Source Code Search Engine*. Master's thesis, Federal University of Pernambuco, Recife, Pernambuco, Brazil.

Egyed, A. (2003). A Scenario-Driven Approach to Trace Dependency Analysis. *IEEE Transactions on Software Engineering*, **29**, 116–132.

Eriksson, M., Börstler, J., and Borg, K. (2005). The PLUSS Approach Domain Modeling with Features, Use Cases and Use Case Realizations. In *In Proceedings of the 9th International Software Product Line Conference*, pages 33–44.

Feldman, R. and Sanger, J. (2007). *The Text Mining Handbook: advanced approaches in analyzing unstructured data*. Cambridge University Press.

Filho, E. D. S., Cavalcanti, R. O., Neiva, D. F. S., Oliveira, T. H. B., Lisboa, L. B., Almeida, E. S., and Meira, S. R. L. (2008). Evaluating Domain Design Approaches Using Systematic Review. In R. Morrison, D. Balasubramaniam, and K. E. Falkner, editors, *2nd European Conference on Software Architecture (ECSA'08)*, volume 5292 of *Lecture Notes in Computer Science*, pages 50–65. Springer.

Frakes, W. B. and Isoda, S. (1994). Success Factors of Systematic Software Reuse. *IEEE Software*, **11**(01), 14–19.

Frank J. van der Linder, E. R. and Schmid, K. (2007). *Software Product Lines in Action*. Springer, New York, USA.

Galvão, I., Shakil, S., Nopper, J., Rummler, A., and Sánchez, P. (2008). *Definition of a traceability framework (including the metamodel and the modelling of processes and artefacts to allow traceability in the presence of uncertainty) for SPLs*. Master's thesis.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. M. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Boston, MA, USA.

Garcia, V. C. (2010). *RiSE Reference Model for Software Reuse Adoption in Brazilian Companies*. Ph.d. thesis, CIn - Informatics Center, UFPE - Federal University of Pernambuco, Recife-PE, Brazil.

Gotel, O. and Finkelstein, A. (1995). Contribution Structures. In *Proceedings of 2nd International Symposium on Requirements Engineering, (RE '95*, pages 100–107. Society Press.

Griss, M. L., Favaro, J., and Alessandro, M. d. (1998). Integrating Feature Modeling with the RSEB. In *Proceedings of the 5th International Conference on Software Reuse*, ICSR '98, pages 76–, Washington, DC, USA. IEEE Computer Society.

Gupta, N. K., Gupta, N. K., Jagadeesan, L. J., Jagadeesan, L. J., Koutsofios, E. E., Koutsofios, E. E., Weiss, D. M., and Weiss, D. M. (1997). Auditdraw: Generating Audits the FAST Way. In *In Proceedings of the 3rd IEEE International Symposium on Requirements Engineering*, pages 188–197, Washington, DC, USA.

Hayes, J. H., Dekhtyar, A., and Osborne, J. (2003). Improving requirements tracing via information retrieval. *Information Retrieval*, pages 138–150.

Helferich, A., Schmid, K., and Herzwurm, G. (2006). Reconciling Marketed and Engineered Software Product Lines. *Software Product Line Conference, International*, **0**, 23.

IEEE (1990). IEEE Std.610.12-1990. Standard Glossary of Software Engineering Terminology. http://standards.ieee.org/findstds/standard/610.12-1990.html.

IEEE (1998). IEEE Std 830-1998 - IEEE Recommended Practice for Software Requirements Specifications. http://standards.ieee.org/findstds/standard/830-1998.html.

Jamwal, D. (2010). Software Reuse: A Systematic Review. In *Proceedings of the 4th National Conference*.

Jirapanthong, W. and Zisman, A. (2005). Supporting Product Line Development through Traceability. *12th Asia-Pacific Software Engineering Conference (APSEC'05)*, pages 506–514.

Jirapanthong, W. and Zisman, A. (2009). XTraQue: Traceability for Product Line Systems. *Software and Systems Modeling*, **8**, 117–144. 10.1007/s10270-007-0066-8.

Kang, K., Cohen, S., Hess, J., Nowak, W., and Peterson, S. (1990a). Feature-Oriented Domain Analysis (FODA) Feasibility Study. *Technical Report CMU/SEI-90-TR-21*.

Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., and Peterson, A. S. (1990b). Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Carnegie-Mellon University Software Engineering Institute.

Kaplan, R. M. (2005). *Inquiries into Words, Constraints and Contexts*. Stanford Universit, Stanford, CA.

Kim, S. D., Chang, S. H., and La, H. J. (2005). Traceability Map: Foundations to Automate for Product Line Engineering. *Software Engineering Research, Management and Applications, ACIS International Conference on*, **0**, 340–347.

Kitchenham, B. and Charters, S. (2007). Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical report, Keele University and Durham University Joint Report.

Kotonya, G. and Sommerville, I. (1998). *Requirements Engineering - Processes and Techniques*. John Wiley & Sons Ltd.

Kowalski, G. and Maybury, M., editors (2002). *Information Storage and Retrieval Systems*. Kluwer Academic Publishers.

Krueger, C. W. (1992). Software Reuse. *ACM Comput. Surv.*, **24**, 131–183.

Krueger, C. W. (2002). Easing the Transition to Software Mass Customization. In *Revised Papers from the 4th International Workshop on Software Product-Family Engineering*, PFE '01, pages 282–293, London, UK. Springer-Verlag.

Lancaster, F. W. (1986). *Vocabulary Control for Information Retrieval*. Information Resources Press, 2 edition.

Larman, C. (2004). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Lientz, B. P. and Swanson, E. B. (1980). *Software Maintenance Management*. Addison-Wesley, Boston, MA, USA.

Lindvall, M. and Sandahl, K. (1996). Practical implications of traceability. *Software Practice and Experience*, **26**, 1161–1180.

Lisboa, L. (2008). *ToolDAy - A Tool for Domain Analysis*. Master's thesis, CIn - Informatics Center, UFPE - Federal University of Pernambuco, Recife, Pernambuco, Brazil.

Lucia, A. D., Fasano, F., Oliveto, R., and Tortora, G. (2007). Recovering traceability links in software artifact management systems using information retrieval methods. *ACM Transactions on Software Engineering and Methodology*, **16**.

Machado, I. (2010). *RiPLE-TE : A Software Product Lines Testing Process*. Master's thesis, CIn - Informatics Center, UFPE - Federal University of Pernambuco, Recife, Pernambuco, Brazil.

Martins, A. C., Garcia, V. C., Almeida, E. S., and Meira, S. R. L. (2008). Enhancing components search in a reuse environment using discovered knowledge techniques. In *2nd Brazilian Symposium on Software Components, Architectures, and Reuse (SBCARS'08)*, Porto Alegre, Brazil.

Mascena, J. C. C. P. (2006). *ADMIRE: Asset Development Metric-based Integrated Reuse Environment*. Master's thesis, CIn - Informatics Center, UFPE - Federal University of Pernambuco, Recife, Pernambuco, Brazil.

McGregor, J. D. (2003). The Evolution of Product Line Assets. Technical report, Software Engineering Institute.

McGregor, J. D., Northrop, L. M., Jarrad, S., and Pohl, K. (2002). Initiating Software Product Lines. *IEEE Software*, **19**(4), 24–27.

McIlroy, D. (1968). Mass-Produced Software Components. In *Proceedings of the 1st International Conference on Software Engineering, Garmisch Pattenkirchen, Germany*, pages 88–98.

Medeiros, F. M. (2010). *SOPLE-DE: An Approach to Design Service-Oriented Product Line Architecture*. M.sc. dissertation, CIn - Informatics Center, UFPE - Federal University of Pernambuco, Recife-PE, Brazil.

Mendes, R. C. (2008). *Search and Retrieval of Reusable Source Code using Faceted Classification Approach*. Master's thesis, CIn - Informatics Center, UFPE - Federal University of Pernambuco, Recife, Pernambuco, Brazil.

Mens, T. and Demeyer, S. (2008). *Software Evolution*. Springer, Springer.

Moon, M., Chae, H. S., Nam, T., and Yeom, K. (2007). A Metamodeling Approach to Tracing Variability between Requirements and Architecture in Software Product Lines. *Computer and Information Technology, International Conference on*, **0**, 927–933.

Moreton, R. (1996). A Process Model for Software Maintenance Software. Change Impacty Analysis. *IEEE Computer Society*.

Nascimento, L. (2008). *Core Assets Development in SPL - Towards a Practical Approach for the Mobile Game Domain*. Master's thesis, Federal University of Pernambuco, Recife, Pernambuco, Brazil.

Neiva, D. (2008). *RiPLE-RE : A Requirements Engineering Process for Software Product Lines*. Master's thesis, CIn - Informatics Center, UFPE - Federal University of Pernambuco, Recife, Pernambuco, Brazil.

Neto, P. (2010). *A Regression Testing Approach for Software Product Lines Architectures*. M.sc. dissertation, CIn - Informatics Center, UFPE - Federal University of Pernambuco, Recife-PE, Brazil.

Nielsen, J. (2011). Ten Usability Heuristics. http://www.useit.com/papers/heuristic/heuristic_list.html/.

Oliveira, T. B. (2009). *RiPLE-EM : A Process to Manage Evolution in Software Product Lines*. Master's thesis, Federal University of Pernambuco, Recife, Pernambuco, Brazil.

Petersen, K., Feldt, R., Mujtaba, S., and Mattsson, M. (2007). Systematic Mapping Studies in Software Engineering. *12th International Conference on Evaluation and Assessment in Software Engineering*, **17**(1), 1–10.

Pfleeger, S. L. (2001). *Software Engineering: Theory and Practice*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition.

Pinheiro, F. A. C. and Goguen, J. A. (1996). An Object-Oriented Tool for Tracing Requirements. *IEEE Software*, **13**, 52–64.

Pohl, K. (1996a). PRO-ART: Enabling Requirements Pre-Traceability. *IEEE International Conference on Requirements Engineering*, **0**, 76.

Pohl, K. (1996b). *Process-Centered Requirements Engineering*. John Wiley & Sons, Inc., New York, NY, USA.

Pohl, K., Böckle, G., and van der Linden, F. (2005). *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer.

Porter, M. F. (1980). An algorithm for suffix stripping. *Program*, **3**(14), 130–137.

Poulin, J. S. (1997). *Measuring Software Reuse - Principles, Practices, and Economic Models*. Assison-Wesley, Boston, MA, USA.

Pussinen, M. (2002). A survey on software product-line evolution. Technical report, Institute of Software Systems.

R. Cooper, S. E. and Kleinschmidt, E. (2001). *Portfolio Management for new Products*. Perseus Publishing, Philadelphia, USA.

Ramesh, B. and Jarke, M. (2001). Toward Reference Models for Requirements Traceability. *IEEE Transactions on Software Engineering*, **27**, 58–93.

Ramesh, B., Powers, T., Stubbs, C., and Edwards, M. (1995). Implementing requirements traceability: a case study. In *Proceedings of the Second IEEE International Symposium on Requirements Engineering*, RE '95, pages 89–95, Washington, DC, USA. IEEE Computer Society.

Ribeiro, H. B. (2010). *An Approach to Implement Core Assets in Service-Oriented Product Lines*. M.sc. dissertation, CIn - Informatics Center, UFPE - Federal University of Pernambuco, Recife-PE, Brazil.

Royce, W. W. (1987). Managing the development of large software systems: concepts and techniques. In *Proceedings of the 9th international conference on Software Engineering*, ICSE '87, pages 328–338, Los Alamitos, CA, USA. IEEE Computer Society Press.

Salton, G., Wong, A., and Yang, C. S. (1975). A Vector Space Model for Automatic Indexing. *Commun. ACM*, **18**(11), 613–620.

Santos, E. C. R., ao, F. A. D., Martins, A. C., Mendes, R., Melo, C., Garcia, V. C., Almeida, E. S., and Meira, S. R. L. (2006). Towards an effective context-aware proactive asset search and retrieval tool. In *6th Workshop on Component-Based Development (WDBC'06)*, pages 105–112, Recife, Pernambuco, Brazil.

Sherba, S. A., Anderson, K. M., and Faisal, M. (2003). A Framework for Mapping Traceability Relationships. In *2 nd International Workshop on Traceability in Emerging*

Forms of Software Engineering at 18th IEEE International Conference on Automated Software Engineering, pages 32–39.

Sommerville, I. (2007). *Software Engineering*. Addison Wesley, 8 edition.

Sousa, A., Kulesza, U., Rummler, A., Anquetil, N., Fct, C. D. I., Lisboa, U. N. D., and Darmstadt, T. U. (2009). A Model-Driven Traceability Framework to Software Product Line Development. *Software Systems Modeling*, **9**, 427–451–451.

Spanoudakis, G. and Zisman, A. (2004). Software Traceability: A Roadmap. In *Handbook of Software Engineering and Knowledge Engineering*, pages 395–428. World Scientific Publishing.

Streitferdt, D. (2001). Traceability for System Families. In *Proceedings of the 23rd International Conference on Software Engineering*, ICSE '01, pages 803–804, Washington, DC, USA. IEEE Computer Society.

Strzalkowski, T. (1994). Robust text processing in automated information retrieval. In *Proceedings of the fourth conference on Applied natural language processing*, ANLC '94, pages 168–173, Stroudsburg, PA, USA. Association for Computational Linguistics.

TechTarget (2004). SearchCIO.com Definitions, whatis.techtarget.com. http://www.whatis.techtarget.com. Last acess on May/2011.

Tiboni, A. C., da Silva Lisboa, F. G., and Mota, L. C. (2009). Uma plataforma livre para padronização do desenvolvimento de sistemas no Governo Federal. *COLIBRI - Colóquio de informática - Brasil - INRIA*.

Tracz, W. (1988). Software reuse: emerging technology. *IEEE Software*, pages 62–67.

Vanderlei, T. A., ao, F. A. D., Martins, A. C., Garcia, V. C., Almeida, E. S., and Meira, S. R. L. (2007). A cooperative classification mechanism for search and retrieval software components. In *Proceedings of the 2007 ACM symposium on Applied computing (SAC'07)*, pages 866–871, New York, NY, USA. ACM.

von Knethen, A. and Paech, B. (2002). A Survey on Tracing Approaches in Practice and Research. *Fraunhofer IESE*, (095), 1–49.

Wohlin, C., Runeson, P., Martin Höst, M. C. O., Regnell, B., and Wesslén, A. (2000). *Experimentation in Software Engineering: An Introduction*. The Kluwer Internation

Series in Software Engineering. Kluwer Academic Publishers, Norwell, Massachusets, USA.

Yau, S. S., Collofello, J. S., and MacGregor, T. M. (1993). Software engineering metrics i. In M. Shepperd, editor, *Software engineering metrics I*, pages 71–82, New York, NY, USA. McGraw-Hill, Inc.

Zhou, X., Huo, Z., Huang, Y., and Xu, J. (2008). Facilitating Software Traceability Understanding with ENVISION. *Computer Software and Applications Conference, Annual International*, **0**, 295–302.

Zisman, A., Spanoudakis, G., mi nana, E. P., and Krause, P. (2003). Tracing Software Requirements Artefacts. In *In: The 2003 International Conference on Software Engineering Research and Practice (SERP'03). 2003. Las Vegas*, pages 448–455.

Appendices

A
Experiment Instruments

A.1 Time sheet of Core Assets Management with TIRT

ID	*Start time*	*End time*	*Total∗*	*Correct∗*	*Incorrect∗*	*Unlisted∗*
1	:	:				
2	:	:				
3	:	:				
4	:	:				
5	:	:				
6	:	:				
7	:	:				
8	:	:				
9	:	:				
10	:	:				

Table A.1 Time sheet used with the TIRT tool.

**Indicators about the core assets recommendation.*

A.2 Time sheet of Core Assets Management with SPLMT

ID	*Start time*	*End time*
1	:	:
2	:	:
3	:	:
4	:	:
5	:	:
6	:	:
7	:	:
8	:	:
9	:	:
10	:	:

Table A.2 Time sheet used with the SPLMT tool.

A.3 Questionnaire for Subjects Profile

Questionnaire for Subjects Profile
How many years since graduation? [ ] years.
How many projects do you have participated according to the following categories? [ ] Low complexity. [ ] Medium complexity. [ ] High complexity.
In case you have already participated in projects cited before, what was your role? Cite the number of times you played each role.
How do you define your experience in Software Reuse? [ ] None. [ ] Low. [ ] Medium. [ ] High.
How do you define your experience in Software Product Lines? [ ] None. [ ] Low. [ ] Medium. [ ] High.
Have you used any CASE tool to core assets management? [ ] Yes. [ ] No.

Table A.3 Questionnaire for Subjects Profile.

A.4 Form for Qualitative Analysis

Questionnaire for Qualitative Analysis
Was the traceability recommendation useful for the core asset management activity? [ ] Yes. [ ] No.
Was the tree view component useful in order to verify the impact analysis? [ ] Yes. [ ] No.
Did you have any problem with the traceability recommendation? [ ] Yes. [ ] No. Cite them:
Do you believe the details of traceability recommendations presented were helpful to perform the analysis? [ ] Yes. [ ] No.
Do you think there is any other important information that must be present in the list of traceability recommendation? [ ] Yes. [ ] No. Cite them:
Did you use the help provided by TIRT? [ ] Yes. [ ] No.
In your opinion, TIRT provides a clean and useful interface?
Did you found any other problems that were not mentioned before? Cite them.
In your opinion, what are the strengths of the TIRT tool?
Please, write down any suggestion you think might would be useful for TIRT.

Table A.4 Questionnaire for Qualitative Analysis.