Mestrado Multiinstitucional de Pós-Graduação em Ciência da Computação - MMCC

NON-FUNCTIONAL PROPERTIES IN SOFTWARE PRODUCT LINES: A REUSE APPROACH

by

Larissa Rocha Soares

M.Sc. Dissertation

SALVADOR
October/2014

LARISSA ROCHA SOARES

NON-FUNCTIONAL PROPERTIES IN SOFTWARE PRODUCT LINES: A REUSE APPROACH

M.Sc. Dissertation presented to the Multi-institutional Master Programme in Computer Science at Federal University of Bahia and Feira de Santana State University in partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

Advisor: *Eduardo Santana de Almeida*

SALVADOR
October/2014

Aos meus pais, Lourival e Adenilde, por todo o amor incondicional, e a Igor, fonte de inspiração e paz.

Agradecimentos

É bem difícil começar aqui sem agradecer primeiro a Deus! Eu sinto a proteção Dele por onde vou, por onde passo. Sinto-me abençoada sim! E tenho muito a agradecer. Obrigada, Pai, pela orientação que me tem dado, por essa conquista, pelo dia de hoje, pela família que me deste e, principalmente, pela Tua presença em minha vida.

Agradeço aos meus pais, Lourival e Adenilde, pelo amor incondicional, pela compreensão, carinho, paciência e principalmente por acreditar em mim, mais até do que eu mesma. Pois, como diz a música, se Deus me desse a chance de viver outra vez, eu só queria se tivesse vocês! Obrigada, meu Deus, por ter escolhido esses anjos para mim.

Agradeço a meu irmão, Wagner e aos seus filhos, Bia e Davi. Obrigada meu irmão por ser esta pessoa em quem me espelho tanto. Sua integridade é invejável, não há como olhar para ti e não sentir a bondade em teus olhos. Seus filhos puxaram isso de ti. Agradeço a Bia que há 5 anos faz da minha vida mais feliz, com sua voz doce e seu sorriso encantador. Agradeço a Davi, que com apenas 15 dias de vida é capaz de me fazer sentir o mais puro amor só em abrir seus lindos olhos pretos. A tia/dinda ama muito vocês.

Agradeço a Igor. O que sinto por você, meu pig, é inexplicável. Obrigada por se doar tanto e me permitir te fazer feliz. Se consegui chegar até aqui, você também merece os méritos. Já que esteve comigo me apoiando em cada passo que dei. Obrigada, meu amor! Ter você ao meu lado é também um presente de Deus.

Agradeço aos meus avós maternos. Mesmo sem tê-los mais aqui comigo, a princesa, como eu era chamada, sente o quanto torcem por mim. Obrigada! Aos meus avós paternos, obrigada pela força e carinho, e também por me incentivarem a continuar na luta.

Agradeço aos meus padrinhos pela dedicação e tanto carinho e confiança depositados em mim. Dona Ana, eu sei o quanto oras por mim. Não tenho palavras para agradecer o tamanho do seu amor. Muito obrigada.

Agradeço aos meus tios e tias, primos e primas, principalmente a Eliene, Telma, Del e Geu! Sei o quanto torcem por mim. Obrigada por todas as palavras de carinho, ligações, mensagens e energias positivas sempre quando eu mais preciso. O amor é recíproco!

Agradeço ao professor/orientador Eduardo pela oportunidade que me deste e confiança de que poderíamos fazer um bom trabalho juntos. Obrigada por ter me guiado e me ajudado a construir esse trabalho.

Agradeço também ao professor Ivica, da universidade da Suécia, a qual passei 5 meses como pesquisadora durante o mestrado. Obrigada pela oportunidade em aprender tanto, pelo seu tempo, por cada reunião que tivemos, por transmitir a mim um pouco dos seus conhecimentos.

Obrigada aos amigos do Laboratório de Engenharia de Software - LES e RiSE, e a todos os amigos da UFBA incluindo o pessoal da secretaria de pós-graduação. Meus últimos anos foram mais prazerosos porque tive vocês, onde pude compartilhar tantas alegrias e tristezas também. Um agradecimento especial a Adriana e Loreno, juntos desde o início dessa jornada!

Obrigada aos amigos da UEFS, onde me formei engenheira de computação, e do colégio também, e aos outros que a vida e Deus me permitiram chamar de meus. Uma vida sem amigos é uma vida vazia, como já dizia Martha Medeiros.

E ao final, após escrever todas essas palavras, percebo o quanto tenho pessoas boas ao meu lado e o quanto eu não seria ninguém sem elas. Um muito obrigada a todos que contribuíram nessa jornada e a todos que ainda contribuem para que eu possa hoje ser quem sou. Obrigada.

"Deus é o que me cinge de força e aperfeiçoa o meu caminho"
Salmos 18:32

The future belongs to those who believe in the beauty of their dreams.

—ELLEANOR ROOSEVELT

Resumo

Reuse de software é um aspecto importante para organizações de software interessadas em produtos personalizados e a custos razoáveis. Engenharia de Linhas de Produtos de Software (SPLE) tem como objetivo alcançar estes desafios. O paradigma de SPLE é dividido em dois principais processos: engenharia de domínio e engenharia de aplicação. Derivação de Productos é a prática de criar produtos distintos durante a engenharia de aplicação.

Com base na seleção de características (features), engenheiros de SPL e interessados podem derivar programas feitos sob medida e de forma eficiente que satisfazem diferentes necessidades. Neste cenário, propriedades não-funcionais (NFPs) surgem de maneira a prover uma derivação de produtos não apenas em relação às características funcionais, mas também aos atributos de qualidade. Uma definição explícita de NFPs durante a configuração de software tem sido considerada uma tarefa difícil, uma vez que NFPs em grandes sistemas resultam da interação de muitos recursos, tornando-os difíceis de serem configurados.

SPL tem sido muito bem sucedida na gestão de features que compõem propriedades funcionais e também um grande número de NFPs. No entanto, existem muitas NFPs que não podem ser expressas e realizadas sob a forma de features, mas requerem diferentes abordagens. Como lidar com elas ainda é um desafio, tanto na teoria como na prática. Atualmente, poucos trabalhos se concentram na análise da NFPs para a engenharia de linha de produto de software.

Nesse sentido, realizamos uma revisão sistemática da literatura publicada em busca de abordagens de SPL que reportam NFPs. Além disso, propomos um framework para especificar NFPs para SPL e também uma abordagem de reuso, a qual promove a reutilização dos valores de NFPs durante a configuração de um produto. Uma vez que a engenharia de SPL promove a reutilização de artefatos de SPL, valores de NFPs também poderiam ser reutilizados. Além disso, estudos de caso foram realizados a fim de avaliar a aplicabilidade do framework e da abordagem de reuso.

Palavras-chave: Linhas de Produto de Software, Derivação de Produtos, Propriedades Não-functionais, Revisão Sistemática de Literatura.

Abstract

Software reuse is an important aspect for software organizations interested in customized products at reasonable costs. Software Product Line Engineering (SPLE) emerges in order to achieve such challenges. The SPLE paradigm is divided in two processes: domain engineering and application engineering. Product derivation is the practice to create distinct products during application engineering.

Based on the selection of features, stakeholders can efficiently derive tailor-made programs satisfying different requirements. In this scenario, non-functional properties (NFPs) arise to provide product derivation regarding not only functional features, but also quality attributes. The explicit definition of NFPs during software configuration has been considered a challenging task, since NFPs in large systems result from the interaction of many features, making them very hard to be configured.

SPL in practice has been very successful in managing features that comprise both functional properties and a large number of NFPs. However, there are many NFPs that cannot be expressed and then realized in form of features, but require different approaches. How to deal with them is still not well established, neither in theory nor in practice. To the best of our knowledge, little work has focused on the analysis of NFPs for SPL engineering.

In this sense, we carried out a systematic literature review of the published literature on SPL approaches reporting on NFPs. In addition, we proposed a framework to specify NFPs for SPL and also a reuse approach that promotes the reuse of NFPs values during the configuration of a product. Once SPL engineering promotes the reuse of SPL artifacts, NFPs values may be reused too, which can be computed by means of specialists, program execution, syntactic measures, predictions techniques, and so on. In addition, case studies were performed in order to evaluate the applicability of both reuse approach and framework.

Keywords: Software Product Lines, Product Derivation, Non-functional Properties, Systematic Literature Review.

Contents

List of Figures

List of Tables

List of Acronyms

FODA	Feature-Oriented Domain Analysis
SPL	Software Product Lines
FP	Functional Property
NFP	Non-Functional Property
SPLE	Software Product Lines Engineering
SLR	Systematic Literature Review
PD	Product Derivation
SCM	Software Configuration Management
GQM	Goal-Question-Metric
CC	Conditional Compilation
MVC	Model View Controller

1 Introduction

1.1 Motivation

Software Product Lines Engineering (SPL) practices have been proved to be widely applicable and a successful approach in both industry and academy. The key principles of SPL development, towards core assets development, product development and management, are playing an increasingly important role in software engineering (Clements and Northrop, 2001). SPL exploit the commonalities among products to achieve economies of scale by developing core assets, entities that will be reused in multiple product instances in the product line, such as requirements, architectures, components, and test cases (Clements and Northrop, 2001).

Through the reusability of a set of assets, SPL engineering allows creating a large number of related products, which can be assembled together to satisfy the demands of a particular domain. Due to its economic advantages over single-system software development, SPL engineering has gained an interest by software companies. These companies look for strategies to handle an increased market demand for better products, delivered at a reduced time and with lower production cost.

SPL engineering encompasses two main processes (Pohl *et al.*, 2005): *domain engineering* and *application engineering*. The former aims at establishing a reusable platform, by identifying common and variable features of products that compose a product line, and thus creating effective core assets. The latter is responsible for binding the variability in the core assets, and deriving product line instances from the platform created in the former process.

SPL products are distinguished in terms of features, i.e., end-user visible characteristics of products (Kang *et al.*, 1990b). Based on the selection of features, stakeholders can derive tailor-made products satisfying a range of functional (FPs) and non-functional properties (NFPs).

FPs implement the tasks/functionalities of a software. On the other hand, NFPs are those that

impose special conditions and qualities on the system (Lohmann *et al.*, 2005), usually observable by end-users. For instance, if a software system runs slower than expected, users may not be interested in using it, regardless the provided functionalities. Functional and non-functional aspects should work synchronized for a product to be viable in the market.

NFPs play an important role in SPL engineering. In the SPL literature, those properties can be also referred to as *quality attributes* (Zhang *et al.*, 2003), *non-functional requirements* (Aoyama and Yoshino, 2008), *extra-functional properties* (Benavides *et al.*, 2005) and *softgoals* (Nguyen, 2009). In this work, we will interchangeably mention quality attributes (QAs) and NFPs.

With the advent of technology, more and more systems are being built for applications where non-functional properties, such as user feedback and hardware restrictions, are as important as functional properties. In this sense, the goal of this dissertation is threefold. First, we review the current state-of-the-art of Software Product Lines and Non-Functional Properties. Then, we present one way to specify NFPs for SPL. Finally, an approach that promotes the reuse of NFPs values during the configuration of a product is described.

1.2 Problem Statement

In contrast to conventional system development, an SPL typically covers a wide variety of application scenarios, environments and customers. Hence, a same SPL can have several different products with many types of NFPs. The explicit definition of NFPs during software configuration has been considered a challenging task (Sincero *et al.*, 2010, 2007). This definition may vary from author to author, with different ways of analysis and measurement process, which complicates to understand their meaning or even to reuse them in other contexts. Generally, the NFPs of a complex system are the result of the interaction of many features, which makes them very difficult to be configured (Sincero *et al.*, 2010).

In this way, this work is intended to further investigate the non-functional properties inside the software product lines context. We present a framework to specify and propose an approach to reuse NFPs values on SPL products. Once SPL engineering promotes the reuse of SPL artifacts, we believe that NFPs values may be reused too, which can be computed by means of specialists, product execution, syntactic measures, predictions techniques, and so on. In addition, a case study was also performed in order to evaluate the applicability of both framework and reuse approach. Finally, we conducted a replicated study under the same conditions of the first case, but using a different domain.

In summary, the main goal of this dissertation can be stated as follows:

> *This work investigates current literature of Software Product Lines, with an emphasis on product derivation process aware of non-functional properties. Furthermore, a framework is presented to better describe the characteristics of a quality attribute and its values. This framework supports a reuse approach where time consuming tasks related to the measurement of NFPs values do not have to be repeated for very similar subsystems.*

1.3 Related Work

Several research efforts have been devoted in the last years to the definition of approaches to deal with NFPs. The following studies were considered related for having similar ideas to this work. They are presented in two categories: (i) literature reviews, and (ii) specification and reuse of NFPs information.

1.3.1 Literature Reviews

The ability to evaluate quality attributes of SPL products involves a set of tasks or practices to perform that span from quality variability modeling through architecture evaluation by taking into account the variability to quality testing (Etxeberria *et al.*, 2008).

Several interesting literature reviews on SPL related aspects have been presented for different scopes. Surveys have been conducted, for example, in order to: (i) investigate how software reuse is adopted in SPLs (Jha and O'Brien, 2009; Lobato *et al.*, 2013), (ii) review the SPLs testing approaches (Neto *et al.*, 2011; Lee *et al.*, 2012), and (iii) assess the quality of the research in requirement engineering within SPL engineering (Alves *et al.*, 2010; Da Silva *et al.*, 2014).

However, to the best of our knowledge, only a few work, some of which are detailed next, are focused on SPLs quality analysis. Moreover, these surveys are often incomplete or focused on different research topics.

The work in Myllärniemi *et al.* (2012) is focused on the research problem of modeling variability in quality concerns for SPLs. Specifically, the primary studies at the Software Product Line Conference (SPLC) have been reviewed in order to understand why, how, and which quality attributes to vary. Other interesting systematic review was carried out on variability management in SPLs (Peng *et al.*, 2011), but they do not take quality aspects into account.

Montagud *et al.* (2012) presented the closest work to ours, where quality attributes and measures have been considered. Specifically, a catalog of measures for quality attributes found

on SPL lifecycle has been proposed. The study found 165 measures related to 97 different quality attributes. Their results indicated that 92% of the measures evaluate attributes that are related to maintainability, during the domain SPL phase. According to them, although their findings may be indicative of the field, further reviews are needed to confirm the results obtained. In this present dissertation, we elaborate on this work, through a systematic review, by investigating how researchers and practitioners are looking for improvements in the inclusion process of runtime NFPs in the SPL approach, mainly during the application SPL phase.

1.3.2 Specification and Reuse of NFPs Information

A framework for component-based embedded systems was initially proposed by Sentilles (2012), where NFPs can be attached to architectural entities of component models, such as ports, interfaces and connectors. Furthermore, concrete NFPs values can be compared and specified during the development process. Her work aimed to provide an efficient support, possibly automated, for analyzing selected properties. The work in this dissertation used the Sentilles (2012) study as a basis to propose a similar framework for SPL. Thus, an adaptation for the product-line context was proposed, in order to provide a way of properly specifying and documenting NFPs for the SPL development, specially on the product derivation process.

Cicchetti *et al.* (2011) and Baumgart *et al.* (2012) discussed about techniques to reuse an NFP value in a new context. On the one hand, Cicchetti *et al.* (2011) presented a mechanism focused on the evolution management of NFPs for component-based embedded systems. Their goal was to discover if an NFP value is still valid when components related to it evolve. Thus, they anticipate the impact analysis of the changes and detect modifications at the modeling level, providing corresponding validation responses.

On the other hand, Baumgart *et al.* (2012) proposed a way to reuse functional software certification for automotive domains. The safety certification process of a critical functionality in vehicles takes too much time of a safety-critical product development project. We elaborate on Baumgart *et al.* (2012) work, in order to discuss an NFPs reuse approach for SPL, so that a reuse of an NFP value can be performed, where time consuming tasks can be avoided.

1.4 Out of Scope

As non-functional properties in Software Product Lines is part of a broader context, a set of related aspects will be left out of this work scope. Thus, the following issues are not directly addressed by this work:

- **Single systems.** This work is concerned to investigate the analysis of NFPs on SPL engineering, though a systematic review and the specifications of a framework and a reuse approach of NFPs values. Our work is focused on SPL and we do not investigate NFPs analysis in single systems scenarios.
- **Other disciplines of the SPL product derivation process.** In this dissertation, we propose an approach for SPL that supports the product derivation (PD) process. During the derivation of a product, SPL engineers can observe quality aspects of features and/ products. Other activities in the PD process are not covered in this dissertation, such as: SPL development and test.
- **New methods to measure NFPs values.** The reuse approach proposed in this work discusses about the activities necessary to perform a product derivation process aware of NFPs, but the way how NFPs are measured/estimated/simulated are not covered by our work.
- **Development of tools to support NFPs measurement.** The aim of this work is to present a framework to specify NFPs and an approach that provides a way to reuse NFPs values. Our work does not focus on automating the product derivation process with new tools to measure NFPs. Thus, its is not discussed about tools or methods used for this purpose.

1.5 Statement of the Contributions

The main research contributions of this dissertation are described as follows:

- **Systematic Literature Review on non-functional properties in SPL.** A review was performed to obtain a holistic overview of SPL approaches that have been reported regarding the analysis of NFPs. A set of research questions was investigated which resulted on a classification of papers in three categories, each one with its own peculiarities: (i) predictions; (ii) estimations; and (iii) features selection-focused approaches.
- **NFPs Framework**. A framework was proposed to specify NFPs in the SPL context. The main goal is to systematically provide additional information about features and products inside the SPL process, more specifically, during the generation of a new product (PD process).
- **NFPs Reuse Approach.** Achieving an efficient reuse approach where time consuming tasks do not have to be repeated for very similar (sub)systems was the motivation to

propose the reuse approach. It presents how to configure SPL products aware of NFPs, and also shows how to reuse NFPs values previously analysed, avoiding unnecessary reanalysis.

- **Case study.** We performed an exploratory case study based on Runeson *et al.* (2012) guidelines. According to them, a case study is an empirical enquiry based on multiple sources of evidence to investigate one or a small number of instances of a contemporary software engineering phenomenon within its real-life context. Thus, through this study, we aimed at investigating the NFPs Reuse Approach applicability in a product-line of text editors, namely, Notepad SPL. As the reuse approach is based on the NFPs Framework, its applicability was also investigated.

- **Replicated case study.** We replicated the exploratory study using a different SPL. In this way, the framework and reuse approach was evaluated using a product line of emergency applications for the mobile domain, the RecueMe SPL. Replications are a way to understand how much context influences the results, which allow that generalizations regarding the research questions can be made (Runeson *et al.*, 2012).

In addition to the contributions mentioned, a paper presenting part of the findings of this dissertation was accepted for publication:

- Soares, L. R., Potena, P., Machado, I. C., Crnkovic, I., and Almeida, E. S. (2014). Analysis of non-functional properties in software product lines: a systematic review. In 40th IEEE EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), Verona, Italy.

Moreover, we are currently submitting other papers to report the remaining results.

1.6 Research Design

The research design approach defined for this work is showed on Figure 1.1. The first step was to investigate the software product line area. This informal study also included to understand how SPL engineers could assembly products taking into account non-functional properties. As a result, we could write out the second chapter with some foundations on these subjects.

From this initial study, many questions were considered. In order to answer them, we carried out a systematic literature review (second step), which provided an holistic overview of the

research field and common practices. This review allowed us to categorize the available evidence and trends in research.

With the review, we identified a gap that if performed could aid stakeholders on work with NFPs. Thus, we proposed a framework to specify NFPs for the SPL context (third step). This framework was the basis to our reuse approach of NFPs value (fourth step). This approach describes the activities required to configure SPL products were NFPs analysis may not be repeated for very similar (sub)systems.

In order to evaluate the applicability of the approach and also of the framework, we conducted two exploratory case studies (fifth step): the first was applied in a text editor SPL of a desktop domain, and the second was focused on a mobile context of emergency applications. We assessed the case studies results to provide the lessons learned and improvements on the reuse process.

Figure 1.1: Research Design

1.7 Dissertation structure

The remainder of this dissertation is organized as follows:

- **Chapter 2** reviews the main topics used throughout this work: Software Product Lines, Product Derivation and Non-Functional Properties.
- **Chapter 3** presents the Systematic Literature Review of the published literature on SPL approaches reporting on NFPs.
- **Chapter 4** describes in details the framework proposed to specify NFPs through five tasks.
- **Chapter 5** explains the NFPs Reuse Approach that aims to discuss the product derivation process aware of NFPs, where the reuse of NFPs values can be performed.
- **Chapter 6** discusses the case study to evaluate the applicability of the reuse approach. The case study protocol, research question, data collection, data analysis, and outcomes are described in details.

- **Chapter 7** describes a replicated case study using an SPL in a different domain, as well as a comparative analysis;
- **Chapter 8** provides the concluding remarks. It presents the main contributions and outline directions for future work.
- **Appendix A** presents the list of primary studies addressed in the Systematic Literature Review of NFPs for SPL engineering.
- **Appendix B** describes some details of the case study performed, such as the feature model and product model of the text editor SPL, besides case study interview questions.
- **Appendix C** provides details of the replicated case study performed, as for example, the feature model and product model of the RescueMe SPL.

2

Foundations on Software Product Lines and Non-Functional Properties

Software development organizations have increasingly been challenged to improve its engineering practice aiming to deliver products faster and cheaper. In this way, the software industry is even more adopting approaches that focus on a high degree of reuse, based on decoupled and cohesive modules (McGregor *et al.*, 2002). In order to achieve such challenges, several strategies have been investigated in the software development scenario. One of them, Software Product Lines Engineering (SPLE), is a paradigm to develop software applications (software intensive-systems and software products) using platforms and mass customization (Pohl *et al.*, 2005). Developing applications using platforms means to plan proactively for reuse, and building applications for mass customization is a large-scale production tailored to individual customers needs (Pohl *et al.*, 2005).

SPLE encompasses two main processes (Pohl *et al.*, 2005): *domain engineering* and *application engineering*. The former aims to establish a reusable platform and define the commonality and the variability of the product line, and the latter is responsible for deriving product line applications from the platform created in domain engineering, where the previously developed assets are assembled to compose a product. SPL products are distinguished in terms of features, i.e., end-user visible characteristics of products (Kang *et al.*, 1990b). Based on the selection of features, stakeholders can derive tailor-made products satisfying a range of functional (FPs) and non-functional properties (NFPs).

This Chapter presents a brief literature review on the Software Product Lines (SPL) area and also on NFPs. It is structured as follows: Section 2.1 contextualizes the main concepts and benefits of Software Product Lines. Section 2.2 describes the product derivation process and its activities. Finally, Section 2.3 presents some concepts about non-functional properties.

2.1 Software Product Lines

The perception of Software Product Lines (SPL) was firstly introduced by Dijkstra (1972) and Parnas (1976). Parnas referred to SPL as a collection of systems that share common characteristics, as a family of systems. He stated the importance of planning before developing a program family. For him, the first thing to think in SPL is the degree of importance of each characteristic so that the resulting program focuses on its purpose properly. Parnas also stated that the order in which the design decisions are made is important, and he suggests that an approach for software families should choose the degree of importance of each aspect and characteristic.

According to Clements and Northrop (2001), an SPL is defined as "a set of software intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way."

Pohl *et al.* (2005) also provided important contributions on SPL, exemplified through the automobiles domain. They claim that formerly the goods were handcrafted for individual customers, however, gradually the amount of people who could buy many kinds of products increased. In the domain of automobiles this led to Fords invention of the product line, allowing the mass customization.

The individual software and standard software (mass produced) have one main weakness. The former is expensive, and the latter lacks sufficient diversification (Pohl *et al.*, 2005). This way, the standard mass customization was not enough to customers. They wanted others kinds of cars, with others features or, for example, a better car than his neighbor. It was the beginning of *mass customization*, which is the production tailored to individual customers needs. In order to facilitate mass customization, artifacts used in different products have to be flexible enough to fit into different systems produced in the product line. According to Pohl *et al.* (2005), this flexibility is a precondition for mass customization and is called *variability*, in the software product line context.

Coupled with the concept of *mass customization*, emerged the concept of *platforms*. Many companies started to introduce common platforms, planning in advance which parts would be used in different types of cars (Pohl *et al.*, 2005). Combining these two concepts allows to bring products out in close accordance with customers wishes while provide to reuse a common base of technology. Pohl *et al.* (2005) define this combination of *Software Product Line Engineering*.

2.1.1 SPL Motivation and Benefits

In many organizations, there is a desire to change from single systems (systems built for a specific purpose according to a single users requirements) to product lines in software engineering (Linden *et al.*, 2007). Increasingly organizations are looking for SPL as a way to achieve reuse, in order to obtain benefits that could help in many problems involving software development (Clements and Northrop, 2001). According to Pohl *et al.* (2005), the benefits from the adoption of SPL are: quality improvement, reduction of time to market, reduction of maintenance effort, and so on. These benefits can be summarized as following:

- **Quality Improvement**. Artifacts in the platform are analyzed and tested in many products. Reusable assets have their quality attested on many occasions, into different contexts, leading to a higher product quality. The quality assurance implies a higher chance of detecting faults and correcting them.

- **Reduction of Time to Market**. One of the main critical success factors for a product line is the time to market. Product line engineering demands a higher upfront investment if compared to single-systems engineering. However, the time to market is significantly shortened as numerous artifacts can be reused in new products.

- **Reduction of Maintenance Effort**. Changes in reusable assets are propagated to all products whenever artifacts of the platform are changed or new artifacts are added into it. It usually leads to a simpler and cheaper maintenance and evolution, if compared to maintain and evolve a bunch of single products in a separate way.

- **Reduction of Development Costs**. An essential reason for introducing product line engineering is cost reduction. When artifacts are reused in several different kinds of systems from the platform, rather than being developed from scratch to each product, it has as consequence cost reduction. Figure 2.1 shows the costs of producing several single systems to the costs of producing them using an SPL approach. This figure also shows the accumulated costs need to develop n different systems. The costs to develop a few systems in an SPL approach are higher than in a single systems approach. However, using product line engineering, the costs are significantly lower for larger systems quantities.

- **Benefits for the Customers**. Software product lines can bring important benefits for customers, since they have the guarantee of getting products adapted to their real needs and wishes. Despite possessing individual features, products of a product line have a lot

in common due to the reused artifacts in the platform. In addition, customers do not have to learn new ways of using another product derived from the same platform.

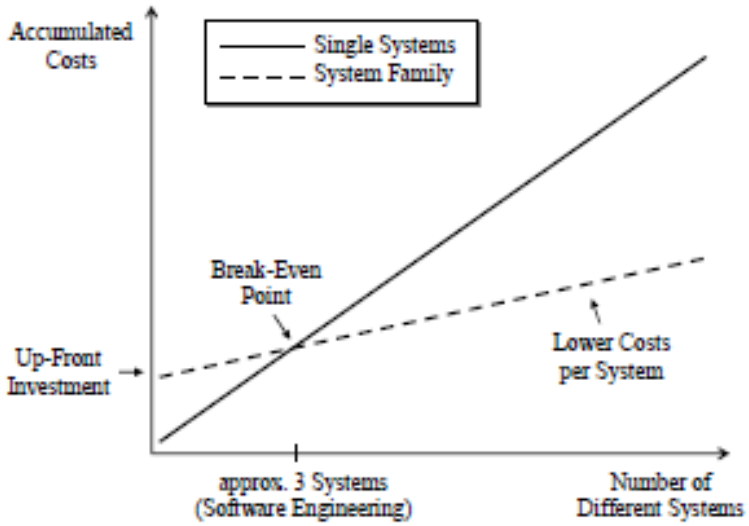

Figure 2.1: Costs from a single system compared to SPL (Pohl *et al.*, 2005)

2.1.2 Essential Activities

Pohl *et al.* (2005) proposed a framework for SPLE paradigm divided in two processes: domain engineering and application engineering. Figure 2.2 shows the processes and the activities for each process.

Domain Engineering is the process that aims to establish a reusable platform and define the commonality and the variability of the product line. Thus, the Domain Engineering is responsible for ensuring that the available variability is suitable for producing applications. It also involves defining the scope of the SPL, building reusable artifacts and providing feedback about the feasibility of realizing the required variability. Domain Engineering is divided in five sub-processes: domain requirements, domain design, domain realization, domain testing, and product management (Pohl *et al.*, 2005).

Application Engineering corresponds to the process responsible for deriving product line applications from the platform created in domain engineering, where the previously developed components are assembled to compose a product. It also includes, as high as possible, reuse of the domain assets; exploring the commonalities and variabilities of the SPL during the development of an application and linking the variability according to the applications needs

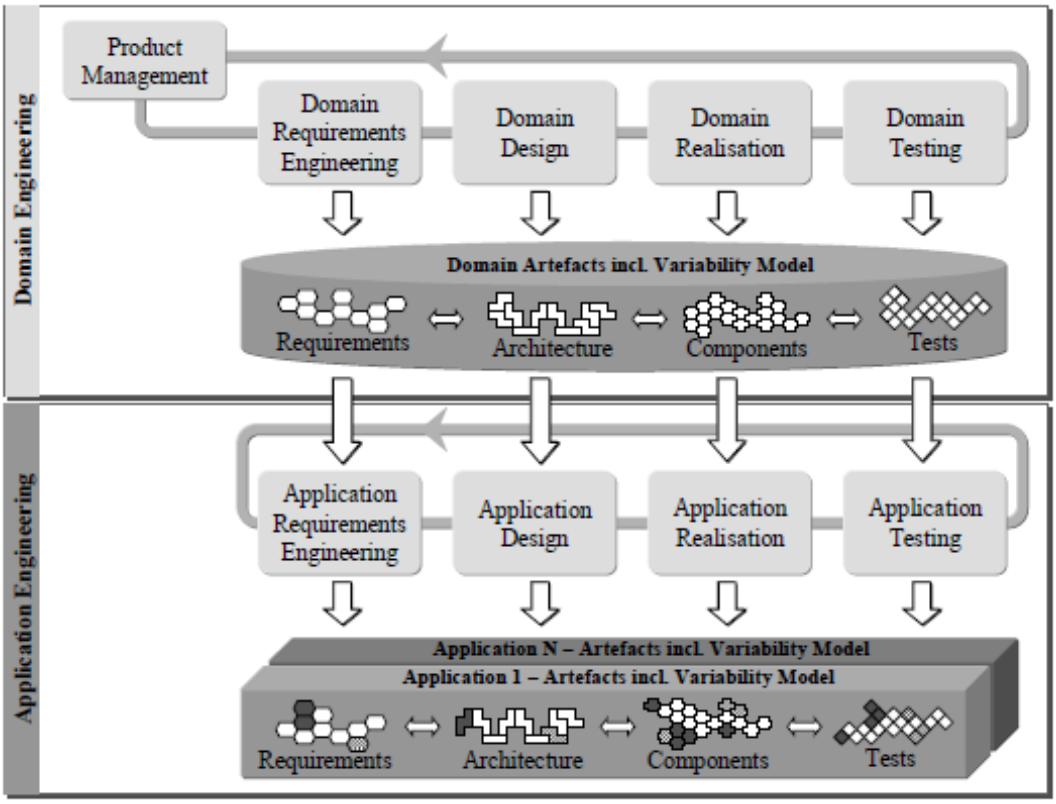

Figure 2.2: SPL framework processes (Pohl *et al.*, 2005)

(Pohl *et al.*, 2005). The application engineering is composed of four sub-processes: application requirements engineering, application design, application realization, and application test.

Clements and Northrop (2001) also discuss the activities for SPL. According to them, there are three essential activities: core assets development, product development and management, as showed in Figure 2.3. Each rotation circle represents one key activity. All three are connected together as they would in motion, showing that all three are closely related and highly interactive.

The main difference between this one and the aforementioned approach is the management activity. The management circle represents the activities of technical and organizational management. Technical management is responsible for requirements control and the coordination between core assets and product development. The organizational management is responsible for the production constraints and defines the production strategy (Clements and Northrop, 2001).

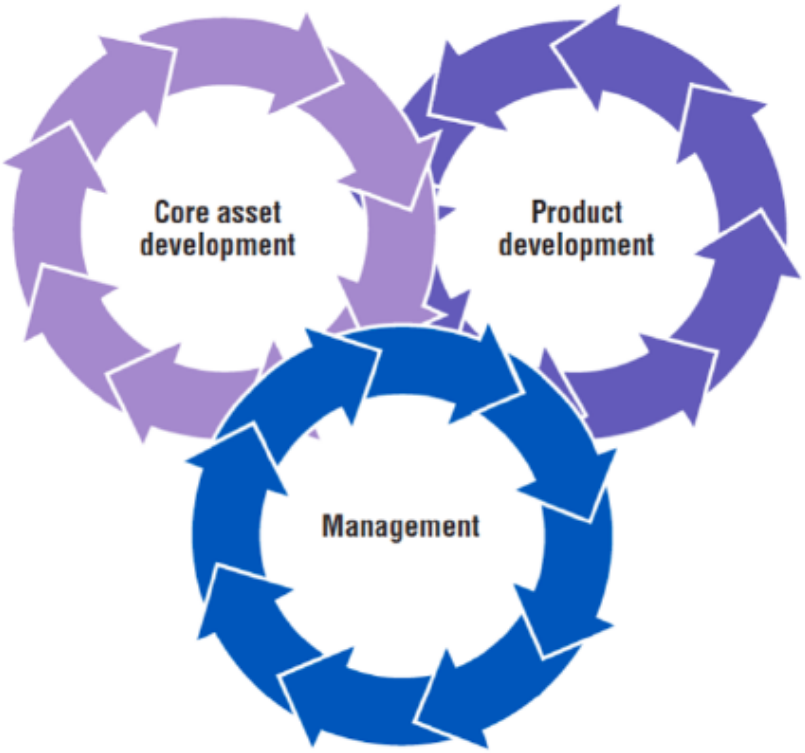

Figure 2.3: Essential Product Line Activities (Clements and Northrop, 2001)

2.2 Product Derivation

The separation into domain engineering and application engineering allows the development of reusable assets, which are shared among the products within that domain. It is during application engineering that the individual products within a product line are constructed. Product derivation is a key activity in application engineering and addresses the selection and customization of assets from the SPL (Deelstra *et al.*, 2005).

The application engineering process involves requirements engineering, design, implementation, and testing so that each of these sub-processes needs to consider the existing reusable assets and their variability to effectively use the product line. According to Rabiser *et al.* (2010), product derivation is about selecting and customizing shared assets during application engineering.

Deelstra *et al.* (2005) give an adequate definition for product derivation: “A product is said to be derived from a product family if it is developed using shared product family artifacts. The term product derivation therefore refers to the complete process of constructing a product from

product family software assets."

Figure 2.4 presents a high-level application engineering process adapted by Rabiser *et al.* (2010) from the product line engineering framework representation defined by Pohl *et al.* (2005). In this figure, the product derivation is represented by the upper white vertical arrows, which represent the selection and customization of reusable assets during the application engineering. The lower white arrows, as stated by Rabiser *et al.* (2010), denote deployment activities necessary to achieve a final product. The domain engineering process is not discussed in their work.

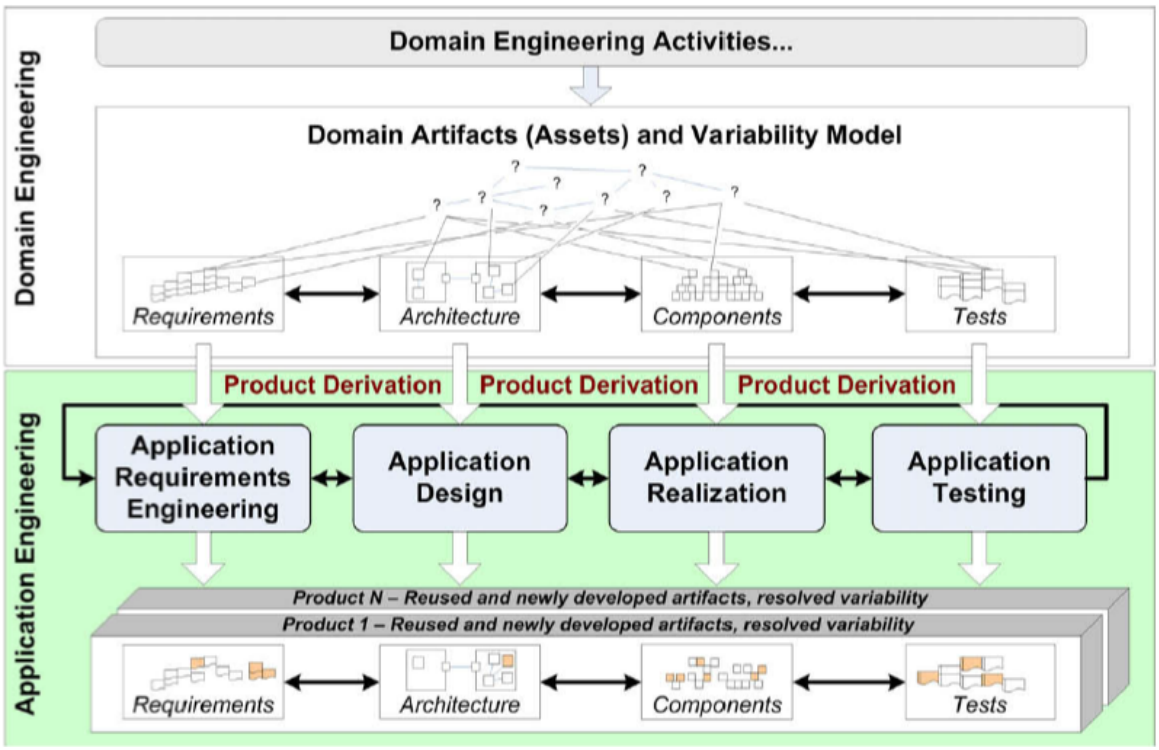

Figure 2.4: The product derivation during the SPL processes (Rabiser *et al.*, 2010)

The investments needed to develop the reusable artifacts during domain engineering, are compensated by the benefits in getting individual products during application engineering. Deelstra *et al.* (2005) state that the basic reason to research and invest in high-level technologies for product families is to obtain the maximum benefit from this initial investment, i.e., minimize the proportion of costs of application engineering.

Rabiser *et al.* (2010) identified in a recent systematic literature review, an increasing number of publications, workshops, and conferences over the last decade showing the general interest in product derivation. However, compared to the vast amount of research results on developing and modeling product lines (Chen and Babar, 2011), only few approaches and tools are available for

product derivation (Rabiser *et al.*, 2011).

2.2.1 Key Activities

In order to identify the key activities for product derivation, Rabiser *et al.* (2011) compared two approaches developed in different and independent research projects: Process framework for Production Derivation (Pro-PD) and Decision-Oriented Product Line Engineering for effective Reuse: User-centered User (DOPLER). Both approaches have been developed and validated in research industry collaborations with different companies. Pro-PD, for example, aims at defining general process framework for product derivation, focusing on the activities, roles and artifacts used to derive products from a SPL. The DOPLER approach has the objective of defining a user-centred, tool-supported product derivation approach, with the goal of meeting the industry needs.

Based on the mapping between those approaches, they defined the key activities for product derivation divided in three groups: (i) preparing for derivation, (ii) product derivation/ configuration, and (iii) additional development/testing, as presented on Figure 2.5.

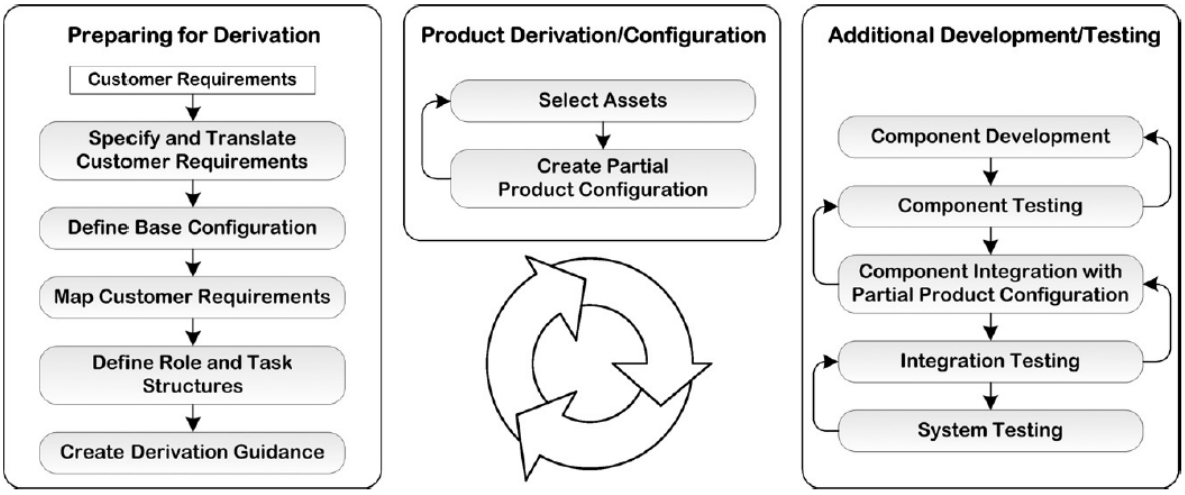

Figure 2.5: Key activities for product derivation (Rabiser *et al.*, 2011).

2.2.1.1 Key activity 1 - Preparing for derivation

The derivation process does not start "from scratch", i.e., by just selecting features or making decisions. From both observed research projects, before actual derivation, some preparatory activities are needed to be conducted (Rabiser *et al.*, 2011):

- **Specify and translate customer requirements** - Clearly specify customer requirements is the starting point for product derivation. They can be translated into the internal organizational language, which goal is avoiding a terminology confused and assets described in customer-specific language.
- **Define base configuration** - From a set of existing platform configurations, a base configuration can also be chosen as a starting point for derivation. Experiences acquired in past projects are of great importance, because customers could have similar requirements providing possibility of reuse. However, if none configuration is appropriate, a new base configuration should be created.
- **Map customer requirements** - Base configuration is used to map the requirements of customers. Requirements which cannot be satisfied by existing assets have to be negotiated with the customer. Questions as profitability of the platform assets for the whole product line must be taken into consideration.
- **Define role and task structures** - The goal is to define who is responsible for the remaining tasks in the derivation process. This is interesting by provide different views on variability for different people involved in the process. In addition, it is important knowing who made what and when.
- **Create derivation guidance** - Creating ways to facilitate the decision making by domain experts is essential, since remaining variability must be explained. Moreover, while sales people need understand variability from a high level, engineers need to know the details.

2.2.1.2 Key activity 2 - Product derivation/configuration

The goal of product derivation is to reuse the platforms artifacts as much as possible, besides minimizing the need for product-specific development. Thus, this step starts with a selection and customizing of assets from platform, identifying if new developments are necessary. It should be an iterative process to ensure that all customer requirements have been fulfilled (Rabiser *et al.*, 2011):

- **Select assets** - As the Key activity 1 defined the role and task structures, now assets must be selected from the product line. Tools support is of great importance and can be used for evaluating the dependencies and constraints between the assets.

- **Create partial product configuration** - A step-by-step, iterative manner, is used to create an adequate partial product configuration, which implements a software product where not all variability has been resolved (Deelstra *et al.*, 2005). In an ideal case, this first product would be enough to satisfy the customer requirements. However, in most projects it is still necessary some additional development, that should have its activities defined and prioritized based on customer requirements. A good alternative for supporting further negotiations with customers is the use of simulations based on partial configurations.

2.2.1.3 Key activity 3 - Additional development/testing

Additional development/testing represents the last set of activities performed during the product derivation process. The product development team is responsible to implement the required changes at the product level (Rabiser *et al.*, 2011).

- **Component development** - In this stage, new functionalities implementations or adaptions of platform components are developed. It is important to take in consideration that new components must be developed with possibility that they can be later updated to a platform asset.
- **Component testing** - Once a component has been developed or adapted, it needs to be rigorously tested. Conventional unit test methods can be utilized (Kauppinen and Taina, 2003).
- **Component integration with partial product configuration** - The newly developed and adapted assets must be integrated with the partial product configuration. In order to do this, it may be necessary writing "glue" code to interfaces; or implementing architectural changes to facilitate integration.
- **Integration testing** - This stage is necessary to verify whether the newly developed or adapted assets are interacting correctly with the existent architecture. The product's consistency and correctness has to be checked.
- **System testing** - Since this is the last stage, the entire product must be checked in order to ensure the product-specific requirements. In the case of this product satisfy the requirements, it is delivered. However, If it is not approved, further iterations are required.

2.2.2 Variability Management

Feature modeling is one of the main techniques used in SPLE to manage the commonality and variability between products of a product line. Feature modeling was proposed by Kang *et al.* (1990c) as part of the Feature-Oriented Domain Analysis (FODA) method and since then it has been applied in a large number of domains, including for instance, telecom systems, template libraries, network protocols, and embedded systems (Czarnecki *et al.*, 2005). Large companies such as Microsoft also integrated feature modeling into their software factories approach (Greenfield and Short, 2003).

Products in a product family tend to vary, and the differences between them can be described in terms of features. According to Bosch (2000), feature is "a logical unit of behavior specified by a set of functional and non-functional requirements". Another definition is proposed by Kang *et al.* (1990a) as "a prominent and distinctive user visible characteristic of a system". Additionally, Czarnecki and Eisenecker (2000a) defined a feature is a system property that is relevant to some stakeholder and is used to capture commonalities or discriminate among systems in a family.

Features can be organized through a feature diagram. Feature models are feature diagrams with additional information such as feature descriptions, binding times and priorities. A feature model starts during the domain engineering process by modeling common and variant features in a specific domain. In application engineering, during the product derivation process, feature models are configured according to the specific requirements of a target application.

A feature model provides a graphical representation of features, including variability relations, features constraints and dependencies. Features in a feature model are usually classified as (Czarnecki and Eisenecker, 2000b):

- **Mandatory feature**: this kind of feature must be selected whenever its parent feature is selected during the configuration process (Figure 2.6(a)).
- **Optional feature**: this kind of feature may or may not be selected when a parent feature is selected during the configuration process (Figure 2.6(b)).
- **OR feature group**: one or more features in the OR feature group must be selected during the configuration of the feature model (Figure 2.6(c)).
- **XOR (alternative) feature group**: one and only one of the features in the XOR feature group must be selected during the configuration of the feature model (Figure 2.6(d)).

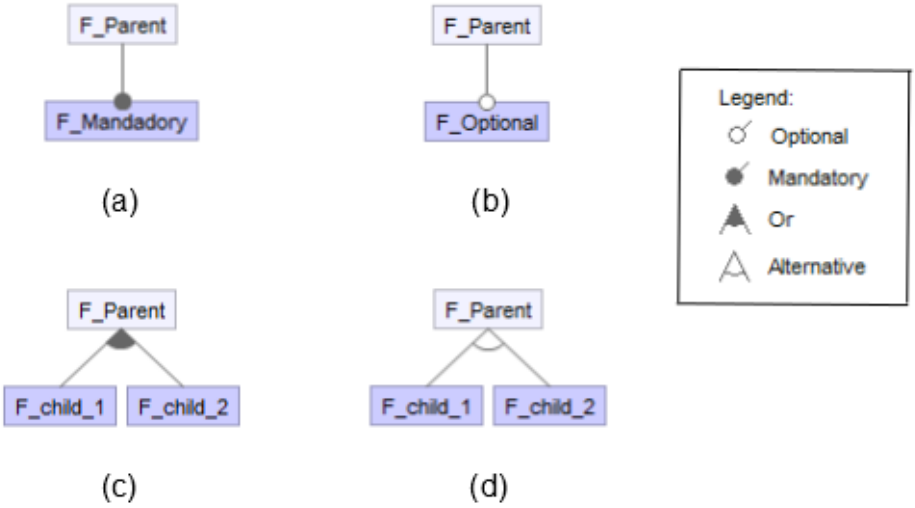

Figure 2.6: Features classification: (a) Mandatory features; (b) Optional features; (c) OR group; (d) XOR group.

Besides the relations between a parent feature and its child features, another relations can be present in feature models, like integrity constraints, which have two types: *requires* and *excludes* relationship (Czarnecki and Eisenecker, 2000b). The former aims at representing that a presence of a given feature requires the inclusion of another feature, for example: if A and B are features present in the same feature model with a requires relationship from A to B, the selection of A in a product implies the selection of B. The latter is the opposite, the presence of a given feature requires the exclusion of another feature, A and B cannot be part of the same product.

In a feature model, features without any children are called atomic or leaf features, and features which are decomposed into sub-feature(s) are called non-atomic or intermediate features (Czarnecki and Eisenecker, 2000b). In addition, we called abstract feature if and only if it is not mapped to any implementation artifacts, and concrete feature if it is mapped to at least one implementation artifact (Thum *et al.*, 2011).

Among the aforementioned feature definitions stated in the beginning of this sub-section, Bosch's definition covers both functional and non-functional aspects. Hereafter, we consider this definition more appropriated for our work, since we intend to configure the feature model based on both functional and non-functional properties.

2.3 Non-Functional Properites

SPLs are used to generate a variety of related products in a specific domain, and these products are distinguished in terms of features. Based on the selection of features, stakeholders can derive tailor-made programs satisfying functional requirements. Nevertheless, tailoring the variants regarding functional requirements alone is often not enough.

In addition to functional requirements, with the advent of many different applications scenarios comes the need for requirements regarding non-functional properties (NFPs). According to Siegmund *et al.* (2012), in the literature, the definition of non-functional properties is not consistent. NFPs, in many publications, are also referred to as quality attributes (QAs) (Robertson and Robertson, 1999; Glinz, 2007; Chung and Prado Leite, 2009).

Robertson and Robertson (1999) define a non-functional property as: "A property, or quality, that the product must have, such as an appearance, or a speed or accuracy property." In this way, NFPs are those that do not express what a piece of software can compute, but how or under what circumstances it achieves its main objectives. Examples of such properties are reliability, security, memory footprint, performance, and so on.

NFPs are especially important in systems with limited resources in which, for example, binary size and memory consumption are limiting factors. These heterogeneous non-functional properties often lead to redevelopment of existing functionality (Siegmund *et al.*, 2010). SPLs should provide, for example, alternative implementations of the same functionality, which differ only with respect to specific NFPs. For instance, by implementing a feature in different ways, e.g., a product with a good performance and one with a smaller value of binary size.

Configuring NFPs in product lines is still a challenge (Sincero *et al.*, 2010). Usually, the result of a single NFP comes from the interaction of many features and even from other NFPs. According to Siegmund *et al.* (2010), the variability supplied by an SPL should allow generation of variants, which are equal with respect to functionality, but differ in their non-functional properties.

The explicit definition of NFPs during software configuration is not a common practice (Sincero *et al.*, 2007). Table 2.1 shows an example of the Linux Kernel configuration tool. This figure presents excerpts of a configuration help provided by this configuration tool, where few and vague descriptions are informed according to the features selected, as "this option **will slow down** process creation **somewhat**".

Table 2.1: Linux Kernel Configuration Messages (Sincero *et al.*, 2007)

Sysctl support: " Enabling this option will **enlarge** the kernel by **at least 8 KB**".
Ckeck for stack overflows: "this option will **slow down** process creation **somewhat**".
SMT Hyperthreading: "at a cost of **slightly** increased **overhead** in **some places**".
Memory Type Range Register: "this can **increase performance** of image write operations **2.5 times or more**".
Voluntary Kernel Preemption (Desktop): "providing faster application reactions, at the cost of **slightly lower** throughput".

Non-functional properties describe the quality characteristics of a system and are essential for the successful working of a system software. Especially for some specific kinds of software, such as embedded systems, those properties have an important role and influence the final outcome of the system. Figure 2.7 shows an example of a feature model for a text editors product line where some features have additional information related to NFPs of the product line: footprint and memory usage.

Some work, such as Sentilles (2012), state that there is a lack of generic support for specifying and manage non-functional properties. According to them, explicit modelling and reasoning about of NFPs is still not widespread; moreover, there are several different NFPs and many of them depend on external factors such as: underlying platform, usage scenario, or the context in which the system is running.

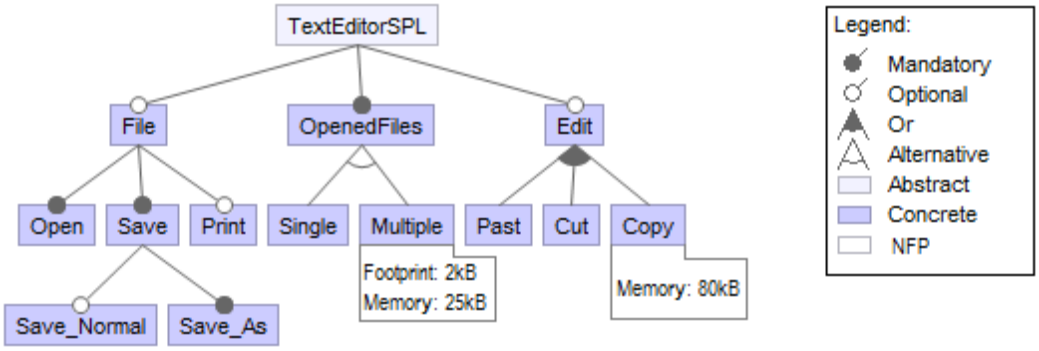

Figure 2.7: Feature Model of a Text Editor SPL.

2.3.1 Types of NFPs

In SPLE there are several kinds of non-functional properties (Mairiza *et al.*, 2010) that are classified in many different ways. One example of classification are the models proposed by ISO/IEC 25000 (2011), named software product quality model and model of quality in use. After years of development, the International Organization for Standardization released in 2011 a reworked software product quality model standard, which was strongly influenced by its predecessor, ISO/IEC 9126 (2001).

These aforementioned models define quality with categories of characteristics to software products and computer systems, besides providing a terminology for specifying, measuring and evaluating system qualities. The product quality model, for example, defines eight categories: Functional Suitability, Maintainability Usability, Performance Efficiency, Security, Reliability Compatibility, and Portability; while the model of quality in use defines five: Effectiveness, Efficiency, Satisfaction, Safety, Usability. Each one of these characteristics is still refined in same others sub-characteristics.

According to Wagner (2013), various critiques point out that the decomposition principles used for quality characteristics are often ambiguous, besides detailed measures are yet missing. Still according to them, less than 28% of the companies use this standard model and 71% of them have developed their own variants, which implies a need for customisation.

Quality attributes may also be divided into two categories, execution and evolution quality attributes (Mari and Eila, 2003). Execution qualities are observable at runtime as the behaviour of the system, and they are described in Table 2.2.

Table 2.2: Execution quality attributes (Mari and Eila, 2003)

Attribute	Description
Performance	Responsiveness of the system, which means the time required to respond to stimuli (events) or the number of events processed in some interval of the time.
Security	The systems ability to resist unauthorized attempts at usage and denial of service while still providing its service to legitimate users.
Availability	Availability measures the proportion of time the system is up and running.
Usability	The systems learnability, efficiency, memorability, error avoidance, error handling and satisfaction concerning users actions.
Scalability	The ease with which a system or component can be modified to fit a problem area.
Reliability	The ability of the system or component to keep operating over the time or to perform its required functions under stated conditions for a specific period of time.
Interoperability	The ability of a group of parts to exchange information and use the information exchanged.
Adaptability	The ability of software to adapt its functionality according to the current environment or user.

Evolution attributes are observable during system lifecycle that characterize different phases in the development and maintenance process, and they are described in Table 2.3.

Table 2.3: Evolution quality attributes (Mari and Eila, 2003)

Attribute	Description
Maintainability	The ease with which a software system or component can be modified or adapts to a changed environment.
Flexibility	The ease with which a software system or component can be modified for use in applications or an environment other than those for which it was specifically designed.
Modifiability	The ability to make changes quickly and cost-effectively.
Extensibility	The systems ability to acquire new components.
Portability	The ability of the system to run under different computing systems: hardware, software or combination of the two.
Reusability	The systems structure or some of its components can be reused in future applications.
Integrability	The ability to make the separately developed components of the system work correctly together.
Testability	The ease with which software can be made to demonstrate its faults.

2.3.2 Measuring NFPs

Measuring and configuring a specific property is a non-trivial task, since many but not all non-functional properties can be measured. For example, it is possible to measure the binary size property of an individual feature and aggregate these values for a specific variant. On the other hand, it is hard to measure, for example, security.

In terms of way to measure, NFPs can be specified in either a qualitative or a quantitative way. The qualitative non-functional properties can be described using scales, as for example, high, medium and low. On the other hand, metric based values are defined for the quantitative non-functional properties. Siegmund *et al.* (2012) categorized non-functional properties based on the measurement theory (Stevens, 1946). The non-functional properties were classified into three different classes: qualitative properties, feature-wise quantifiable properties and variant-wise quantifiable properties:

- **Qualitative properties** typically require domain knowledge and can only be described in a qualitative way using an ordinal scale, i.e., there is no metric from which we can retrieve quantifiable measures. Common representatives of this class are security, availability, reliability and usability.

- **Feature-wise quantifiable properties** can be measured on a metric scale. An important requirement for feature-wise quantifiable properties is that we can measure a single feature

directly or infer the results of the measurement of a variant to single features. A feature-wise measurement allows to annotate each feature and the implementation unit of an SPL with a specific value and to compute a value for a feature selection. Maintainability is an example of this class, and according to McCabe (McCabe, 1976), it can be measured with code metrics such as lines of code and cyclomatic complexity. Other examples are footprint, adaptability and price of a feature.

- **Variant-wise quantifiable properties** have no meaning for single features. It is not able to quantify the influence of individual features on the NFPs of a concrete product. Usually, it is necessary to execute the program for measuring some NFPs. Considering the huge number of possible variants, the NFPs to this kind of property should be measured only for a predefined set of selected features. Representatives of this class are: performance, response time and memory consumption.

2.4 Chapter Summary

This chapter presented the literature review, which is mainly focused on Software Product Lines (SPL), Product Derivation and Non-Functional Properties (NFPs). It included fundamental aspects of SPL and motivations for applying it as a software development strategy. Regarding the processes of an SPL, Domain Engineering and Application Engineering, we highlighted the characteristics of a configuration activity that is performed by the Product Derivation process in the application engineering.

Product Derivation is one of the main activities in SPL. It is the process of building a product from shared product family artifacts (Deelstra *et al.*, 2005). In a product line organization, the use of an adequate product derivation process helps to ensure the return of investment required to develop the platform assets (O'Leary *et al.*, 2009). However, the area of product derivation is still rather immature (Rabiser *et al.*, 2010).

Another also immature area is the management of NFPs during the product derivation process, which means, assembling a product taking into account functional and non-functional properties. Next chapter presents a Systematic Literature Review (SLR) focusing on this point, SPL approaches that have been reported regarding the analysis of NFPs, besides categorizing NFPs used in the scientific literature regarding development of SPL.

3

A Systematic Literature Review on NFPs in SPL

Software Product Lines (SPL) engineering establishes a systematic software reuse strategy. The goal is to identify commonality (common functionality) and variability points among applications within a domain, and build reusable assets to benefit future development efforts (Pohl *et al.*, 2005).

SPL in practice has been very successful in managing features that comprise both functional properties (FPs) and a large number of non-functional properties (NFPs). However, there are many NFPs that cannot be expressed and then realized in form of features, but require different approaches. How to deal with them is still not well established, neither in theory nor in practice. Thus, there is a need to provide a systematic knowledge about this.

Etxeberria and Sagardui (2005) showed that most approaches for quality analysis aims at investigating "product line quality attributes (QAs)[1]", i.e., not observable via execution or development attributes. In this sense, this Systematic Literature Review (SLR) focuses on execution/runtime properties, visible and measurable at source-code or during the program execution (Crnkovic *et al.*, 2005), such as reliability and performance. These are highly relevant properties in next generation computing applications, such as embedded and real-time systems (Siegmund *et al.*, 2012). In fact, emerging computing application paradigms require systems that are not only reliable, compact and fast, but which also optimize many different competing and conflicting objectives, e.g., as response time and consumption of resources (Harman *et al.*, 2012).

To the best of our knowledge, few work has focused on the analysis of QAs for SPL engineering (Peng *et al.*, 2011; Myllärniemi *et al.*, 2012). Besides, they are often incomplete

[1] In the following, we will interchangeably speak of QAs and NFPs.

in terms of comprehensive coverage, or focused on different research topics. Montagud *et al.* (2012) proposed a catalog of measures for QAs found in the SPL lifecycle. In this present investigation, we elaborate on this work, by investigating how runtime NFPs are introduced in the SPL approaches.

We carried out a systematic literature review (Kitchenham and Charters, 2007) of the published literature on SPL approaches reporting on NFPs. The goal of this is twofold: (i) to present a holistic overview of SPL approaches that have been reported regarding the analysis of runtime NFPs, and (ii) to categorize NFPs used in the scientific literature regarding development of SPL.

The remainder of this chapter is organized as follows. Section 3.1 describes the research method used in this review. Section 3.2 reports the results. Section 3.3 discusses the implications of our results for research and practice, along with the limitations of the review. Finally, Section 3.4 presents our summary including conclusions and suggests areas for further research.

3.1 Research Method

A systematic literature review (SLR) was performed to find and summarize available evidence of runtime NFPs in SPL engineering. A SLR is a research method applied to identify, analyze, and interpret all available information related to a particular research question or area of interest (Kitchenham and Charters, 2007). It provides an objective assessment of a research topic in a reliable, rigorous, and methodological manner (Montagud *et al.*, 2012), aiding to both understand the current direction and status of research, and to provide background to identify research challenges (Zhang *et al.*, 2011).

Along this Section, we detail each element composing this SLR, mainly the research questions, the strategy employed to locate primary studies, the data extraction and synthesis process.

3.1.1 Research Questions (RQs)

This review aims at presenting a holistic overview of the existing studies that have been reported regarding the analysis of runtime NFPs. We employed the PICOC (Population, Intervention, Comparison, Outcome and Context) structure (Petticrew and Roberts, 2006) to define the research questions in this review. The structure is defined as follows.

- *Population.* Research handling runtime NFPs in SPL.

- *Intervention.* Published studies in the field, including methods, techniques and processes.
- *Outcomes.* A set of approaches handling runtime NFPs in SPL engineering, and the leveraged quality attributes and application domains.
- *Context.* SPL engineering with a strong focus on quality attribute aspects.

Particularly, the comparison criterion was not considered in this review, as we do not compare interventions. Concretely, we stated the following RQs:

RQ1. How SPL approaches handle runtime NFPs?

This question aims at identifying the approaches that handle runtime NFPs in SPL engineering. We analyzed the main goals of the approaches and which NFPs they address. The term "approach" is herein used to refer to a published document describing a method, technique, or process related to the topic under investigation. Thus, we split this question in three sub-questions:

RQ1.1 What approaches handle runtime NFPs in SPL? The purpose of this question is to leverage approaches handling runtime NFPs in SPL.

RQ1.2 What NFPs emerge at runtime? This question aims at identifying NFPs that are commonly handled at runtime. Such kind of properties are those which usually depend on observations via execution or operation (Etxeberria and Sagardui, 2005).

RQ1.3 What application domains are best covered by the existing approaches? This question aims to leverage the application domains (e.g., scientific systems, business systems, etc.) where the identified approaches were used.

RQ2. How much evidence is available to support the approaches?

This question assesses the level of evidence of each identified approach. The level of evidence is a way to evaluate the maturity of methods and tools. Based on evidence-based healthcare research (NHMRC, 2000; Higgins and Green, 2011), Kitchenham and Charters (2007) proposed a hierarchy of Software Engineering study designs, classifying studies into five levels **(L)** of evidence. Later on, Alves *et al.* (2010) proposed a more practical assessment, by defining the following hierarchy:

- **L1** - No evidence.

- **L2** - Evidence obtained from demonstration or working out toy examples.
- **L3** - Evidence obtained from expert opinions or observations.
- **L4** - Evidence obtained from academic studies, e.g., controlled lab experiments.
- **L5** - Evidence obtained from industrial studies, e.g., causal case studies.
- **L6** - Evidence obtained from industrial practice.

The hierarchy consists of classifying from weakest to strongest evidence which each study presents. We found it to be a suitable classification to provide answers to this RQ.

RQ3. What are the limitations of the existing runtime NFPs support?

Understanding the limitations of the investigated approaches helps ensuring accurate interpretation of findings. Thus, we performed a critical appraisal on the quality of the included studies, under established criteria. The criteria are quality indicators, which may be useful to identify the likely limitations of an approach. Table 3.1 lists the quality criteria used in this assessment. They were defined based on the work of Bass *et al.* (2003) and Alves *et al.* (2010), and cover the following issues (Bass *et al.*, 2003):

- **Reporting:** assesses the quality of the study's rationale, aims, and context.
- **Rigor:** assesses the rigor of the research methods employed to establish the validity of data and the trustworthiness of the findings.
- **Credibility:** assesses the credibility of the study methods for ensuring that the findings were valid and meaningful.
- **Relevance:** assesses the relevance of the study for the software industry at large and the research community.

3.1.2 Search strategy

The process of identifying primary studies started by executing a search query on the electronic databases for research (the *search engines*). The repositories used were: *ACM Digital Library, IEEE Xplore, Science Direct, Springer,* and *Scopus*. They, together, cover almost all important journals, conferences, and workshops papers in the Software Engineering field.

Table 3.1: Quality assessment criteria

	Criteria	Issue
RQ3.1	Is the paper based on research?	*Reporting*
RQ3.2	Is there a clear statement of the aims of the research?	
RQ3.3	Is there an adequate description of the context in which the research was carried out?	
RQ3.4	Was the research design appropriate to address the aims of the research?	*Rigor*
RQ3.5	Was the data analysis sufficiently rigorous?	
RQ3.6	Is there a way to validate the methods?	*Credibility*
RQ3.7	Is there a clear statement of findings?	
RQ3.8	Are there any guidelines to practitioner interested in using this method?	*Relevance*

Based on our research goal, which is investigating runtime NFPs approaches in SPL engineering, the following major search keywords were used to formulate the search query: *quality attributes* and *software product lines*. Alternative words and synonyms were also used for such keywords. Then, it was created an initial pilot search string by joining major keywords with Boolean AND operators, and the alternative words and synonyms with Boolean OR operators. As a means to calibrate the search string, we first applied it in the search engines, in a general search, the query was computed on the whole body of searchable content. Next, it was carried out more specific searches by delimiting the scope to include some conferences and journals which we were aware of their likely results. Hence, after some trials, the search string employed in all source engines is represented as follows:

> *(“quality attributes”* ***OR*** *“non-functional”* ***OR*** *“extra-functional”)* ***AND*** *(“software product line”* ***OR*** *“software product family”* ***OR*** *“SPL”).*

We faced some problems when carrying out the automated search: a huge number of papers were not relevant to the context of this study since it was found a number of publications, in the form of, e.g., summaries, tutorial abstracts, and so on, that cannot be considered as research papers. Conversely, our search did not retrieve important papers for the field we were aware of. In this effect, as a means to capture as many relevant citations as possible, we also carried out a manual, issue-by-issue, search over peer-reviewed journals, and conference and workshop proceedings. The reviewed sources are listed in the Table 3.2.

Table 3.2: Conferences and journals

Conferences
Asia-Pacific Software Engineering Conference
International Conference on Automated Software Engineering
International Conference on Advanced Information Systems Engineering
ACM Sigsoft Symposium on Component-Based Software Engineering
International Symposium on Code Generation and Optimization
Empirical Software Engineering and Measurement
Euromicro Conference on Software Engineering and Advanced Applications
European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering
International Conference on Software Engineering
International Conference on Software Engineering Advances
International Conference on Software Reuse
International Symposium on Empirical Software Engineering
International Workshop on Software Architectures for Product Families
Model-Driven Engineering Languages and Systems
International Workshop on Non-functional System Properties in Domain Specific Modeling Languages
Software Product-Family Engineering
Conference on Quality Software
International ACM Sigsoft Conference on the Quality of Software Architectures
Requirements Engineering: Foundation for Software Quality
International Conference on Software Engineering and Knowledge Engineering
Software Product Line Conference
International Workshop on Variability Modelling of Software-intensive Systems
Journals
ACM Transactions on Software Engineering and Methodology
Communications of the ACM
IEEE Computer
IEEE Software
IEEE Transaction on Software Engineering
IET Software Journal
Information and Software Technology Journal
Journal od Software Maintenance and Evolution: Research and Practice
Software Practice and Experience
Software Quality Journal
Automated Software Engineering
Journal of Software
IBM Journal of Research and Development
International Journal of Software Engineering and Knowledge Engineering
Empirical Software Engineering
ACM Computing Surveys

3.1.3 Study selection criteria

The search performed with the search string is a syntactic search, that results in a set of papers in which the research string appears. However, it is possible that a retrieved paper matches the search string, but it addresses topics other than the focus of this research. Therefore, a semantic check served to complement the syntactic search.

The semantic search we followed was earlier applied as a process to screening papers for subsequent review (Neto *et al.*, 2011). The first step is to read the title and abstract of the papers in order to ensure that the collection is consistent with the research area under study. Hence, while reading the titles and abstracts of the retrieved papers, we applied the inclusion and exclusion criteria to select the papers. Based on such criteria, we could select all candidate primary studies to be included in this review. These are papers that matched the inclusion criteria, and did not fit in any exclusion criterion. The inclusion and exclusion criteria used were:

- **Inclusion criteria:**
  - Papers published up to June in 2013.
  - Papers dealing with runtime NFPs in SPL engineering.
  - Papers which handle QAs from the exectution/runtime viewpoint, i.e., attributes that usually cannot be analyzed before product assembly.
- **Exclusion criteria:**
  - Papers not written in English.
  - Papers presenting QAs not suitable for prediction/estimation/measurement in final products.
  - Introductory papers for special issues, books or posters.
  - Duplicate copy of the same research study.

Figure 3.1 illustrates the search and selection process employed in this review. It shows the number of papers found in both automated and manual search, and also the amount of papers remaining after further screenings. Phase 1 denotes the automated search carried out within the search engines. The search engines retrieved a pool of 2,635 publications. We used the StArt tool (Fabbri *et al.*, 2012) and electronic spreadsheets to help organizing the references. From such a set, 103 studies were duplicate, as more than one search engine indexed the same venues. We also found 3 literature reviews. Moreover, by reading the titles and abstract of the remaining

papers, the number of 2,394 were considered to be irrelevant to the focus of this review, and as such they were discarded at this point.

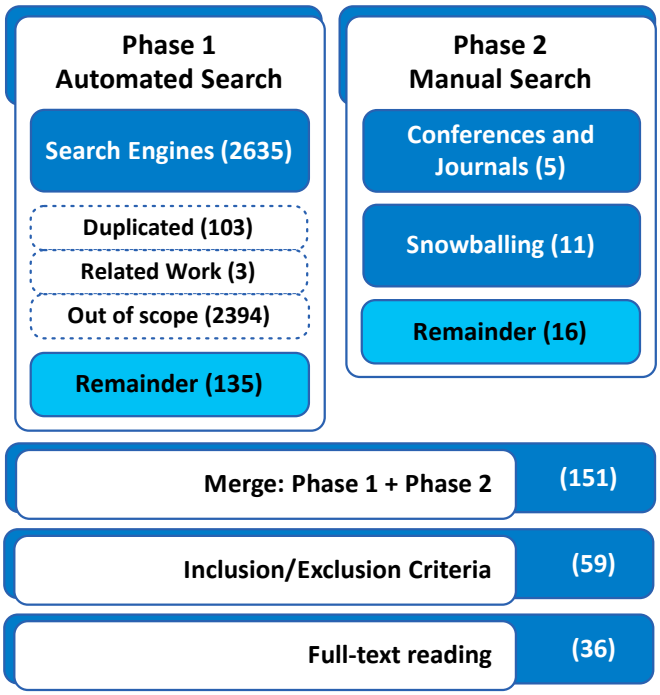

Figure 3.1: Search and selection process.

Phase 2 encompasses the manual search. By visiting the proceedings of conferences and workshops, and journals, we found 5 additional papers. Furthermore, as part of the manual search, a snowballing search was employed, in which the references of every retrieved paper was visited. It resulted in 11 more papers. Hence, the manual task contributed with 16 more papers.

After merging the results from both automated and manual search, we had a pool of 151 papers. We applied the inclusion/exclusion criteria in this set, to identify those to consider for a full-text reading. Then, we had a set of 59 papers. After reading the 59 primary study candidates, 36 papers were selected as primary studies in this SLR.

3.1.4 Data extraction and quality assessment

We created a data extraction form for the purpose of answering the RQs and extracting relevant information from the primary studies (S). The synthesis process and data extraction was carried out by reading all 36 papers, listed in the Appendix A.1, and extracting relevant data, as follows.

- Venue, title, year, and author(s).

- A short description of the proposed approach (RQ1), including motivation and goal of the study (RQ1.1).
- QAs addressed in the paper (RQ1.2).
- The application domains to which runtime NFPs are worked on (RQ 1.3).
- The evidence level of the proposed approaches, according to the list earlier presented in Section 3.1.1 (RQ2).
- The quality of the study, according to the quality assessment criteria presented in Table 3.1 (RQ3).

3.2 Results

Table 3.3 gives an overview of the distribution of the primary studies based on their publication channels, along with the number of studies from each source. Most of these 36 studies were published in leading journals, conferences or workshops that belong to the most cited publication sources in the area of software product lines engineering and the global software engineering community. Specifically, we have identified 9 journal papers, 22 conference papers and 5 workshop papers (see Table 3.3). As described in Section 3.1, all the studies fulfill the criteria for quality assessment. Most of the conferences papers, for example, have been published in the Software Product Line Conference (SPLC), which is the premier international forum for researchers and practitioners.

Figure 3.2 shows the temporal view of the primary studies, in which the published papers were identified from the year 2003. We may notice a trend curve revealing an increase of the number of papers as of the year 2007. The small amount of studies in 2013 (compared with 2012) is justified by the fact that this SLR only covered the first semester of 2013. Such a trend indicates an increasing interest by the SPL community in creating novel approaches to satisfy users NFPs.

As described in Section 3.1, we extracted and synthesized data to answer our RQs. In this section, we discuss the findings of the primary studies (listed in Appendix A.1) by considering the RQs.

Table 3.3: Studies distribution per publication source

Source	#
Journals	
Information and Software Technology	3
Journal of Software Quality	2
Journal of Systems and Software	2
IEEE Software	1
Journal of Software Quality Control	1
Conferences/Workshops	
Intl. Software Product Line Conference (SPLC)	9
Asia-Pacific Software Engineering Conference (APSEC)	3
Intl. Conf. on Advanced Information Systems Engineering (CAiSE)	2
Intl. Conf. on Software Engineering (ICSE)	2
Intl. Workshop on Non-functional System Properties in Domain Specific Modeling Languages (NFPinDSML)	2
Intl. Workshop on Variability Modelling of Software-intensive Systems (VaMoS)	2
ACM Symposium on Applied Computing (SAC)	1
Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS)	1
Hawaii Intl. Conf. on System Sciences (HICSS)	1
Intl. Conf. on Physics Education (ICPE)	1
Intl. Conf. on Software Engineering Advances (ICSEA)	1
Intl. Conf. on the Quality of Information and Communications Technology (QUATIC)	1
Intl. Workshop on Software Engineering for Embedded Systems (SEES)	1

3.2.1 RQ1 - SPL approaches

The quality of an SPL product is affected by the selected features and their composition. Different feature selections or compositions may differ for the software products NFPs, such as reliability, availability and performance. We next provide more details about the SPL approaches.

3.2.1.1 RQ1.1: What approaches handle runtime NFPs in SPL?

We categorized the primary studies into three groups, two for approaches presenting quality analysis (*Quality Prediction (QP)* and *Quality Estimation (QE)*), and one for approaches aimed at *Feature Selection (FS)*, as Figure 3.3 shows.

QP approaches seek to predict runtime NFPs of SPL products based on either historical data, expert knowledge or software metrics thresholds, or a model of the feature composition (i.e., by deriving the model of a product from the family model). **QE** approaches set runtime NFPs on the basis of measures obtained by either analyzing the source code of SPL products or observing

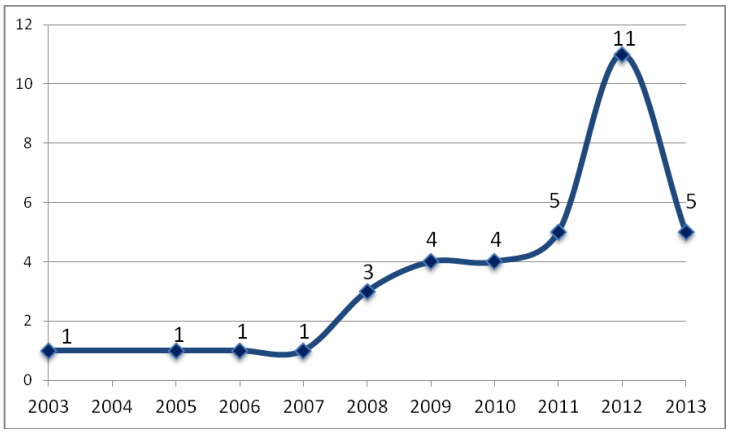

Figure 3.2: Number of studies by publication year.

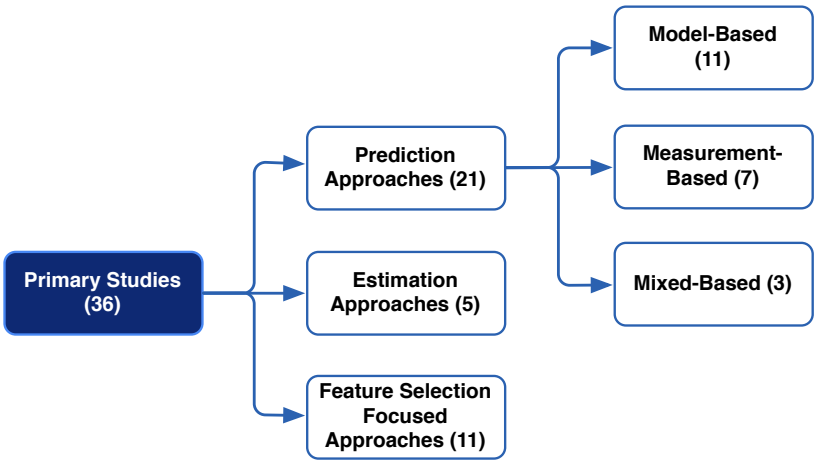

Figure 3.3: Primary studies classification

their running software[2].

FS-focused approaches provide guidelines to select features according to FPs and NFPs. Therefore, their main goal is not the quality analysis. Also, some of these approaches suggest the optimal set of product configuration actions needed to tackle required NFPs.

It is worth mentioning that the quality analysis can be *quantitative* - returning numerical results (e.g., performance and reliability bounds) and *qualitative* - providing more general considerations on products quality. Next, we describe the categories of the primary studied identified.

[2] An SPL product (also called *variant*) is produced by mapping the selected features to implementation units (e.g., software components or services).

Prediction Approaches

The approaches for predictive NFPs analysis of SPL products can be categorized by the kind of techniques used: (a) model-based, (b) measurement-based, and (c) mixed - combination of (a) and (b). These are discussed next [3].

(a) *Model-Based* approaches basically give the guidelines to predict NFPs by considering UML models, like sequence diagrams [S5][S12], and variability modelling techniques, such as feature models [S10][S29]. These diagrams are enriched and annotated with QAs information provided, for example, by experts' knowledge.

On the one hand, the quantitative analysis of NFPs is based on, for example: (i) Bayesian Belief Network (BBN) model, to capture the design knowledge and experiences of domain experts [S13][S28]; or (ii) Analytic Hierarchical Process (AHP), which needs less experts efforts than BBN, according to [S29]; Also evaluation strategies have been adopted. For example, queuing network models have been used to support product performance evaluation [S25][S26], and parametric model checking techniques have been exploited to verify NFPs of different configurations of an SPL [S5][S12]. On the other hand, for example, in [S10] a qualitative framework has been introduced for reasoning impacts of inadequate/insufficient domain assumptions on NFPs, while in [S35] the feature-oriented domain analysis has been extended with qualitative goal-oriented analysis.

(b) *Measurement-based* approaches aim at predicting NFPs based on, for example, the identification of relevant feature interactions [S18], or relevant/common features sets [S9][S22][S23]. Generating all products from an SPL to perform the quality analysis for each is impractical. In an SPL, the number of products increases exponentially with the number of features. The ongoing work [S09], for example, predicts quality by using feature sets defined in a sampling process based on static program analysis. The work in [S22][S23] provide a feedback to stakeholders from products measured through a limited set of features, based on a testing infrastructure. Other works are able to predict program performance by detecting performance-relevant feature interactions and measuring to what extent they interact [S18].

(c) *Mixed* approaches provide both quality variability modeling and measurement techniques. The ongoing work in [S27] investigates (i) qualitative modeling techniques that better fit in security and performance properties, besides (ii) how to use expert knowledge on security,

[3] Note that this kind of classification is typically used for quality prediction approaches (see Becker *et al.* (2004) for performance prediction of component-based systems).

and static analysis of feature implementations to derive performance metrics. Regarding the specification of NFPs, model-driven engineering principles have been exploited, for example, in [S6]. In particular, a mechanism for the validation of NFPs fulfillment, through the association of measures and thresholds, is presented. The work in [S4] proposes to reduce evaluation cost and effort in evaluating the different products by creating a generic evaluation model. This latter also discuss how to select a limited set of designs or products to quantitatively evaluate the quality of products.

There is a sub-group of prediction approaches that, besides quality analysis, provides support for the optimization of product configuration using the Constraints Satisfaction Problem (CSP). For example, [S19][S20][S21] developed a technique called SPL Conqueror. [S20] works with NFPs limited to the evaluation scenario, such as picture frequency; [S21] works with quantifiable NFPs, exemplified by footprint and memory consumption; and [S19] is more generic.

Estimation Approaches

This category mainly encompasses studies that, besides providing a means to estimate NFPs, provide support for the selection of implementation alternatives for the selected features. For example, on the one hand, the work in [S16] focuses on a quantitative approach based on the source code analysis for estimating the binary size of feature implementations. On the other hand, the approach is based on the run of a benchmark, proposed for performance evaluation of the whole product. The work in [S17] consider refactorings (changes in the source code structure) in correspondence to certain NFPs. The estimation method [S16] is also exploited in the holistic approach [S19][S20][S21] to optimize NFPs in SPL engineering. Other works also developed approaches to remove/replace some components with alternative ones, in order to achieve required quality levels [S02][S11][S30]. Additionally, [S02] proposes an aspect-oriented methodology, [S11] addressed a service-oriented approach, and [S30] is based on a model-driven strategy.

Feature Selection focused

The works of this category are not specifically focused on the quality analysis. Thus, NFPs parameters are simulated, or random values are generated. Usually, these works do not explicitly distinguish between predicted values and estimated means.

The feature selection activity has been supported from different perspectives in order to, for example: (i) generate a CSP from the feature model (see, e.g., [S1][S33]) or an orthogonal variability model [S15] associated with quality information; (ii) exploit the multi-objective

optimization for the product derivation [S14]; (iii) drive the feature selection with product usage contexts [S32]; or (iv) adopt artificial intelligence techniques, based on a fuzzy propositional language [S31], genetic algorithms [S36], and Hierarchical Task Network artificial technique [S24].

Another class of related papers proposes, for example, a model-driven approach for quality evaluation [S8], or deal with the integration of quality modeling with feature modeling (see, e.g., [S3][S7]), showing how the customer decisions affects the QAs [S3]. Also, the filtered cartesian flattening approximation technique [S34] was used for the feature selection.

3.2.1.2 RQ1.2: What NFPs emerge at runtime?

In Software Engineering there are different kinds of QAs (Mairiza *et al.*, 2010) classified in different ways (Crnkovic *et al.*, 2005; ISO/IEC 25000, 2011; Mari and Eila, 2003). Mari and Eila (2003) specified QAs into two categories: evolution and execution QAs. Whereas evolution attributes are observable during system development lifecycle, execution ones are observable at runtime, and dependent on factors such as system behaviour, number of existing features and which ones are selected for the product.

Research efforts have been devoted for evolution attributes. There are several work that analyze quality of the SPL (Etxeberria and Sagardui, 2005), but they focus on quality of the SPL process (mainly during design stage), not in SPL product. These work are concerned about quality of requirements, SPL architecture, reusability, modularity, i.e., they assess quality of SPLs in general, from the viewpoint of different kinds of stakeholders (e.g. domain engineers or salesman). In this sense, our proposal focuses on execution/runtime QAs, which are highly relevant and need to be properly addressed.

Mari and Eila (2003) defines the execution/runtime category by using eight attributes (as already described in Table 2.2): performance, security, availability, usability, scalability, reliability, interoperability and adaptability. Each attribute represents a more conceptual term and encompasses a set of fine-grained attributes. For example, performance is related to the relationship between the level of performance of the software and the amount of resources used, under stated conditions. It can be composed by the attributes, such as response time, space, latency, throughput, execution speed, resource usage, memory usage, accuracy, etc. (Mairiza *et al.*, 2010).

Table 3.4 reports the details of our NFPs categorization, found in the 36 primary studies. Despite some of them are domain-specific attributes and resource-related constraints, such as Channel Capacity/Latency, WLAN bandwidth, Cycle and Speed, we considered the NFPs

nomenclature exactly as mentioned in the original sources. As presented in the table, the large majority of studies is focused on performance issues (53.84%). However, we can also see from the primary studies' results that other attributes (like usability - 17.30%, reliability - 11.53%, and security - 11.53%) are also being investigated, and research effort have already been spent for their evaluation (see, e.g., [S5] for reliability).

Compared to Mairiza *et al.* (2010), a systematic review with 182 studies, resulting in 114 identified different types of NFPs, our investigation identified a smaller number of attributes. Our results are grounded on two main points: (i) we only considered studies handling SPL; and (ii) the QAs investigated are those from execution point-of-view, usually dependent on system behaviour, application domain and users' needs. There is an increasing number of systems built for applications where the user feedback and hardware restrictions are the most important requirements. In this scenario, execution NFPs are as important as functional requirements.

Table 3.4: Non-Functional Properties (NFPs)

1. Accuracy	14. Dispatch response time	27. Memory consumption	40. Scalability
2. Availability	15. Ease of use	28. Message size	41. Security
3. Available CPU	16. Effort	29. Minimum waiting time	42. Service time
4. Channel capacity/latency	17. Energy consumption	30. Performance	43. Severity of product failures
5. Comfort	18. Energy efficiency	31. Picture frequency	44. Space - main memory
6. Communication performance	19. Execution time	32. Position accuracy	45. Space - secondary storage
7. Completeness	20. Fault tolerance	33. Privacy	46. Speed
8. Confidentiality	21. File size	34. Range	47. Time - throughput
9. Convenient use of the system	22. Footprint	35. Recognition time	48. Usability
10. CPU consumption	23. Friendly user interface	36. Reliability	49. Usability for APT resident
11. Customer satisfaction	24. Integrity	37. Response time	50. Usability for VIPs
12. Cycle	25. Usability for handicapped	38. Safety	51. Weight
13. Data access security	26. Latency	39. Save energy	52. WLAN bandwidth

3.2.1.3 RQ1.3: What application domains are best covered by the existing approaches?

The 36 studies concern seven main application domains (i.e., automotive, database management systems, web-based, embedded, mobile, operating systems and end-users' domain-specific). Figure 3.4 shows an overview of the distribution of the studies based on their application domain. Each primary study considers one or more application domains.

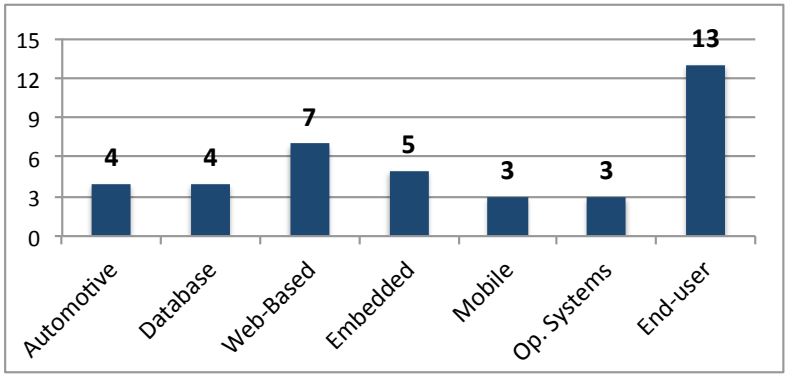

Figure 3.4: Number of studies by application domain

A mapping concerning each type of domain and its associated quality attributes is presented in a Venn diagram, Figure 3.5. This kind of diagram shows the logical relations between sets, which in this case shows the scattering of attributes among the respective domains. Each number in the diagram represent a quality attribute from the set of 52 different attributes, conform presented in Table 3.4. It is worth to mention that four attributes are not presented in the diagram (19, 21, 43 and 48), since that it was not identified in which kind of system they were investigated.

According to the diagram, only two types, from the eight types of general execution attributes (i.e., performance, security, availability, usability, scalability, reliability, interoperability and adaptability) discussed on Section 3.2.1.2 are typically considered in all kinds of systems: performance and reliability. For instance, these following performance sub-attributes, accuracy (1), energy consumption (17), communication performance (6), energy efficiency (18), available CPU (3), memory consumption (27) and response time (37) are present respectively in automotive domain, end-user applications for general purpose, web-based systems, embedded domain, mobile applications, database domain and operating systems. In this way, there are at least one performance (or reliability) sub-attribute in each kind of system.

Security is a general attribute considered in four types of systems (automotive, end-user, web-based and embedded) and usability is presented in three types (end-user, web-based and embedded). There was not a kind of sub-attribute present at the same time in the seven kinds of domain applications, but four of them, (27), (30), (36) and (37), belong to six different domains. The sub-attributes (27), (30) and (37) are from performance class and (36) is from reliability.

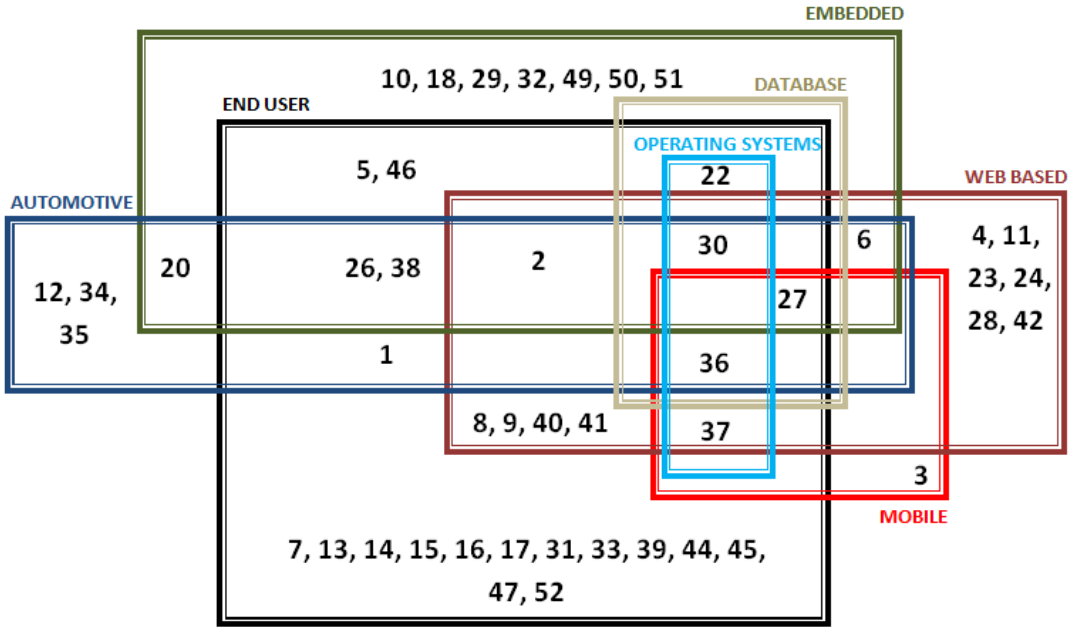

Figure 3.5: Domains and NFPs.

3.2.2 RQ2 - Available evidence

As described in Section 3.1, six different levels were used to assess the level of evidence of each identified approach. We can see from the primary studies' results that large majority of the 36 studies, about 53%, are academic work (L4); about 42% of studies show evidence obtained from demonstration or working out toy examples (L2), while about 5% of studies do not show maturity of their methods and tools (L1). No studies were found neither with evidence from expert opinions or observations (L3), from industrial studies (L5) nor industrial practice (L6).

The closest of an industrial study, S2, utilized specifications from a real automotive product line, Toyota automaker. However, the study used a very small set of requirements and resources, which does not constitute a real industrial study. They also state that the methodology, although it has not been validated in a real automotive system with large number of components, can be effective. S30 is another example of study that also used a small part of an industrial system for an academic study, in this case the Linux Kernel was the example.

3.2.3 RQ3 - Limitations of the existing support

The primary studies were categorized according to the quality assessment criteria presented in Table 3.1. These criteria correspond to (YES or NO) questions. Regarding questions related to

Reporting issue (i.e, RQ3.1, RQ3.2 and RQ3.3), all approaches were based on research with clear aims and suitable context's description. Thereby, these three criteria related to *Reporting* issue in general received a positive answer.

For the first question of the *Rigor* issue (RQ3.4), the results highlight, in general, that the studies present an appropriate design to address their goals. On the contrary, for RQ3.5 and other three questions related to *Credibility* and *Relevance* issues, discrepancies become more evident. Figure 3.6 shows percentage distributions of the answers (YES or NO) for these four questions (i.e., RQ3.5, RQ3.6, RQ3.7, RQ3.8).

For RQ3.5, the results show that few approaches (about 39%) perform a rigorous data analysis (e.g., by presenting experimental material, design and procedures). For the *Credibility* issue (RQ3.6), results show that about 58% of studies deal with the validation of their approaches. This result compromises the trustworthiness and usefulness of the approach and prevents the comparison with others approaches. On the contrary, the studies typically provide a clear statement of findings (RQ3.7), as well as a validation future work plan. However, in general, the studies do not provide guidelines to follow when practitioners reuse the approach (RQ3.8). This result highlights that there are open research challenges due mainly to the lack of rigor in validation or lack of flexibility for approach reuse.

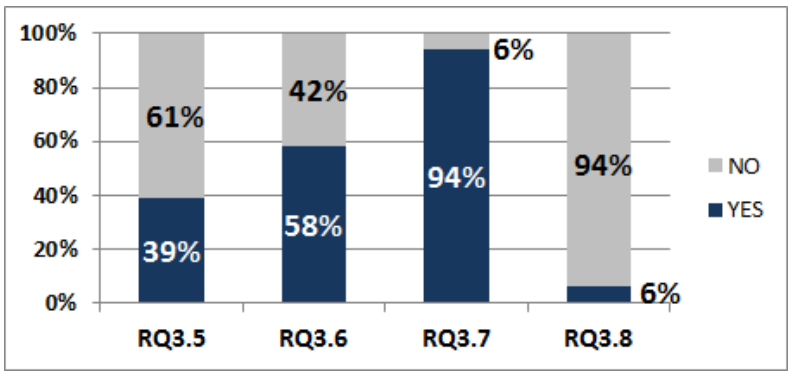

Figure 3.6: Percentage distributions of the answers for RQ3.5, RQ3.6, RQ3.7 and RQ3.8

3.3 Analysis and Discussion

This section discusses the key findings of our study, as well as validity threats in this review.

We identified three main categories of approaches that handle runtime NFPs in SPLs (see Figure 3.3). Each of the approaches identified in the review has its specific focus and context that it is appropriate for.

An aspect that made the analysis of each primary study harder was the fact that each study handled with NFPs in a different way, and this diversity of ways to analyze NFPs made the process of understanding and categorization of the founded QAs more complicated. The usability and meaning of each property had a huge variation from paper to paper.

In fact, to get better understanding of the development of the field, we provided a critical appraisal on the quality of the included studies, under established criteria for the maturity, the validation rigor, and the approach applicability.

Since the effort of measuring all products can be very costly and error-prone during quality analysis, predictions arise as a strategy to overcome this problem. **QP** approaches usually make use of three strategies: (i) modelling techniques in combination with domain experts' judgments and experiences (*model-based*); (ii) measurement of a small set of products *(measurement-based)*; or even a combination with both *(mixed-based)*. These strategies have positive and negative points. For example, for measurement-based approaches, producing real products is a time-consuming task and source-code is necessary. Approaches based on models avoid instantiating real products, but provide rough quality assessment. The choice between these approaches must take into account aspects such as type of QAs analysed, the development stage of interest and stakeholders' needs.

For **QE** approaches, this group provides implementation alternatives in order to minimize/ maximize NFPs according to customers preferences. **FS** approaches, although not providing an explicit quality analysis with discussions to predict or estimate NFPs, support optimizations by providing, in most cases, tools for automated feature selection during the product derivation process.

In the last years, the topic of definition and analysis of NFPs in SPLs has been studied in several communities (e.g., in the automotive domain [S2]). In fact, the trend of the temporal curve reveals an increase of the number of papers (see Figure 3.2). This is due to the fact that SPL principles have been proved to be widely applicable and a successful approach. However, notwithstanding the increasing interest and diffusion of these practices, grand research challenges still need to be addressed.

Although in recent years there was an increasing number of publications, in general only few researches involve real-world products (e.g., [S18][S19][S21]). Most primary studies used case studies as a proof of concept, as a way to validate or show their approaches in "practice". The methods are usually not properly applied and important details are missing, mainly related to experiment design and material, variables and analysis procedure.

General remarks in relation to the current state of the art are as follows: (i) the analysis of few NFPs is in general performed, and (ii) the SPL approaches considers the NFPs important for

the SPL practices. In particular, the outcomes confirm that NFPs are highly relevant properties in next generation computing applications. Moreover, tradeoff analysis among multiple competing and conflicting objectives should be supported, which is in general missing in the state of the art and practice today.

Validity threats. The main threats to validity are bias in our selection of primary studies, classification and data extraction. We developed a review protocol containing detailed information about RQs, search strategy, selection criteria, data extraction and quality assessment. The protocol was designed by the main researcher, and was then reviewed by other two researcher to check if each research step was appropriately planned. Although the protocol was revised several times until the authors have agreed on every point addressed, there was a risk of missing some relevant studies, not covered by our search strategy. To mitigate such a threat, we also carried out a manual search within the most important venues in the SPL engineering field. In addition, to handle threats surrounding the classification of NFPs, the authors undertook an in-depth analysis on every included primary study, in order to collect enough information to the review, and also discussed the results until reach an agreement on the classification. However, it is possible that our classification may not be similar to other in the literature. To ensure correctness of the data extraction, a template was proposed for a consistent RQs extraction. Furthermore, RQ3 was designed aimed at assessing the limitations of every included primary study. It is a means to improve the reliability and guide the interpretation of the findings.

3.4 Chapter Summary

The main objective of this systematic review was to obtain a holistic view of SPL approaches that have been reported regarding the analysis of runtime NFPs. We have identified 36 primary studies. Each of them share a lot in common, e.g. focus, goal and application context. We have extracted commonalities and summarized the studies into three main categories of themes.

This systematic review might have implications for both research and practitioners. The analysis of the primary studies indicates a number of challenges and topics for future research: (i) it should be provided support for the tradeoff analysis among competing NFPs both at domain engineering and application engineering levels; and (ii) it is also necessary to analyze dependencies between different kinds of NFPs and the SPL lifecycle-related practices.

The QAs identified in each approach, together with the application domains, served as input for the analysis, where performance attributes were the most commonly identified runtime properties, similarly to Mairiza *et al.* (2010). As aforementioned, almost 54% of the QAs are

related to the system performance.

We claim that addressing the highlighted challenges will require the contributions from researchers and industrial experts in different fields. We could include optimization formulation (e.g., several metaheuristic techniques with different characteristics could be adopted depending on the nature of application domain), system properties assessments, and experimentation on real world large SPL, considering realistic model parameter values.

Another observed aspect after performing this review was the lack of a systematic and uniform specification of NFPs for SPL, as previously indicated. Each study had its own way of describing the non-functional properties, where for the same attribute it was possible to find, for instance, different concepts and units of measure. Next Chapter addresses this problem by proposing a framework to define and specify different kinds of NFPs in a systematic way. This framework is composed by five main tasks, which are responsible by providing an uniform structure to integrate NFPs in SPLE.

4

An NFPs Framework for SPL

One of the key challenges still remaining in Software Product Lines Engineering (SPLE) is the management of Non-Functional Properties. In the previous chapter, Chapter 3, we performed and presented the results of our systematic literature review, and as one of the outlined points we highlighted that specifications of NFPs regarding to features/products is either not defined or not uniform. Each paper handles with NFPs in a different and non systematic way.

In order to address the diversity in NFPs specifications for SPLE, this chapter presents a *NFPs Framework*. It was proposed to allow a clearer integration process of NFPs with features and products, since it aims an uniform and systematic specification of NFPs for SPL. For "framework", we mean a detailed model of attribute specification and documentation addressed to SPL context.

The remainder of this chapter is structured as follows: Section 4.1 gives an overview of the NFPs Framework for SPL; Section 4.2 presents the work that our framework is based on; Section 4.3 describes the main task of our NFPs Framework for SPL; and finally, Section 4.4 presents the chapter summary.

4.1 Overview

The configuration of NFPs in SPL has been considered a challenging task. Generally, the NFPs of a complex system are the result of the interaction of many features, which makes them very difficult to be configured. Furthermore, the explicit definition of NFPs during software configuration is not a common practice (Sincero *et al.*, 2007, 2010).

In the literature, there are some approaches and tools, such as Sincero *et al.* (2007, 2009, 2010) and Villela *et al.* (2012), to automate product derivation considering NFPs. However, they present initial solutions in this direction and address few NFPs, mostly performance and/or

security. The goal of the *NFPs Framework* defined in this dissertation is not work with a couple of quality attributes, but is to provide a way to define any kind of attribute through a systematic specification, that is generic enough to be used in different NFPs approaches. The main idea is that even different approaches, such as Sincero *et al.* (2007) and Villela *et al.* (2012), can use the same framework as a basis for defining their attributes, such that the integration of attributes into the SPL process can be facilitated, besides more practical and effortless.

Thus, the framework intends to systematically provide additional information about features and products inside the SPL process, more specifically, during the generation of a new product (Product Derivation process). In this context, the *NFPs Framework* focuses on determining a way to enhance the product configuration to provide the necessary fundamentals to efficiently support, in a systematic way, the management of NFPs in product lines.

4.2 Related work

Sentilles (2012) proposed a framework for component-based embedded systems. It allows the specification of multi-valued and context-aware extra-functional properties[1]. Multi-valued means that an attribute may assume more than one valid value in the same development context; and context-aware represents the dependency that extra-functional property values usually have on their usage context. Capturing this dependence facilitates the reuse of extra-functional properties together with the component they describe.

According to Sentilles (2012), extra-functional properties are multi-valued and context-aware artifacts that must be integrated into component models and managed in a systematic way. Her work also highlights that the concept of multi-valued context-aware extra-functional properties is defined through the definition of four main terms: attribute type, attribute instance, attribute registry and metadata type. These definitions correspond to the foundations towards the systematic specification, management and integration of extra-functional properties in component-based development.

In her framework, extra-functional properties can be attached to selected architectural entities of component models. In addition, their concrete values can be compared and refined during the development process. The objective is providing an efficient support, possibly automated, for analyzing selected properties. The main contributions of the framework include: i) a study of the possible usage of extra-functional properties in component-based development, ii) a specification of multi-valued context-aware extra-functional properties, iii) an investigation

[1] Non-functional properties and extra-functional properties have the same meaning in this context.

of the necessary supporting mechanisms for specifying, managing, refining extra-functional properties, and iv) the implementation of an extensible prototype for the proposed solutions.

We proposed an adaptation of the Sentilles (2012) work for the SPL context. This adaptation was facilitated thanks the similarity that exists between Software Product Lines and Components based Systems. In order to conduct this adaptation, specific SPL needs were included, and an rearrangement of activities were performed, which resulted on five key tasks presented in the next section.

4.3 NFPs Framework Key Tasks

The main purpose of this conceptual framework is to systematically provide an uniform structure to handle, define and integrate NFPs in SPLE. Thus, some key tasks were proposed:

TASK 1: Definition of an Attribute
This is the starting point of our approach. The purpose of this task is to present the definition of a Quality Attribute (or only Attribute).

TASK 2: Definition of Attribute Type and Attribute Registry
An Attribute is composed by two main parts: the Attribute Type and the Attribute Instance. The purpose of this task is to show the specifications of the Attribute Types, besides presenting that they can be organized in the form of an repository, here called Attribute Registry.

TASK 3: Definition of Attribute Instance
The purpose of this task is to present the other part of an Attribute, the Attribute Instance, that represents the concrete values of an Attribute.

TASK 4: Definition of Attribute Value Metadata and Metadata Registry
The context of Attribute Instances is defined by a set of information that can help to understand how and under which conditions a given Attribute Instance was analysed for a given product-line. The purpose of this task is to show these conditions, here called Attribute Value Metadata, besides presenting the repository where they are organized, Metadata Registry.

TASK 5: Definition of Value Selection

As an SPL can have many different Attributes, and a same Attribute can assume several values, the purpose of this task is to show how to select only the Attribute Instances of interest for a given product.

4.3.1 Task 1: Formal definition of an Attribute

Improper handling are given to Non-functional Properties in SPL, they are usually defined and specified in an informal way (Rosa *et al.*, 2002; Liu *et al.*, 2010), which justifies a lack of appropriate methods to incorporate them at the SPL development phases. Additionally, the large amount of possible NFPs to consider in software systems is never-ending (Crnkovic *et al.*, 2005), and this list gets even bigger with the fact that a single NFP can have many possible ways of representation. In the following, we will interchangeably speak of attributes, properties and NFPs.

Accordingly, a formal way to specify NFPs that can be used throughout the SPL process is a remaining challenge. In order to address this problem and properly specify NFPs, the concept of attribute is defined as (Sentilles, 2012):

Definition 1. *An Attribute is defined as:*

$$Attribute = (Type,\ Value^*)$$

A NFP, also called *Attribute*, is here based on two definitions: (i) the *Attribute Type*, that define a class of NFPs, and (ii) the attribute instance, afore called *Attribute Value*, that refers to a given NFP value associated with a feature or product of a SPL-based design.

Thus, a NFP is represented through **only one** *Attribute Type* and **at least one** *Attribute Value*. Therefore, the *Attribute Value* has a particularity: an attribute can have more than one value, where the difference can be on different metadata and/or validity conditions. For this, Sentilles (2012) comes to the concept of *Multi-Valued NFPs*, i.e., attribute values (instances) can co-exist simultaneously. Multi-value is the ability of an attribute to have multiple values to cope with information coming from different contexts of utilization, obtained through different methods, or to compare a range of possible values to make a decision. For example, a NFP defined as *CPU Consumption* can have a estimated value obtained from early phases of development, and also a measured value after the product is completely developed. In this case, the *Attribute Type* which name is *CPU Consumption* has two instances: one for an estimated value and another for a measured value. This will be better discussed on Section 4.3.3.

Another way of reasoning is thinking that an specific *Attribute* (NFP) is composed by two main parts: the mandatory part, *Attribute Type*, and the variable part, *Attribute Value* or *Attribute Instance*. Each of these parts will be better detailed in the following sections.

4.3.2 Task 2: Formal definition of Attribute Type and Attribute Registry

This task is responsible for defining two primary elements of the NFP Framework: *Attribute Type* and *Registry*.

4.3.2.1 Attribute Type

The *Attribute Type* specifies all the commonalities shared by the instances of a given attribute. It details how a given NFP is represented, what data type is required for its values and how they should be manipulated. Thus, the *Attribute Type* provides a solid definition for the representation and usage of NFPs, defined as (Sentilles, 2012):

Definition 2. *An Attribute Type is defined as:*

$$\textit{Attribute Type} = (\textit{TypeID}, \textit{Description}, \textit{Attributable*}, \textit{Variability}, \textit{DataFormat}, \textit{Documentation})$$

TypeID

TypeID is a unique identifier for the *Attribute Type*. It represents a key that allows retrieving the corresponding *Attribute Type*. As it is an identifier, it must be unique in order to ensure that it is not possible to have more than one *Attribute Type* with the same *TypeID*, even if they have different *Description* or *DataFormat*.

The type identifier element (i.e. TypeID) is the key that allows retrieving the corresponding attribute type. In order to simplify, the name of the property is used as the unique identifier in the remaining of the dissertation, as illustrated in Table 4.1. Table 4.1 gives a representation of some attribute type specifications.

Description

Description offers in few words the meaning of the corresponding *Attribute Type*. We believe that only the *TypeID* is not enough to understand the goal of the attribute.

Attributable

Attributable represents the element kinds to which a NFP of type *TypeID* can be attached to. For SPL, not only a feature can have an additional information, but also products. It means

that the attributable elements are features and products. But, in the case of features, only leaf features are important to attach NFPs information. Leaf features are at the end of feature models and cannot be abstract features. For reasoning about non-functional properties in SPL abstract features are not relevant (Thum *et al.*, 2011). For products, we consider a combination of more than one feature as a product. Thus, our SPL attributable entities, features and products, are treated in the same way regarding the definition and usage of attributes.

Variability

As explained on Section 2.3, a feature can be part of a system or not. If it is always present in the SPLs products, the feature is mandatory, and if it may or may not be present in a product, the feature is named optional. Similarly, some NFPs will be mandatory (as they are common across the product-line) whereas some others will be optional. In this way, an *Attribute Type* which identifier is *TypeID* can be: (i) mandatory, i.e., necessary for all features and/or products, or (ii) optional, features or products can have it or not. For example, for the *Attribute Type* which *TypeID* = "*CPU Consumption*", and *Attributable* = "features", if the *Variability* = "Mandatory", all the features of the given SPL must have this attribute attached to. But, if the *Variability* = "Optional", the application engineer has to decide if that product will have this attribute or not, which means that it has to be decided if this attribute is interesting or not for a given product .

DataFormat

DataFormat corresponds to the data type used to represent the values of an *Attribute Type*. Thus, it can be defined as:

Definition 3. *An DataFormat is defined as:*

$$DataFormat = (GeneralFormat, SpecificFormat)$$

The *GeneralFormat* defines that an NFP can be qualitatively specified or quantitatively measured in the context of SPL. Qualitative properties are NFPs that can only be qualitatively described using ordinal scale. There is no metric from which we can retrieve quantifiable measures. Examples of this class are: Reliability, Security, Trustability, Availability, Usability, Integrity and Completeness (Siegmund *et al.*, 2012).

The quantitative format defines properties that can be measured on a metric scale. With this class of attributes it is possible to compute to which extent a feature influences an NFP. An example is the footprint of a feature, which can be measured per implementation unit (Siegmund *et al.*, 2008). Other examples are: Accuracy, Price of a Feature, Adaptability, Interoperability

and Modularity. Inside this same class there are other NFPs that we are not able to quantify the influence of individual features on them. Usually, such properties emerge when a product is executed, and usually requires to execute and to measure the NFP for a product, for example, by running benchmarks. Examples are: Performance, Response Time, Resource Behavior (e.g., energy and memory usage), Bandwidth.

Based on Sentilles (2012), the *SpecificFormat* has to support:

- Qualitative types: high, medium, low.
- Primitive Quantitative Types: integer, float, etc.
- Structured Types: arrays, etc.
- Complex Types: external models, images, formulas, etc.

Documentation

Documentation describes the NFPs in natural language. It must provide enough information in order to clarify the meaning of the given *Attribute Type* with more details that other categories did not cover. This *Documentation* can be attached to the *Attribute Type* in the form of a file, such as, pdf or Microsoft Word files.

4.3.2.2 Attribute Registry

Since several attributes have already been defined, we need to store that set of *Attribute Types* in a kind of repository, in order to organize them and to ensure the uniqueness of each one. A way to do so is to keep a *registry* of *Attribute Types*, which contains the pool of NFPs that can be assigned to features and/or products (Sentilles, 2012).

An *Attribute Registry* is a set of all *Attribute Types* available in a given context of a product-line or in the supporting developing environment for a given product-line. Table 4.1 presents an *Attribute Registry* for a given SPL with two examples of *Attribute Types*.

It's worth to highlight that the *Attributes Registry* does not have measured or even estimated values for each defined *Attribute Type*. It only has the specifications of each attribute, including the definitions for its values. For example, for the *Attribute Type* which *TypeID* is `CPU Consumption`, the value must be represented in Integer, as described in Table 4.1. The *Attribute Type* has the specification and the instances of `CPU Consumption` will contain the

Table 4.1: Attribute Registry with two Attribute Types

TypeID	Description	Attributables	Variability	DataFormat	Documentation
CPU Consumption	Computer's usage of processing resources, or the amount of work handled by a CPU	Products	Optional	{Quantitative, Integer}	CPUconsump.pdf
Memory Usage	Amount of main memory that a program uses or references while running	Features, Products	Mandatory	{Quantitative, Float}	Musage.doc

concrete values. Each *Attribute Type* can have more than one value calculated in different ways, which means they can be instantiated as much as necessary. Next section will detail how to instantiate an *Attribute Type*.

4.3.3 Task 3: Formal definition of Attribute Instance

The Definition 1 presented that an *Attribute* is composed by the *Attribute Type* and *Attribute Value*. Since the previous section, Section 4.3.2, discussed about the *Attribute Type* specifications, this present section focuses on the *Attribute Value* or *Attribute Instance*. The instance of an attribute is just to add a concrete value to it, so that *Attribute Value* and *Attribute Instance* have the same meaning for us.

Definition 4. *An Attribute Instance is defined as (Sentilles, 2012):*

$$Instance = (TypeID, Data, Metadata, ValidityConditions)$$

A specific instance of an attribute must have: (i) *TypeID*, an identification for its type; (ii) *Data*, concrete value for the NFP; (iii) *Value Metadata*, complementary information on data that allows to distinguish among them; and (iv) the conditions under which the value is valid, *ValidityConditions*. Figure 4.1 presents an excerpt of the NFP Framework meta-model with the aforementioned structures (Sentilles, 2012), and Table 4.2 presents two examples of instances for each *Attribute type* from Table 4.1.

TypeID

TypeID is the unique identifier of the corresponding *Attribute type*. It identifies which *Attribute type* the instance correspond to.

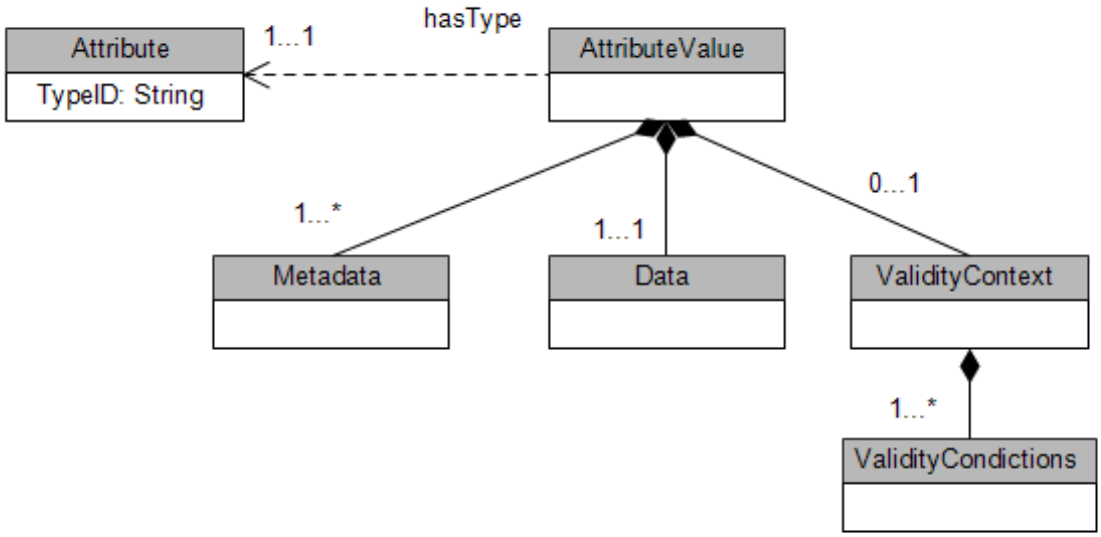

Figure 4.1: AttributeValue Meta-model

Data

Data contains the concrete value for an specific *Attribute Instance*. This value must comply the *DataFormat* specified for the corresponding *Attribute Type*. For example, Table 4.1 presented that for the *Attribute Type*, which *TypeID* is "CPU Consumption", that the *DataFormat* must be a integer number. In this way, two instances of "CPU Consumption" are detailed on Table 4.2, which integer *Data* are: 25% and 45%.

Metadata

Besides capturing the concrete value of an *Attribute Type* in order to create an instance for it, it is also important to identify the context in which the corresponding value has been obtained. *Metadata* represents a set of information responsible for presenting the context in the moment that an attribute value is gathered.

Table 4.2 presents three examples of *Metadata*: CreationTime, Source and Complexity. The detailed structure for the components that are part of the *Attribute Value Metadata* will be presented on Section 4.3.4.

ValidityConditions

For Software Product Lines, reusability is a key concept. A feature must be designed and

implemented looking forward to get a good level of reusability, being able to be reused in as much software product as necessary. This way, Non-Functional Properties (or only *Attributes*) could be reusable too, such a way to be used in more than one software product. This means that the validity of the *Attributes* information must still be accurate in the new context in which the feature is reused. Thus, theses specifications of context restrictions are referred as *ValidityConditions*.

ValidityConditions correspond to a set of restrictions of the applicability context of *Attribute Instances*. For instance, features interactions are an interesting example of a *ValidityCondition*. Sometimes, a NFP is the result of the combination of many features, where their arrangement can modify (increase ot decrease) the specific value of a non-functional property.

Examples of other restrictions are: specification of usage profile, constraints on the underlying platform, interaction towards others attributes, and so on. Figure 4.2 shows an excerpt of the NFPs Framework meta-model that represents those structures.

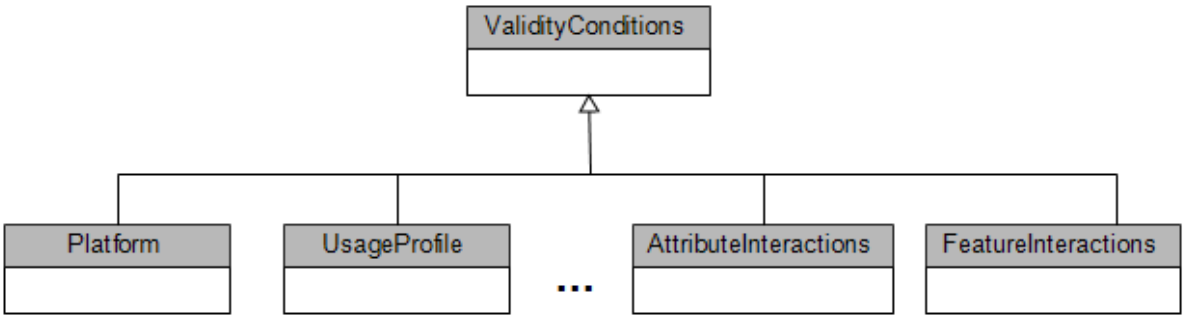

Figure 4.2: Validity Conditions Meta-model

Table 4.2: Attribute Instances

TypeID	#	Data	Metadata	ValidityConditions
CPU Consumption	1	25%	- CreationTime = “11.06.14-08:24” - Source = “Simulation” - Complexity =“Low” - etc.	- Feature Interactions = {F1, F5, F12}
	2	45%	- CreationTime = “11.06.14-18:40” - Source = “Measurement” - Complexity =“High” - etc.	- Feature Interactions = {F5, F3}
Memory Usage	1	15kB	- CreationTime = “10.06.14-22:15” - Source = “Measurement” - Complexity =“Medium” - etc.	- Feature Interactions = {F7, F3}
	2	22kB	- CreationTime = “11.06.14-10:30” - Source = “Measurement” - Complexity =“Medium” - etc.	- Feature Interactions = {F7, F20}

4.3.4 Task 4: Formal definition of Attribute Value Metadata and Metadata Registry

This task defines the concepts and details of *Attribute Value Metadata* and *Metadata Registry*. *Metadata* corresponds to any context information that contributes to understand how and under which conditions a given Attribute Value was analysed for a given product-line.

4.3.4.1 Attribute Value Metadata

Similarly to *Attributes*, the concept of *Attribute Value Metadata* or only *Metadata* also distinguishes between *Metadata Type* and *Metadata Instance*. The *Metadata Type* defines the commonalities of context shared by all the instances of an *Attribute Type*, and *Metadata Instance* is the value of an metadata that follow the specifications defined by the given *Metadata Type*.

For example, the *Metadata Type*, which ID as "Source" can be defined as *Source* = {"Measurement", "Estimation", "Simulation"}. When we create an instance of "Source" for a given instance of an *Attribute Type*, we have to choose just one kind of the specified "Source". Table 4.2 presents, for example, for the *Attribute Type* "CPU Consumption", two *Attribute instances*, where one has *Source* = "Simulation" and the other has *Source* = "Measurement". It is worth emphasizing that *Attribute Value Metadata* is part of an *Attribute Value/Instance*, as defined in our previous Section 4.3.3, and contributes to define a context of a given *Attribute Value*.

Definition 5. *An Metadata Type (MetaType) is defined as:*

$$MetaType = (MetaID,\ ValueFormat,\ Variability,\ Description)$$

MetaID

MetaID is a unique identifier for the *Metadata Type*. For simplicity, it is the name of the metadata.

ValueFormat

ValueFormat specifies the types used to represent the values.

Variability

The *Variability* specifies if the *Metadata Type* is mandatory or optional for an Attribute Value.

Description

Description corresponds to a simple description of the *Metadata Type*.

4.3.4.2 Metadata Registry

Alike *Attribute Type* that needs a repository in order to organize the Attribute *TypeIDs* for a given SPL, *Metadata Types* also need to be stored. The *Metadata Registry* follows the same idea of the *Attribute Registry* and contains a set of metadata that can be assigned to *Attribute Values/Instances*, which in turn can be assigned to another set of *Attribute Types*. Figure 4.3 presents an example of this relation among attributes, values and metadata.

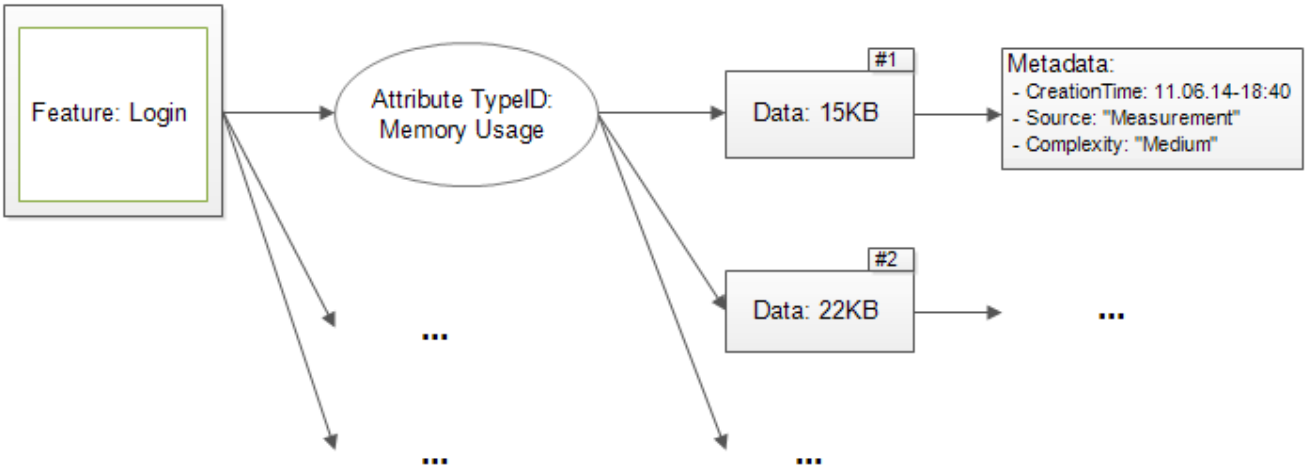

Figure 4.3: Example with a feature, Attribute Type, Attribute Instances and Metadata

In Figure 4.3 we have an example of a classical kind of feature of a given SPL for web systems, here named "Login". This feature can have a couple of NFPs (attributes), such as "Memory Usage", present in this figure. In this example, we present two *Attribute Instances* (values) for "Memory Usage": #1 with 15kB and #2 with 22Kb. For the first instance, #1, we have a set of metadata types with its values, such as "CreationTime: 11.06.14-18:40" and "Complexity: Medium". Table 4.3 lists more examples of metadata types and its specifications.

After specifying: (i) the formal definition of an Attribute (Task 1 Section 4.3.1), (ii) Attribute Type and Attribute Registry (Task 2 Section 4.3.2), (iii) Attribute Instance (Task 3 Section 4.3.3), and finally (iv) Attribute Value Metadata and Metadata Registry (Task 4), Figure 4.4 presents the complete meta-model of the NFPs Framework for SPL, based on (Sentilles, 2012).

Table 4.3: Metadata Registry

MetaID	**ValueFormat**	**Variability**	**Description**
CreationTime	TimeStamp	Mandatory	Creation data and time of the Attribute Value
ModificationTIme	TimeStamp	Optional	Modification data and time of the Attribute Value
Version	Integer	Mandatory	Version of the Attribute Value
Source	{“Estimation”, “Measurement”, “Simulation”}	Mandatory	The way a value is analysed
SourceTool	String	Optional	If any, the tool responsible to analyse the Attribute Value
Complexity	{“Low”, “Medium”, “High”}	Mandatory	Complexity to get an Attribute Value
Comment	String	Optional	Any other additional comment

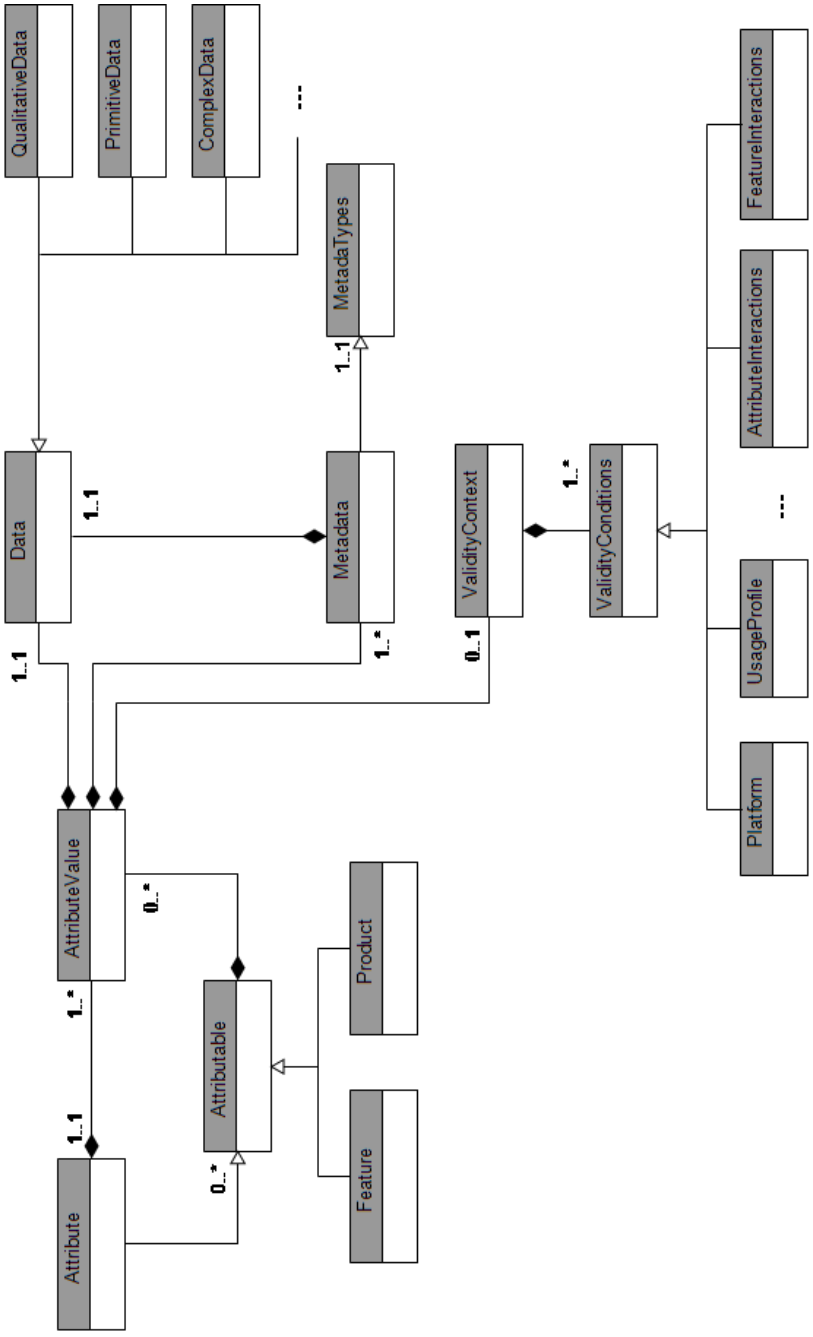

Figure 4.4: AttributeValue Meta-model

4.3.5 Task 5: Formal definition of Value Selection

As a same attribute can have many different values (Section 4.3.3), for a given SPL it is possible that we have a large amount of attribute instances. In this way, this task is responsible for detailing how to perform a selection of values such a way to get only the attributes of interest for a given product.

In order to make this selection of attribute values easier, some principles could be follow, as for example the principles described by Conradi and Westfechtel (1998) for Software Configuration Management (SCM). According to them, there are two types of versioning elements:

- **Versions**, also called revisions, identify evolution of an item over time. Besides a latest version of an item, an older one can be used too.
- **Variants** allow the existence of different versions of the same item at the same time.

Both concepts, Versions and Variants, can be used to the management of multiple attribute values in SPL, instances of *Attribute Types*, which are distinguished through *Metadata* and *ValidityConditions*.

Attribute instances can be obtained through the use of *matching conditions*, formally specified in Definition 6. A matching condition is derived from the the set of *Metadata*, *ValidityConditions*, or a list of predefined keywords, such as: Latest, the latest version; TimeStamp, the latest version created before the specified date; and VersionName, a particular version designed by a name.

Definition 6. *Formally, a matching condition is defined as (adapted from Sentilles (2012)):*

$$
\begin{array}{lll}
\langle Condition \rangle & ::= & \langle Cond \rangle \mid \langle KeyCond \rangle \\
\langle Cond \rangle & ::= & \langle TypeCond \rangle \langle Op \rangle \langle ValueCond \rangle \\
\langle TypeCond \rangle & ::= & \langle MetID \rangle \mid \langle AttID \rangle \mid \langle NameAtt \rangle \\
\langle OP \rangle & ::= & \text{“} = \text{”} \mid \text{“} \neq \text{”} \mid \text{“} < \text{”} \mid \text{“} \leq \text{”} \mid \text{“} > \text{”} \mid \text{“} \geq \text{”} \\
\langle KeyCond \rangle & ::= & a\ set\ of\ predefined\ keywords \\
\langle MetID \rangle & ::= & existing\ metadata\ type\ identifiers \\
\langle ValueCond \rangle & ::= & \langle Value \rangle \mid \langle KeyCond \rangle \\
\langle Value \rangle & ::= & values \\
\langle AttID \rangle & ::= & existing\ attribute\ type\ identifiers \\
\langle NameAtt \rangle & ::= & name\ of\ an\ Attribute\ Type
\end{array}
$$

A sequence of matching conditions combined with AND or ELSE operators is called *Configuration Filter* (Sentilles, 2012). It provides a better control over the values to retrieve

by using one or more matching conditions. In the value selection point of view, *Metadata* and *ValidityConditions* are equivalent. The Configuration Filter defines constraints over *Metadata* and *ValidityConditions* in the same way, as specified in Definition 7.

Definition 7. *Formally, a Configuration Filter is defined as (Sentilles, 2012):*

$$
\begin{array}{lll}
\langle Filter \rangle & ::= & \langle ConditionOr \rangle \mid NULL \\
\langle ConditionOr \rangle & ::= & \langle ConditionAnd \rangle \mid \langle ConditionAnd \rangle \ \textbf{ELSE} \ \langle ConditionOr \rangle \\
\langle ConditionAnd \rangle & ::= & \langle Condition \rangle \mid \langle Condition \rangle \ \textbf{AND} \ \langle ConditionAnd \rangle
\end{array}
$$

The Configuration Filter can be applied on the set of features or the entire product from a given SPL. The ELSE conditions are neatly tested until a subset of attribute instances is selected, as follow (Sentilles, 2012):

$$
\begin{array}{llll}
\text{Condition}_1 & \textbf{AND} & Condition_2 & \textbf{ELSE} \qquad (line1) \\
\text{Condition}_3 & \textbf{AND} & Condition_4 & \textbf{ELSE} \qquad (line2) \\
\ldots & & &
\end{array}
$$

The ELSE condition of line 1 is examined first. If there is no attribute value corresponding to the matching condition, then the second ELSE of line 2 is examined, and so on until either values are found or there is no value that corresponds to the configuration filter (Sentilles, 2012). An example of Configuration Filter can be expressed as follows:

(Source = Estimation) **AND** (Platform Operation System = Ubuntu 12.10) **ELSE**
(Label = “Release 2.0”) **ELSE**
Latest

This example means that we would like to: select attribute values that have been assessed by estimation for Ubuntu 12.10, or alternatively, values which have been defined for the release 2.0. In case of no value was found that corresponds to these criteria, the latest values can be selected.

4.4 Chapter Summary

This Chapter presented the details and specifications of the *NFPs Framework* for SPL. As mentioned on the previous sections, the framework was inspired on the one developed by Sentilles (2012) for component-based embedded systems. The main difference between their work and ours is the insertion of intrinsic characteristics of SPL, such as features and variability.

In order to better explain the NFPs Framework for SPL, it was divided in five main tasks: Task 1 - for the definition of an quality attribute; Task 2 and Task 3 - for definitions of the parts of an attribute, respectively attribute type and attribute instance; Task 4 - for the definition of metadata, which represents the context of an attribute value; and finally Task 5 - for understanding how a value can be selected from the set of several values that a given SPL can contain.

The Attribute Type (Section 4.3.2) has the specifications of an attribute, and an Attribute Instance (Section 4.3.3) is an Attribute type with concrete values. However, a value of an attribute must take into account the context which it was analysed. For this context we call Attribute Value Metadata (Section 4.3.4). It contains a set of information that help us to understand under which context a given Attribute Value was analysed. This set of information makes possible the reusability of a given Attribute Value in another product.

During the derivation of a new product from a given SPL, the application engineer can find a huge number of Attribute Values, and he needs a way to filter them in order to only analyse the ones of interest. In order to do it, a Configuration Filter (Section 4.3.5) was defined with the goal of assisting the engineer during this process. With the filter, he can insert his preferences in the way of *matching conditions* and only filter the most required Attribute Values.

Next Chapter aims at presenting how this *NPFs Framework* can be used in a reuse approach which goal is to systematically integrate the reuse of NFPs values (attribute instances) inside the SLP life-cycle.

5

An Approach of Non-Functional Properties Reuse in SPL

After creating a feature model and implementing features during the domain engineering phase, products are derived in the application engineering phase by selecting a set of features and deselecting irrelevant ones in the feature model, and instantiating reusable assets according to the requirements of a target application (product derivation process).

However, sometimes selecting a set of features for a product that corresponds only to functional requirements is not enough, but also non-functional properties should be satisfied. An application engineer may want to create a software product by selecting features with good values of NFPs based on the stakeholders' needs, in order to select the variant that better fits into the stakeholders' requirements. For example, one stakeholder may ask for a product with high security and high performance, but with cost lower than $800; and can mention that the performance is more important than security.

One of the SPL main objectives is the reusability of assets. Since features can be reused among many different SPL software products, analysis of non-functional properties could also be reused among them, as a way to avoid unnecessary re-analysis. If two products are very similar according to the set of features, usability purpose and application context, it is possible that they have very similar NFPs values too. In this way, NFPs values may not need to be calculated twice.

In order to provide this reusability of NFPs values, this Chapter describes our **NFPs Reuse Approach**, which intends to reuse previous NFPs analysis in order to minimize the effort of performing a new analysis. The idea is to integrate the *NFPs Framework* inside the SPL life-cycle, providing a systematic reuse of NFPs values during the product derivation process. This chapter is structured as follows: Section 5.1 presents an overview of the Reuse Approach;

Section 5.2 discusses about related work; Section 5.3 shows the necessary steps to perform the approach; and Section 5.4 presents the chapter summary.

5.1 Overview

A general view of a feature model configuration process in the application engineering phase is showed in Figure 5.1, where stakeholders consider requirements over functional and non-functional aspects. The application engineers can formulate requirements in terms of constraints and preferences and make configuration decisions to select proper features.

Figure 5.1 represents the same product derivation process shown on Figure 2.5, but under a simpler and general view where the only difference is that now we are adding a new asset/artifact, the A-base. It is responsible for storing NFPs already specified for that target SPL. In accordance with *NFPs Framework* presented on Chapter 4, this database contains the attributes with its values, also called attribute instances.

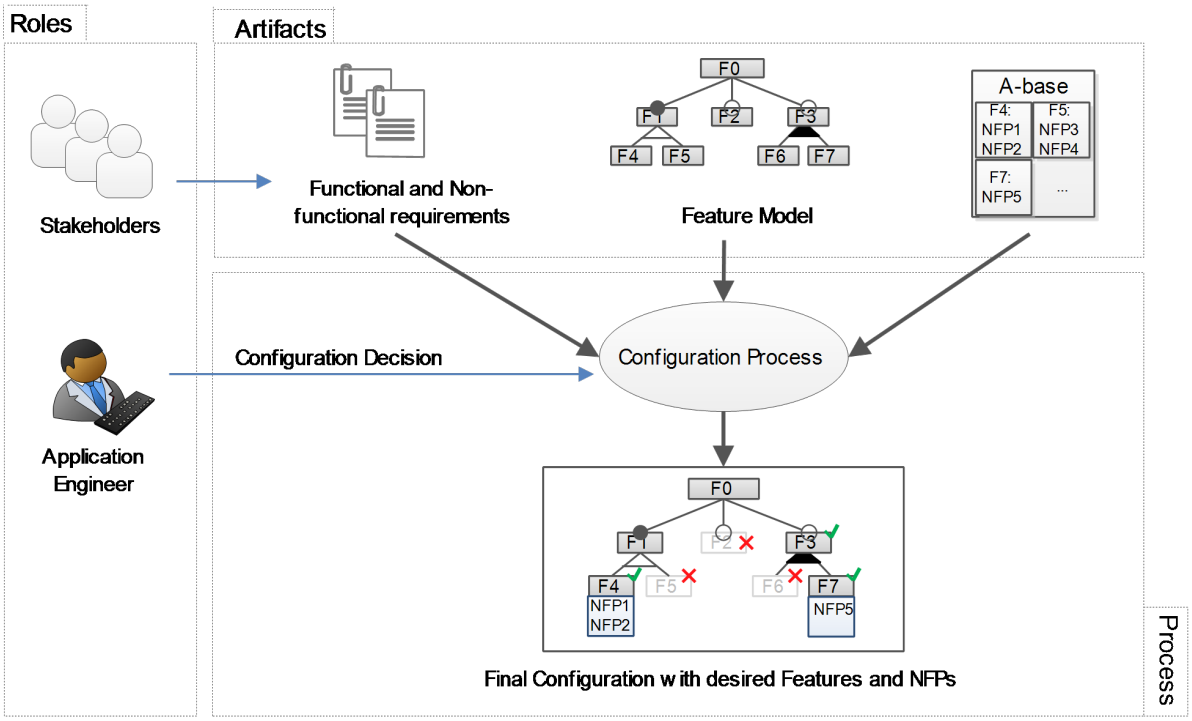

Figure 5.1: Feature model configuration process (adapted from Asadi *et al.* (2014)).

During the configuration process, after the stakeholders had defined the functional and non-functional requirements, it is up to application engineer to define the product that better fits

into the stakeholders' specifications, but now based on the feature model and also in a repository (A-base) of NFPs values, where he can analyse if there are values that are appropriated for the newest assembled product. In this way, during the configuration process, the application engineer has the option to analyse the features based on its NFPs values. These NFPs values can help the engineers to generate the best product, and in the final of the process, the feature model will be similar to that one in the Figure 5.1, where the engineer is aware about the properties of that specific product, such as the products' strengths and weaknesses.

5.2 Related Work

Cicchetti *et al.* (2011) presented concepts and mechanisms that allow to automatically discover whether a property value is still valid (using preconditions) when related components evolve. Their work discussed the evolution management of NFPs for component-based embedded systems. According to them, analysis of NFPs are time consuming and often difficult tasks, and for this reason reuse of the results of the analysis is as important as the reuse of the component itself.

Cicchetti *et al.* (2011) work proposed to mitigate the problems that arising when managing NFPs in evolving scenarios. They anticipate the impact analysis of the changes conducted to the system as early as possible, by detecting modifications at the modeling level and providing corresponding validation responses.

Baumgart *et al.* (2012) proposed a work for reuse of functional software certification. For automotive domains, electronic systems implement safety critical functionality in vehicles where the safety certification process is a time consuming task. According to them, this process represents a huge part of the expenses of any safety-critical product development project.

They believe that a bottom-up-approach not only could focus on reuse of the components, but could also focus on reuse of their safety evidence. However, the researchers mention that achieving an efficient functional safety certification where time consuming tasks do not have to be repeated for very similar subsystems is a challenge.

Baumgart *et al.* (2012) work is an initial study that only points out challenges and a preliminary analysis towards a viable solution. They identify three different types of reuse of functional software certification. In the first type, a reuse is completely possible, since the component had been used and certified in an exactly similar configuration and environment. For the second type, a complete reuse is not possible and the component must be changed in order to fit it in the new context. For such components, the safety certification effort is typically less than

the original certification effort. And for the last type, the reuse of the component is not at all possible. The functional safety effort of the new component is typically at least as high as the original functional safety certification effort for the original component.

We generalized the Cicchetti *et al.* (2011) idea and defined an approach where the reuse of analysis of NFPs can be used for any purpose of a general product line, where SPL stakeholders just got to be interested in evaluating NFPs. We elaborate on Baumgart *et al.* (2012) work, in order to discuss and describe for the **NFPs Reuse Approach** in which situations the reuse of a NFP value can be made.

5.3 Additional steps for the standard SPL derivation process

Achieving an efficient reuse approach, where time consuming tasks do not have to be repeated for very similar (sub)systems, has a set of implications. For instance:

- **Question 1:** Are there values in the beginning of the SPL implementation?
- **Question 2:** Among the available values, how to filter the ones of customer interest?
- **Question 3:** How to know if the gathered values (from other products) apply to the new product that has just been derived?
- **Question 4:** If there are no appropriated values, what to do?

In order to answer these questions and provide a systematic reuse approach of NFPs values, a *NFPs Reuse Approach* was proposed, which adds a couple of steps to the standard SPL process, as showed in Figure 5.2. The steps discuss about how to:

- **Step 1 -** populate the A-base with the attribute instances;
- **Step 2 -** configure the NFPs Filter in order to search for values in the A-base;
- **Step 3 -** analyse the filtered values (Reuse Diagnosis activity) ;
- **Step 4 -** create new instances if there are no appropriate values for the derived product.

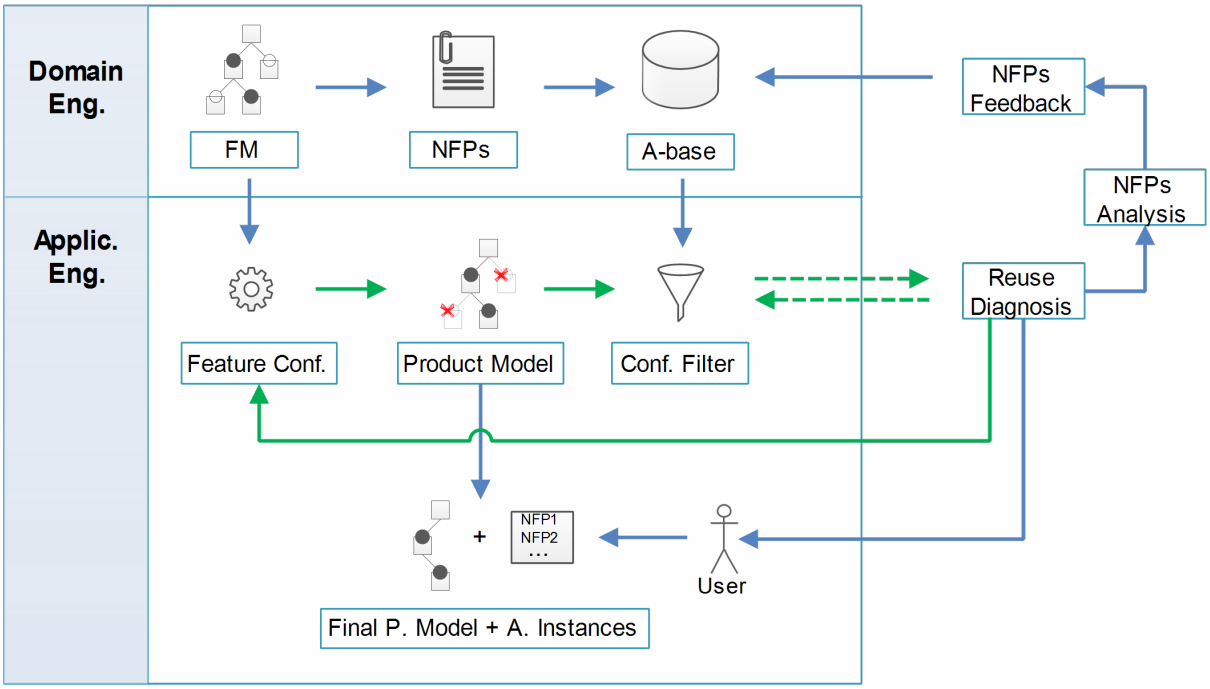

Figure 5.2: Approach to derive a product aware of NFPs

5.3.1 Step 1: Populating the A-base

Figure 5.2 shows our approach of NFPs reuse spread over two main SPL process: domain engineering and application engineering. In the domain engineering phase, the SPL implementation units are designed and developed according to the features specified through the stakeholders' requirements. Still in this phase, the NFPs for the target SPL also have to be identified. We consider that NFPs can be identified in two moments: (i) for the entire SPL during the domain engineering, where NFPs are common for all products; and (ii) per product during the application engineering, in the course of the product derivation new NFPs can be required for a specific product.

Thus, as during the domain phase it is possible to identify NFPs, we propose the creation (and population) of the registries, defined in the *NFPs Framework*: Attribute Registry (Task 2, Section 4.3.2) and Metadata Registry (Task 4, Section 4.3.4). These registries can be seen as repositories that describe how each identified NFP must be analysed, besides how and under which conditions its values have to be measured.

The Attribute Registry should contain all the attributes specifications for the given SPL. This registry has the commonalities shared by the instances of each attribute, as described on Definition 2 and exemplified on Table 4.1. On the other hand, the Metadata Registry consists of

a set of characteristics that defines the context in which the attribute values are obtained. Table 4.3 presented a couple of *Metadata Types* examples that can be used for product-lines.

Still during the domain engineering phase, and after implementing the product-line necessary features, the **Attribute Values Base (A-base)** must be created. It will contain all the attribute instances (values) for the NFPs of the target SPL and future instantiated products. Examples of instances are presented on Table 4.2. Even in this phase, it is possible to analyse some NFPs through estimations by, for example, qualitative measures. But, if there are experts with a large knowledge in analyzing software quality aspects, they can estimate NFPs values with percentage or more accurate quantitative means. In this way, even during the domain phase, when the SPL assets are being built and there are not derived products, it is possible to populate the A-base with some NFPs instances. These instances must necessarily follow the specifications from the registries (Figure 5.3).

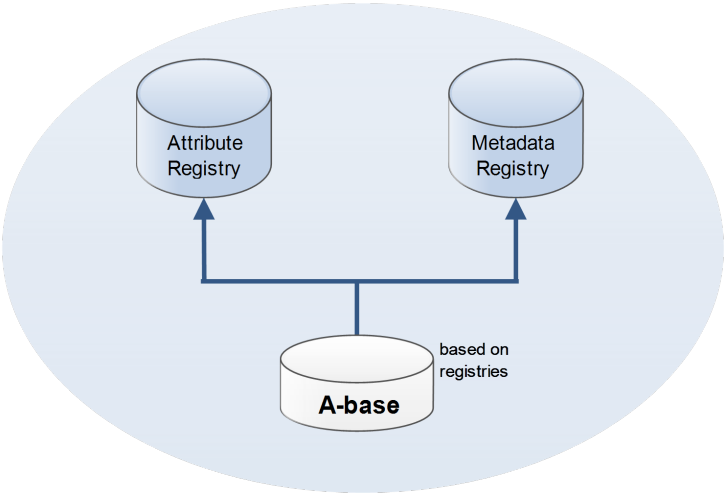

Figure 5.3: A-base

Population activities of registries and A-base, which correspond to the Step 1, can be visited during any moment of the SPL life-cycle. This means that, not only during the domain engineering, but also on application engineering, new insertions or modifications can be made. For example, either a new Attribute Type can be inserted into the Attribute Registry, or a new Metadata Type into the Metadata Registry, or a new Attribute Instance (value) into the A-base.

Another fact about this step is that we can easily associate it with the key activities from Product Derivation process, described in Section 2.2. Before selecting the desired assets to assemble a new product, the product derivation process defines some activities grouped as "Preparing for derivation" group. The Step 1 can be considered as part of this group, since it is responsible to prepare new assets: attribute and metadata repositories, and the A-base, which

contains the NFPs values intrinsic to the SPL. These assets will be part of and also support the "Product Derivation/Configuration" group, second group of activities of the product derivation process (Figure 2.5).

5.3.2 Step 2: Configuring the NFPs Filter

After the "Preparing for derivation" group, the next set of activities is responsible for assembling a new product according to the stakeholders' requirements (here considered functional and non-functional properties). This configuration happens during the application engineering process.

In this way, as showed in Figure 5.2, through the **FM** (feature model) a new product begins to be configured by choosing the features that better fits into the target requirements, represented by the **Feature Conf.** (feature configuration) activity. Based on this configuration, a new model is generated only with the desired features, the **Product Model**.

Once the product model has been generated, a new asset created to support the NFPs Reuse Approach, the **Conf. Filter** (configuration filter), needs to be defined such a way to obtain the NFPs values. One value is related to one attribute, and a same attribute can have many different values. The filter idea is to search for NFPs values of stakeholders' interest among all the values of a given SPL contained in the A-base, mainly through conditional expressions that limit the search space.

The filter should reflect the desire of clients about the specified product. It must be created using the matching conditions, which represent restrictions on metadata and validity conditions present in the attribute values specification. The filter can be expressed for one condition or a combination of them using operators like AND or ELSE, as it was specified in details on Task 5, Section 4.3.5, more precisely on Definition 7.

As many conditions can be part of a filter, after applying it on the values presented in the A-base, it is possible that more than one value is filtered, from the same attribute type or different types. For example, if we use the configuration filter example described on Task 5 for the attribute instances of Figure 5.4, one value (or instance, both means the same) that correspond to the attribute type `MemoryUsage` (#1), and one from `CPUConsumption` (#4) will be selected, since the filter looks for NFPs values that were *estimated* and under the *Ubuntu 12.10* operational system.

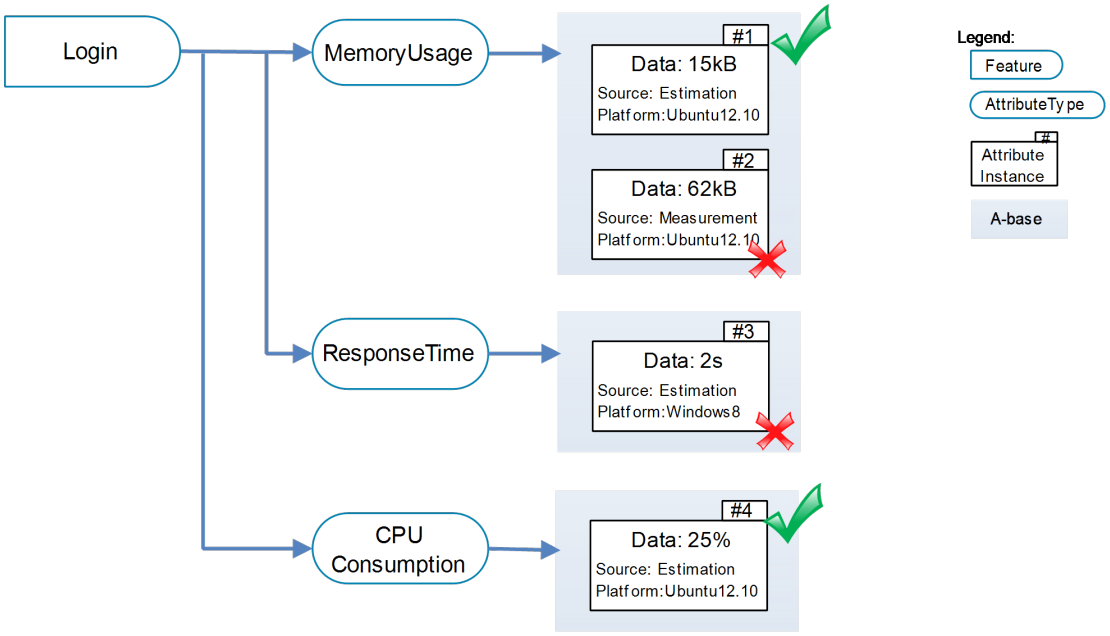

Figure 5.4: Example of a filter application

5.3.3 Step 3: Reuse Diagnosis activity

Although the filter defined by the user (developer, engineer, etc.) selects only the values that conforms to the matching conditions, many values can be selected. The user must analyse and decide if they are applicable to the product or not. For this activity of decision, we named **Reuse Diagnosis** and it is responsible for managing the attributes values retrieved after applying the filter.

At this point, some decisions must be taken by the user in order to provide the final model annotated with the NFPs values. To do this, the user must decide what is the case in which the retrieved values belong to:

- Case 1: the value is useful in the current development context.
- Case 2: the value is not directly applicable in the current development context, but it would be interesting to use it with some adaptations.
- Case 3: the value is not at all applicable into the current development context, or no value was filtered.

5.3.3.1 Case 1: the value is useful

If Case 1, the **Final P. Model** (final product model contains only the features chosen to the current product) is generated and annotated with the attribute values from the **Conf. Filter**, the **A. Instances** (attribute instances). Thus, a product is assembled where the users are aware of NFPs for that specific product. The next step, if Case 1, comprises the third group of key activities of the product derivation process, "Additional Development/Testing", as discussed in Section 2.2. Since, the product was defined, it must be checked and tested to certify that it meets the initial requirements.

Case 1 represents an ideal case, where the filtered values are adequate and correspond to the features of the product, and mainly to the goals specified by the stakeholders/clients. Corresponding to the features means that all the necessary conditions for using that filtered NFPs values were accepted by users for the derived product. These conditions are specified on the Metadata and Validity Conditions of each filtered attribute value, and the user must analyse each of them.

5.3.3.2 Case 2: the value is not directly applicable

If Case 2, when the retrieved values are not applicable, but the user would like to have those qualities on the derived product, the user has the option to return to the **Feature Conf.** activity and change the set of features in order to obtain that desired good values of NFPs. Each change on the arrangement of features will result in different NFPs values. The measurement process of a given NFP related to a specific feature may depend on other features that are directly associated with the feature that is being analyzed. Thus, if the user add or delete any feature of the product, it is not possible to guarantee the same NFP value of a previous combination of features, before the changes.

For *not directly applicable* we mean that the filter found some good values, but they are dependent of Validity Conditions or Metadata restrictions, such as feature interactions necessary to at least maintain the value. And, in this case, as already explained, the user can go back to the activity of choosing features (**Feature Conf.**) and change the configuration for that product, creating a new one that preserves the restrictions of the target instance (value). This is a cyclic process and is represented by the green arrows of Figure 5.2.

5.3.3.3 Case 3: the value is not at all applicable or no value was filtered

If Case 3, when the filtered values are not at all applicable to the product, the user has to deal with situations where it is not possible to comply with the conditions requested by the Metadata and Validity Conditions of the filtered attribute values. Functional requirements can, for example, no longer meet to that ones required by the stakeholders/clients. In this case, the values are actually not appropriated to the product, i.e., it is not possible to reuse these values filtered from the A-base. The user has two options:

- Option 1 is to change the filter. The user can search either for Metadata or Validity Conditions (as showed on Figure 5.4), or directly specify on the filter the NFPs that he is looking for. Since the process is cyclic (dashed green arrows on Figure 5.2), the user can change the filter as much as necessary.
- Option 2 is to feed the A-base with new NFPs values. If the A-base does not have the NFPs values searched by the user, he needs to measure the NFPs for that product, and after that, to put these new information (in the way of Attribute Instances) inside the A-base, making them available to others. The next subsection, Step 4, discusses in details about this creation of new instances.

5.3.4 Step 4: Creation of new Attribute Instances

The main goal of the **NFPs Reuse Approach** is to avoid unnecessary analysis of NFPs values while an application engineer is deriving a new product, making the derivation process faster. If it maintains a repository (A-base) with NFPs values that have already been analysed (estimated, measured, simulated, etc.) for other products, it is possible that if the user wants to generate a new product similar to one already derived before, adequate NFPs values can be found for the new product, avoiding to calculate them again.

However, the reuse of NFPs values (or attribute instances) cannot always be done. For example, when: (i) the SPL is really new, with a few or no product derived; or (ii) even with an SPL that has many products, no value may be appropriated for the new product that is being generated. In these cases, no NFP value can be reused for the new product. But, if the users want to derive the new product aware of what are the NFPs values intrinsic to it, they need to do this analysis for the first time, and consequently make it available in the A-base. As explained on Section 5.3.1, Step 1, at any time a new insertion or modification of Attribute Instances can be made on the A-base.

The reuse approach activity of Figure 5.2 named **NFPs Analysis** represents this process of analysis of NFPs values for a target product. This activity is only visited if the user did not find any value in the A-base, and now he has to measure/estimate the NFPs of interest for the product.

There are several ways of measuring or estimating a NFP value. Chapter 2, Sections 2.3.1 and 2.3.2 presented types of NFPs and ways they are measured, through quantitative or qualitative measures. In addiction, we carried out a SLR, described on Chapter 3, with 36 papers that handle with NFPs in SPL. A couple of them showed how NFPs are, for example, papers [S18][S19][S20][S21] (Appendix A.1) presented various facets of an approach called SPL Conqueror. They showed how NFPs can be qualitatively and quantitatively measured in the context of SPLs.

The work in [S19], for instance, presented the analysis of some NFPs, such as reliability and performance. According to them, the former, reliability, is a quantitative NFP that can be analysed by selecting the features and implementation units that improve or degrade the NFP. To do so, experts can rank the features according to their influence on the NFP by defining a value for each feature. These values estimate the impact of a feature on the property, positively or negatively. For example, a system can have two features that improve reliability, F1 and F2, where F1 has a positive impact of +10 and F2 has +5. In this case, it means that F1 has a higher influence on reliability than F2. In order to analyse the latter, they consider performance as a product NFP, i.e., they need to execute the product and measure the NFP for the entire product, not for individual features. This measurement process is performed by running a benchmark for each product in terms of transactions per second, through for example, Oracle's standard benchmark.

However, it is noteworthy that is out of our work to discuss or provide methods of NFPs measurements. It is up to the people involved in the derivation process to decide the best way of measuring or estimating a property.

After analysing the desired NFPs, and in order to maintain the approach functional, the user of the NFPs Reuse Approach must feed the **A-base** with the newer values. This activity, we named **NFPs Feedback**. To do so, some points should be highlighted according to the NFPs Framework from Chapter 4:

- Attribute Registry (Task 2 Section 4.3.2). It is necessary to verify if the NFP that the user intends to measure has already been specified in the Attribute Registry. This registry is a repository of Attribute Types (a type is an specification of an NFP inside the framework). Case the user did not find the Attribute Type that he intends to measure, he has to create a

new one.

- Attribute Instance (Task 3 Section 4.3.3). An instance of an attribute represents the concrete value of an Attribute Type, i.e., one instance is associated to one type, that means one value is from one NFP. Besides the concrete value, an Attribute Instance also has the TypeID which refers to the Attribute Type, the Metadata and Validity Conditions. Examples of Attribute Instances are presented on Table 4.2.

- Metadata Registry (Task 4 Section 4.3.4). This registry is a repository of Metadata Types. During the measurement process of a given NFP value (Attribute Instance data), if the metadata present on the Metadata Registry are not enough to describe the conditions under which the value is being analysed, a new category of information can be added to the registry, i.e., a new Metadata Type.

The **A-base** is a repository of Attribute Instances, and whenever a new instance is created it must be added to the base. In this way, this new instance will be available to other users during the derivation of other products. This process is of great importance to ensure the efficiency and usefulness of the reuse approach. Figure 5.5 shows the work flow of Step 4, which corresponds to the creation of a new Attribute Instance. This flow must be followed to each NFP value that the user wants to measure for the target product.

According to Figure 5.5, firstly the user must search for the attribute in the Attribute Registry, such a way to discover if the Attribute Type he wants to measure exists or not. If the attribute exists, case “Yes”, it has to be measured with the measurement techniques chosen by the user. But, If the attribute has not been defined in the registry yet, case “NO”, the user can create and add it (following the specification of the NFPs Framework) in the Attribute Registry. After that, it can be measured.

Measuring an attribute means creating an instance for the Attribute Type with: `TypeID`, which is a reference for the Attribute Type that the instance is from of; `Data`, which contains a real value measured for the Attribute Type; `Metadata`, with all the metadata for this Data; and `Validity Conditions` with the conditions for the validity of the Data, and consequently, the validity of the instance. The Metadata must follow the Metadata Types defined in the Metadata Registry. If desired the user can, for example, add new Metadata Types. In fact, when creating a new instance and measuring a value, it is also necessary to add the context in which the value was measured, this is done by instantiating Metadata Types from its registry, the Metadata Registry.

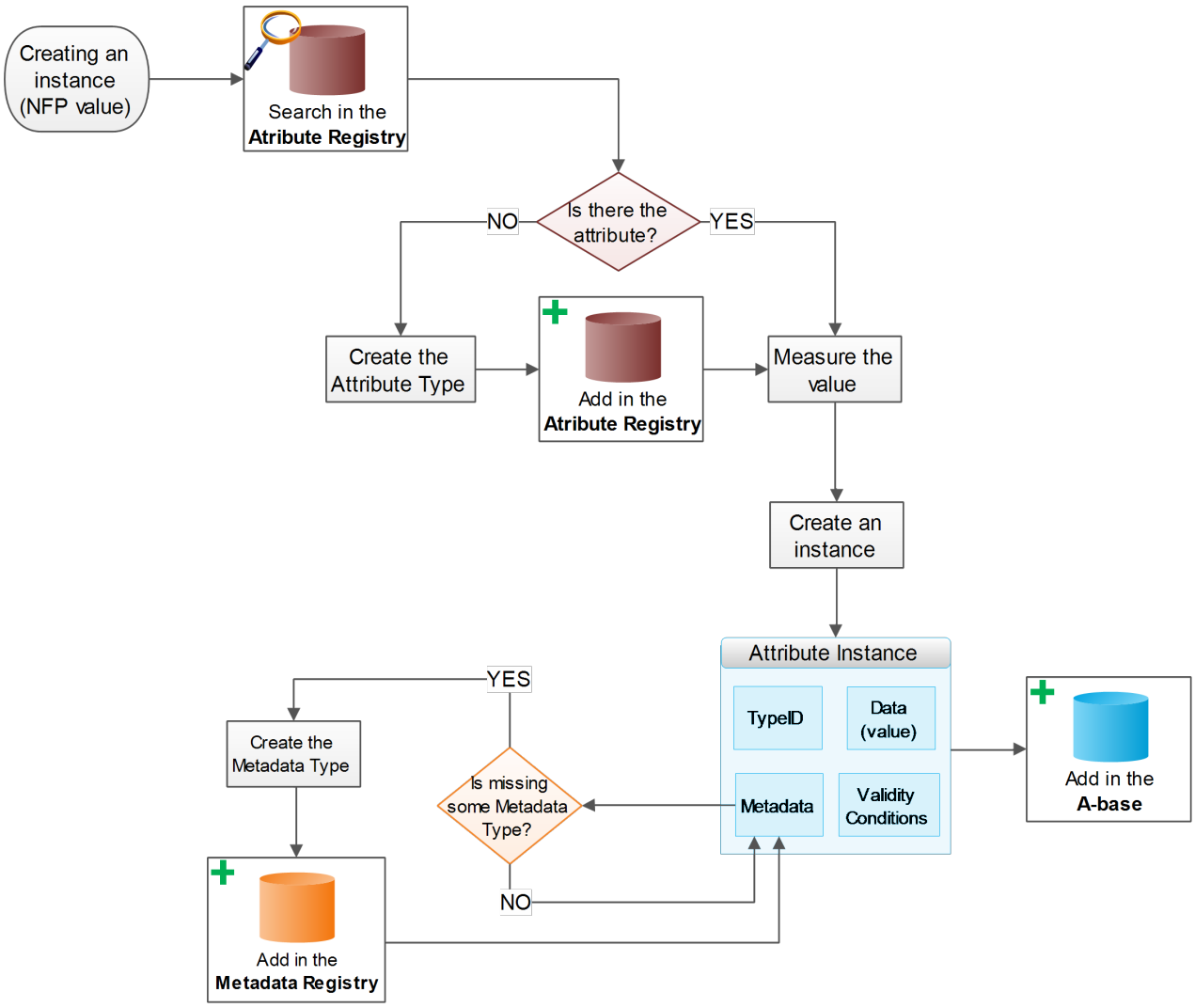

Figure 5.5: Work flow of Step 4

5.4 Chapter Summary

This Chapter focused on our **NFPs Reuse Approach** for SPL, showed in Figure 5.2. The approach intends to introduce the necessary steps for conducting a derivation of products aware of NFPs, such a way new analysis of NFPS can be reused, minimizing the effort of performing them again for each new product. This approach assumes that, in an SPL, similar products are derived and sometimes, the NFPs values that are measured for other products can be adequate or adaptable to new products. Doing so, we avoid calculating these values unnecessarily.

In order to better explain our approach, we presented it in four steps: Step 1, Step 2, Step 3 and Step 4. These steps correspond to the execution of new activities, specific for NFPs reuse,

proposed on the approach. The first one is a preparation step and explains how to populate the A-base with the Attribute instances (values), besides details that the Attribute Registry and Metadata Registry have to be fulfilled first. Attribute Instances are instances of Attribute Types present on the Attribute Registry; and Attribute Instances have Metadata Instances that are instances of Metadata Types present on the Metadata Registry. This Step answers the Question 1 formulated on Section 5.1, since it shows that it is possible to populate the A-base during the domain engineering, when no product have been derived yet.

Step 2 comes after populating the registries and A-base, and explains how to define a filter that searches for NFPs values for a target product. The filter contains matching conditions (from the NFPs Framework, Section 4.3.5), where the users define the conditions to leverage only the NFPs of interest. This Step answers Question 2, such a way it discusses about the process of filter creation, which aims at obtaining only the values of interest to the derived product.

Step 3 explained what happens after executing the NFPs Filter. This process is shared in three different cases, when the filtered instances (values) (i) are useful; (ii) are not directly applicable; and (iii) are not at all applicable or no value was filtered. In order to decide what is the appropriate case for the filtered instances, users have to analyse the Metadata and Validity Conditions of each instance. They have the details of the context in which the instance is valid. This Step explains Question 3 by presenting that an Attribute Instance has all the information for its applicability.

The last step, Step 4, corresponds to the answer for Question 4. It explains that third case of Step 3, when there are not appropriated values that the user can reuse for the derived product. For this case, the user has to measure the NFPs for the new product and make the new values available on A-base. This process corresponds to the creation of new Attribute Instances, and represents the worst case of the NFPs Reuse Approach, which is more likely to happen when SPL has only few derived products.

In order to show a real use of the NFPs Reuse Approach, next Chapter 6 presents an exploratory case study for a text editor SPL in the desktop context.

6

The Case Study

Through the SPL activity of features selection, performed by the product derivation process, both functional and non-functional properties should work fine to produce SPL products. Since functional properties are reused among products of an SPL, NFPs might be reused too. By the time a new product is being derived, stakeholders can have access to a repository of NFPs already analyzed (for other products) and can decide for reusing some property value. By doing so, they minimize the effort of performing a new analysis. If previously they had to analyse/measure required NFPs for each new derived product, now they have the option to avoid this work by reusing NFPs of other products. For this process of reusing NFPs, we defined an NFPs Reuse Approach, as discussed in the previous Chapter.

In the software engineering context, empirical studies such as case studies, surveys, and experiments, play an important role. The progress in any discipline depends on the ability of people to understand the basic units necessary to solve problems (Basili *et al.*, 1986). In particular, case studies provide a deep understanding of the phenomena under study, and offer an approach that does not require a strict boundary between the object of study and its environment (Runeson *et al.*, 2012).

This Chapter presents an exploratory case study research. Runeson *et al.* (2012) used three of the main widespread general definitions of the term `case study` (Benbasat *et al.*, 1987; Yin, 2009; Robson, 2002), in order to derive a specific definition for software engineering. Thus, according to Runeson *et al.* (2012), a case study is an empirical enquiry based on multiple sources of evidence to investigate one or a small number of instances of a contemporary software engineering phenomenon within its real-life context.

The case study presented on this Chapter aims to investigate the NFPs Reuse Approach applicability in a product line of text editors. The remainder of this Chapter is organized as follows. Section 6.1 presents the research design with the most important definitions around the

case study research. The results and findings are described on Section 6.2. Section 6.3 discusses the threats to validity; and finally, Section 6.4 presents the chapter summary.

6.1 Case Study Protocol

The research design for a specific exploratory case study is documented as a case study protocol. Once a case study is a research project with resource constraints and time limits, a protocol with a detailed planning is essential. The following elements are part of this protocol (Runeson *et al.*, 2012): rationale and objective of the study (6.1.1), the bounded system or case (6.1.2), units of analysis (6.1.3), case study research questions (6.1.4), data collection instruments (6.1.5) and data analysis procedures (6.1.6).

6.1.1 Rationale and Objective

Rationale for the study aims at explaining the reasons for undertaking the study, and the objective of the study describes what is expected to be achieved with the study (Runeson *et al.*, 2012).

The case study performed in this work has a exploratory purpose. Exploratory characteristic aims at searching for new insights, finding out what is happening, besides generating ideas and hypothesis for new research (Runeson *et al.*, 2012).

We undertook this case study to make a novel contribution to the body of knowledge on the SPL product derivation process, taking into account the reuse of NFPs values.

The objective is to investigate the application of the NFPs Reuse Approach (and consequently, the NFPs Framework) inside an SPL context. This is the overall goal of our study, but according to the Goal-Question-Metric (GQM) template (Basili *et al.*, 1994; Wohlin *et al.*, 2000) other details should also be set out, such as: object of study, purpose, quality focus, perspective and context. Thus, following the GQM template our study aims to:

- Analyze the **NFPs Reuse Approach**,
- for the purpose of **investigating**,
- with respect to its **applicability**,
- from the view point of **domain and application engineers**,
- in the context of a **small SPL**.

The result of applying GQM approach is a specification of a model, which targets a particular set of issues and a set of rules for data interpretation (Wohlin *et al.*, 2000). The resulting model has three levels, conceptual (goal), operational (question) and quantitative (metric). The goal has already been defined, but next sections show more details about it (mainly concerning to view point and context). Also, the other two levels, question and metric, will be described on research questions section (6.1.3).

6.1.2 The Case

According to Runeson *et al.* (2012), the case in software engineering is anything that is a contemporary software engineering phenomenon in its real-life context. For "contemporary", they state as a necessity to allow data collection from the case. Further in accordance with them, software projects are an obvious candidate as an object of study in a case study.

The product-line under study is an academic text editor SPL implemented by the RiSE Labs group[1]. The development team was comprised of two M.Sc. students. This SPL was inspired in seven different text editors programs independently developed in a course on feature-oriented design, at the University of Texas at Austin. Another study (Apel and Beyer, 2011) already used the same programs in order to explore feature cohesion characteristics. The text editors can be downloaded at their project website[2].

The text editors SPL implemented by the RiSE Labs group, called Notepad SPL, is currently in its second version, which contains more features than the initial project and improvements in usability. Figure 6.1 shows a screenshot of the most complete product of the Notepad product-line developed by RiSE Labs group.

The Notepad SPL is an desktop application implemented in JAVA language, using a Model-View-Controller (MVC) architectural pattern. The SPL has 1,803 lines of code and 14 java class spread over the 3 packages of MVC pattern. Figure 6.2 presents this structure of packages and class.

In SPL engineering, products can be generated from a common platform by specifying a selection of features. This includes implementation techniques, such as conditional compilation (CC), to allow the variability among products. The CC technique enables to include or exclude code fragments from a program compilation. It is responsible for managing SPL variability by marking code regarding the varying features. Directives mark the varying lines of code stating which parts of the code are associated to given features. By selecting features for a product,

[1] http://labs.rise.com.br
[2] http://www.fosd.de/FeatureVisu/

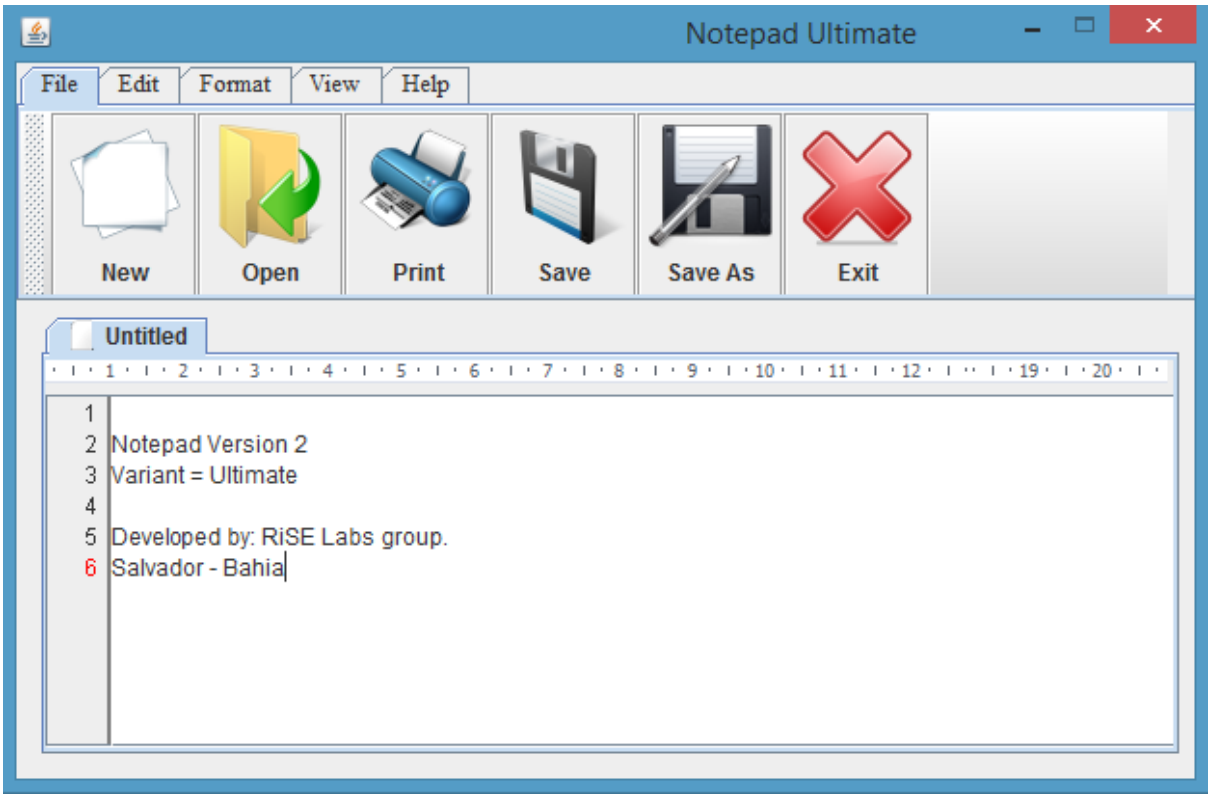

Figure 6.1: Screenshot a Notepad SPL product named Notepad Ultimate.

associated code are enable to generate the product in question (Couto *et al.*, 2011).

For the Notepad SPL, CIDE tool (Kästner, 2010) is used to support and manage the CC technique, providing the SPL assets reuse. The CIDE tool is an Eclipse IDE plug-in that marks the lines of code by annotating them with background colors. Each feature receives a different color, and developers paint the lines that refer to a given feature. Figure 6.3 shows an excerpt of the Java class "EditModel" from Eclipse IDE, where the code are annotated (painted) for two different features, one in pink and the other in brown.

Notepad SPL contains forty features and three main products: Notepad Lite, Notepad Standard and Notepad Ultimate, each one with 10, 31 and 35 features respectively. Notepad Lite contains only the basic features to allow an edition of texts; Notepad Standard has more functionalities than the former, but also does not have all ones; And Notepad Ultimate is the most complete product. Figure 6.4 shows this relation among the products. In addition, Appendix B.1 presents the feature model of the complete SPL. A Feature model represents features of the product line and the variation of them among products, through optional and alternative features. Appendix B.2 shows the product map, which lists the features present in each product of the SPL.

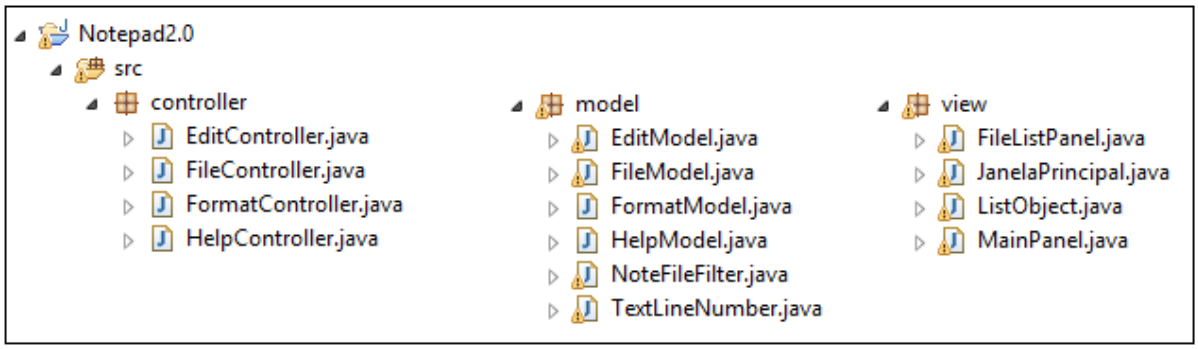

Figure 6.2: Packages and class of the Notepad SPL

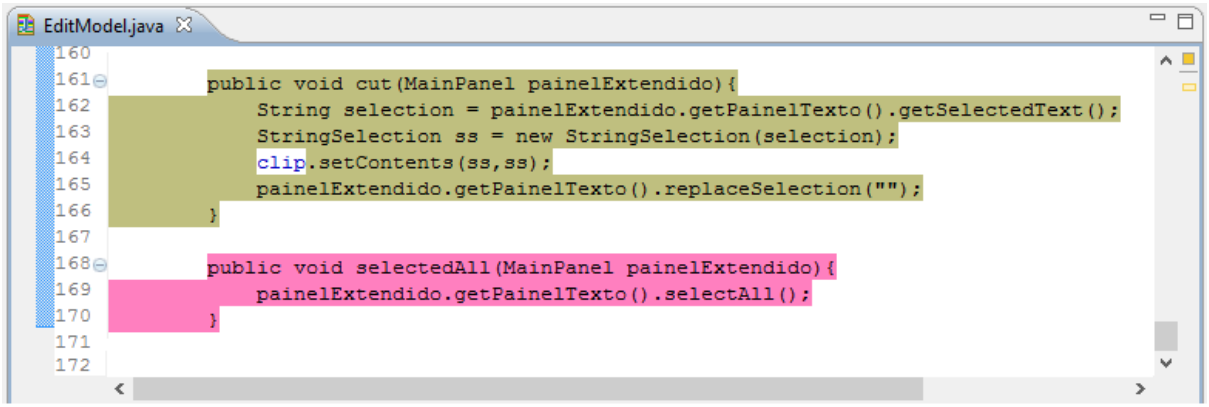

Figure 6.3: CIDE annotations in a Notepad SPL class

6.1.3 Units of Analysis

The NFPs Reuse Approach is the unit of analysis for the case study. However, this approach in analysis adds a couple of steps into the product derivation process. For example, A-base and Configuration Filter are two new artifacts into this process, which we intend to evaluate. Furthermore, the NFPs Framework is a third artifact intrinsic to those, since Attribute Instances contained in the base and filtered by the filter are defined by the framework.

6.1.4 Case Study Research Questions

According to Runeson *et al.* (2012), a literature review can identify areas for new contributions to knowledge as well as provide (part of) the justification for the case study. Issues identified in the primary studies may identify interesting gaps in the literature review and these gaps can demonstrate the importance of conducting the case study.

Still in accordance with them, research questions needs to be precise and unambiguous.

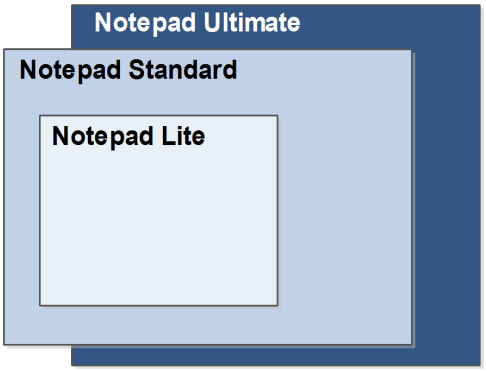

Figure 6.4: Relation among products in Notepad SPL

Because of intrinsic characteristics of an exploratory case study, where there is an absence of established theory, the researcher may simply ask questions such as: "What is going on here?" and "How has a particular outcome come about?" (Runeson *et al.*, 2012). In these situations, propositions could be outputs from the study rather than elements of the design of the study.

The research questions of our exploratory had their origins in the systematic literature review of Chapter 3. From this review and the identified gaps, came the NFPs Framework conception and, consequently, NFPs Reuse Approach. These questions evaluate the case under the domain and application engineers perspective. In this case study, engineers are responsible for applying the NFPS Reuse Approach concepts into life-cycle of the Notepad SPL.

As previously mentioned, the main objective of this study is to investigate the application of the NFPs Reuse Approach inside an SPL context. In order to address this objective, small goals based on it can also be defined. These small goals are:

- to apply the reuse approach to the SPL product derivation process of the Notepad SPL;
- to analyse the effort to specify NFPs values according the NFPs Framework guidelines.
- to assess the results in order to provide the lessons learned and improvements on the reuse process;

Thus, in order to characterize these issues, we defined the following research questions with its rationale.

Q1: What is necessary to learn to start using the approach?
Rationale: this question investigates the learning process for applying the NFPs

Reuse Approach. It aims to analyse the additional tasks a domain engineer and application engineer need to understand compared to a conventional product derivation process. We consider as a conventional process, the SPL product derivation process as described in Chapter 2.2, without this specific approach for NFPs reuse.

Q2: Does the NFPs Reuse Approach avoid unnecessary (re)analysis of NFPs values?
Rationale: the goal is to verify the reusability of NFPs values, and clarify if new measurements of NFPs values can be avoided.

Q3: What is the effort spent to fill the A-Base with new instances of attributes?
Rationale: this question evaluate the creation of new NFPs values (instances) and their addition in the A-base when no value was reused and the engineer has to measure new ones.

Q4: What are the drawbacks and benefits of the NFPs Reuse Approach?
Rationale: the goal is to understand if and how the approach can be improved, besides knowing the approach advantages.

Each research question has a common metric: subjective evaluation by the engineer; which is an usual qualitative GQM metric, as presented by (Basili *et al.*, 1994). Only Q3 has a different metric related to the effort characteristic. Table 6.1 illustrates the GQM template filled in with the specifications of our case study.

6.1.5 Data Collection

Runeson *et al.* (2012) discuss data collection techniques from three degrees: (i) direct methods; (ii) indirect methods; and (iii) independent analysis. For the first degree, researchers are in direct contact with the interviewees and collect data in real time. Examples are interviews and focus groups. In the second degree, researchers collects raw data without actually interacting with the interviewees; and the last one works with methods where researchers independently analyse only available artifacts.

For this exploratory case study, direct methods were used, since the researcher has a face-to-face contact with the SPL engineers and a free contact with the Notepad SPL. This is possible thanks the academic characteristic of the product line and physical proximity between researchers and engineers involved. This study adopted three data collection methods, namely documentation analysis, observation, and interviews.

Table 6.1: GQM template

Goal	Object of study	NFPs Reuse Approach.
	Purpose	Investigate.
	Focus	Applicability.
	Point of View	Domain and application engineers
	Context	Small SPL
Question	Q1	What is necessary to learn to start using the approach?
Metrics	M1	Subjective evaluation by the engineers
Question	Q2	Does the NFPs Reuse Approach avoid unnecessary (re)analysis of NFPs values?
Metrics	M1	Subjective evaluation by the engineers
Question	Q3	What is the effort spent to fill the A-Base with new instances of attributes?
Metrics	M1	Subjective evaluation by the engineers
	M2	Time spent by the engineer on specific NFPs Reuse Approach tasks, such as to create and maintain Attribute Types and Attribute Instances.
Question	Q4	What are the drawbacks and benefits of the NFPs Reuse Approach?
Metrics	M1	Subjective evaluation by the engineers

Documentation Analysis. This technique focuses on the documentation generated by software engineers (Shull *et al.*, 2008). Documents related to SPL output artifacts from product derivation process was analyzed, which include feature model and product map. Artifacts from the NFPs Reuse Approach were important to be analysed too, such as Attribute Types and Attribute Instances documentation. These artifacts are essential to understand and derive products of the SPL.

Observation. This method usually has the benefit of providing a deep understanding of the phenomenon that is being studied. The observation method follows the advices from (Runeson *et al.*, 2012) through the use of the "think aloud" protocol, where the researcher repeatedly asks questions to remind the subjects to think aloud. The observed roles were that ones already mentioned, domain and application engineers. The product derivation process using the reuse ideas from the NFPs Reuse Approach was observed during over 1 hour.

Interviews. In case studies of software engineering area, interviews are one of the most frequently used data sources (Runeson *et al.*, 2012). The type of interview adopted in this case study was an unstructured interview. In this type, questions are formulated in an open way according to general concerns and interests from researchers. Our interview questions were formulated based on the defined research questions (Q1, Q2, Q3 and Q4). Thus, Face-to-face interviews were conducted after the SPL engineers had applied the reuse approach. The

participants were free to talk about the process deficiencies and benefits of the approach. Some notes were taken during the interview, but the main documentation way was the sound recording, which was transcript as part of the analysis. The asked questions in the interview of this case study are presented in Appendix B.3.

6.1.6 Data Analysis

The data analysis was performed through a direct qualitative analysis aiming to answer our specific research questions. In order to observe the application of the reuse approach inside the Notepad SPL (desktop) context, the SPL engineers were submitted to three activities: initial phase, observation and interview.

Activity 1 - Initial Phase. The researcher performed a brief explanation to the SPL engineers about the NFPs Framework and NFPs Reuse Approach, which was responsible to present their motivations and details. These details concern about the framework key tasks (Chapter 4.3) and the NFPs Reuse Approach steps (Chapter 5.3).

Activity 2 - Observation. This activity consisted of observing the SPL engineers during the product derivation process aware of NFPs, i.e., using the NFPs Reuse Approach to derive the SPL products. The engineers followed a workflow, with five points, such a way it was possible to analyse how they conduct the reuse approach steps. Thus, they were asked to:

1. choose a couple of NFPs for the Notepad product, and to specify them in the way of Attribute Types. Documentation used: Attribute Types Registry, Attribute Metadata Registry;

2. measure those NFPs for the three available SPL products, Notepad Lite, Notepad Standard and Notepad Ultimate;

3. specify and to document the measured NFPs values in the way of Attribute instances, according to the NFPs Framework. Documentation used: A-base (repository with the instances);

4. derive a new product with a specific NFP need. Thus, according to the NFPs Reuse Approach, they derived a product and created a Configuration filter with the specific need chosen, as presented on Figure 5.2. Documentation used: Feature Model and Product Map.

5. analyse the instances retrieved from A-base according to the filter that has been drawn. This process of analysis is referred, by the reuse approach, as “Reuse Diagnosis activity” and it is composed by three cases, as presented in Chapter 5.3.3.

Activity 3 - Interview. After the SPL engineers had completed the aforementioned steps from Activity 2, we applied an interview following the form of Appendix B.3. The audio file has twenty minutes, and it was transcript to facilitate the analysis process.

6.2 Results and Findings

In this section, the findings of the case study are presented describing the application of the NFPs Reuse Approach for the Notepad SPL desktop context, through the answers of the case study research questions. Some general data from the case study are depicted in Table 6.2.

Table 6.2: Case Study General Data

Activity	**Addressed to**	**Time**
Initial activity: brief explanation on - NFPs Framework - NFPs Reuse Approach	Application Engineer	30 min.
Observation 1: NFPs Reuse Approach domain phase - Populating A-base	Domain Engineer	25 min.
Observation 2: NFPs Reuse Approach application phase - Product Derivation	Application Engineer	60 min.

The Notepad SPL was developed by two SPL engineers from RiSE Labs group. They are M.Sc. students with three years of experience in SPLE. For this case study, one of them played the role of a domain engineer and the other was the application engineer. Table 6.2 showed that the brief explanation activity was only submitted to the application engineer, since in this case, the domain engineer was the author of this work. This activity lasted over 30 minutes and some details are presented on Appendix B.3.

The observation activity was shared in two parts, one for the domain engineer and the other for the application engineer.

6.2.1 Observation activity part 1

This first part includes only the three first points of the observation workflow (shared in five points) presented on previous section. As the Notepad SPL has already been implemented, with three different products and there were no non-functional requirement documented, the domain engineer could start performing the Step 1 of the reuse approach: *Populating the A-base*. As described on Chapter 5.3, the reuse approach is composed by four steps, but only the first one is part of the domain engineering phase, although it can be accessed at any time of the SPL life-cycle.

For this step, the domain engineer was responsible to leverage the non-functional attributes for the SPL, creating the Attribute Type Registry and Metadata Type Registry (workflow point 1). Figure 6.5 and Figure 6.6 show these registries.

Attribute Type Registry			
TypeID:	Footprint_Product	**TypeID:**	Performance
Attributables:	Product	**Attributables:**	Feature
Variability:	Optional	**Variability:**	Optional
DataFormat:	{Quatitative, KiloBytes (KB)}	**DataFormat:**	{Qualitative, Low/Medium/High}
Description:	Binary size of compiled files (bin files).	**Description:**	Qualitative performance of a product feature specified by experts.
Documentation:	no	**Documentation:**	no
TypeID:	Memory_Consumption	**TypeID:**	Usability
Attributables:	Product	**Attributables:**	Feature
Variability:	Optional	**Variability:**	Optional
DataFormat:	{Quantitative, MegaBytes (MB)}	**DataFormat:**	{Qualitative, Low/Medium/High}
Description:	Consumed main memory during the program execution.	**Description:**	Ease of use of products or features specified by experts.
Documentation:	no	**Documentation:**	no

Figure 6.5: Attribute Type Registry for Notepad SPL

The domain engineer specified four different Attribute Types, namely, `Footprint_Product`, `Memory_Consumption`, `Performance`, and `Usability`. The first two can be measured for entire products, and the last two are assigned to individual features. All are optionals, i.e., the engineers/clients decide if this NFP is necessary for a given product or not. The metadata registry was based on the example given on Table 4.3.

After specifying some types of attributes, the domain engineer analysed two attributes. At this stage, it was analysed only NFPs *attributable to* features, not for products, that according to

Metadata Type Registry	
MetaID: Creation_Time **ValueFormat:** Time_Stamp **Variability:** Mandatory **Description:** Creation data and time of Attribute Values.	**MetaID:** Source **ValueFormat:** Estimation/Measurement/Simulation **Variability:** Mandatory **Description:** The way a value is analysed.
MetaID: Modification_Time **ValueFormat:** Time_Stamp **Variability:** Optional **Description:** Modification data and time of Attribute Values.	**MetaID:** Source_Tool **ValueFormat:** String **Variability:** Optional **Description:** If any, tool responsible for analysing the value.
	MetaID: Complexity **ValueFormat:** Low/Medium/High **Variability:** Mandatory **Description:** Complexity to analyse a value
MetaID: Version **ValueFormat:** Integer **Variability:** Mandatory **Description:** Version of Attribute Values.	**MetaID:** Comment **ValueFormat:** String **Variability:** Optional **Description:** Any other additional comment.

Figure 6.6: Attribute Metadata Registry for Notepad SPL

the actual registries are: `Performance` and `Usability` (workflow point 2):

- Feature: Linear_Search -> performance: Low.
- Feature: Binary_Search -> performance: Medium.
- Feature: Single_OpenedFiles -> usability: Medium.
- Feature: Multiple_OpenedFiles -> usability: High.

Thus, two Attribute Instances were created for `Performance` and other two for `Usability` (workflow point 3), shown on Figure 6.7. All these specifications, registries and instances, were documented on electronic spreadsheets. The time needed to perform all these tasks was in average 25 minutes. It is worth to mention that the domain engineer has a deep understanding on Reuse Approach activities and the Notepad SPL too, since she was one of the SPL developers.

6.2.2 Observation activity part 2

For the second part of the observation (Table 6.2), the application engineer has particular differences compared to the other engineer. Firstly, he did not know the guidelines of the NFPs Reuse Approach in a deep way like the domain engineer, besides he does not work on domain engineering phase, but on application engineering, which is responsible for the product derivation process.

This second observation part includes the five points of the observation workflow presented on previous section (6.1.6), which lasted over 60 minutes. Hereafter, it will be presented the results of each workflow point.

1. He chose two NFPs, but they had already been specified as Attribute Types at the registry (Figure 6.5) by the domain engineer: `Footprint_Product`, `Memory_Consumption`.

2. He measured the NFPs for the three available SPL products. Appendix B.2 shows the product map of these products. The measured values were:

   - Product: Notepad Lite -> footprint: 116 KB, memory: 23.1 MB.
   - Product: Notepad Standard -> footprint: 232 KB, memory: 27.3 MB.
   - Product: Notepad Ultimate -> footprint: 260 KB, memory: 29 MB.

3. He specified and documented the measured NFPs values in the way of Attribute instances, according to the NFPs Framework, as shown on Figure 6.8 and Figure 6.9. Thus, from here the A-base can be seen as a set composed by the instances from Figure 6.7, Figure 6.8 and Figure 6.9.

4. According to the NFPs Reuse Approach, the analysis of the filtered instances are performed by the activity named Reuse Diagnosis, which is composed by three cases: (i) the derived product does not need new measurements, because the filtered values were useful; (ii) the filtered values were not applicable, and the engineer changes the product to make it fit; (iii) the filtered values were not useful, and the engineer needs to measure their required NFPs to get its values.
   Three new different products were derived, one for each case. Appendix B.4 shows the product map of these products. For each product a specific Configuration Filter was also specified:

   - Product: Notepad Lite+2
     Configuration Filter: (Source = Measurement) AND (TypeID = Footprint_Product)
   - Product: Notepad Standard+1
     Configuration Filter: (Footprint_Product = Lower) ELSE (Memory_Consumption = Lower) ELSE (Performance = Best) ELSE (Usability = Best)
   - Product: Notepad Ultimate-1
     Configuration Filter: (TypeID = Execution_Time)

5. At this point, the engineer evaluated the results of each filter one at a time:

   (a) The first Configuration Filter is able to filter from the A-base only the instances from the Figure 6.8. A-base is here represented by the Figure 6.7, Figure 6.8 and Figure 6.9. This analysis fits to Reuse Diagnosis case 1, since the engineer classified the filtered values as useful, where no other measurement needs to be computed. The application engineer is an expert on the Notepad SPL, thus, from the filtered information, he could realise that the new product (Notepad Lite+2) will have a value of footprint smaller than the Notepad Lite and greater than the Notepad Standard, so that:

   $$Footprint\ (Notepad\ Lite) \leq Footprint(NotepadLite+2) \leq Footprint(NotepadStandard)$$

   (b) The second Configuration filter aimed to filter only the best values of each NFP, where the application engineer realised that the derived product could be modified in order to attend one good interesting value: to include the feature Multiple_OpenedFiles to obtain a better level of usability. According to the Reuse Diagnosis activity, this analysis fits to case 2. The list of the filtered values were:
   -Lower Footprint = Product Notepad Lite;
   -Lower Memory = Product Notepad Lite;
   -Best Performance = Feature Binary_Search;
   -Best Usability = Feature Multiple_OpenedFiles.

   (c) The third Configuration Filter was not able to filter any value, since the attribute required does not exist in the Attribute Registry, and thus, there are no instances of it yet. This analysis fits to the case 3 of the Reuse Diagnosis activity. In this way, the application engineer had to measure this attribute for the product and also included it in the documentation: a new Attribute Type and Attribute Instance were created.

A-base (Attribute Instances)

TypeID: Performance

1	ProductName/FeatureName:	Feature: Linear_Search	
	Data:	Low	
	Metadata:	Creation_Time:	28.07.2014-22:01
		Version:	1.0
		Source:	Estimation
		Complexity:	Low
		Comment:	Experts defined the Data based on the feature implementation
	ValidityConditions:	---	
2	ProductName/FeatureName:	Feature: Binary_Search	
	Data:	Medium	
	Metadata:	Creation_Time:	28.07.2014-22:03
		Version:	1.0
		Source:	Estimation
		Complexity:	Low
		Comment:	Experts defined the Data based on the feature implementation
	ValidityConditions:	---	

TypeID: Usability

1	ProductName/FeatureName:	Feature: Single_OpenedFiles	
	Data:	Medium	
	Metadata:	Creation_Time:	29.07.2014-17:19
		Version:	1.0
		Source:	Estimation
		Complexity:	Low
		Comment:	- Files are opened one at a time during program execution.\ Experts defined the Data based on the feature
	ValidityConditions:	---	
2	ProductName/FeatureName:	Feature: Multiple_OpenedFiles	
	Data:	High	
	Metadata:	Creation_Time:	29.07.2014-17:20
		Version:	1.0
		Source:	Estimation
		Complexity:	Low
		Comment:	- Multiples Files can be opened during program execution.\ Experts defined the Data based on the feature
	ValidityConditions:	---	

Figure 6.7: A-base (Part 1 of 3) with performance and usability.

A-base (Attribute Instances)			
TypeID: Footprint_Product			
1	ProductName/FeatureName:	Product: Lite	
	Data:	116KB	
	Metadata:	Creation_Time:	28.07.2014-20:48
		Version:	1.0
		Source:	Measurement
		Complexity:	Low
	ValidityConditions:	FeatureInteractions:	6F: Open, New, Save_As, Line_Wrap, Exit, SingleOpenedFiles
2	ProductName/FeatureName:	Product: Standard	
	Data:	232KB	
	Metadata:	Creation_Time:	28.07.2014-20:52
		Version:	1.0
		Source:	Measurement
		Complexity:	Low
	ValidityConditions:	FeatureInteractions:	23F: File_Open, File_New, File_Save_As, File_SaveNormal, File_Print, Edit_Find, Edit_Find_Next, Edit_Paste, Edit_Cut, Edit_Copy, Edit_SelectAll, View_File_Property, View_Highlight_Syntax, View_Line_Number, View_RuleBar, Line_Wrap, Font_Type, Font_Style, Font_Size, Font_Preview, Exit, Standard_SearchAlgorithm, Single_OpenedFiles
3	ProductName/FeatureName:	Product: Ultimate	
	Data:	260KB	
	Metadata:	Creation_Time:	28.07.2014-21:08
		Version:	1.0
		Source:	Measurement
		Complexity:	Low
	ValidityConditions:	FeatureInteractions:	25F: File_Open, File_New, File_Save_As, File_SaveNormal, File_Print, Edit_Find, Edit_Find_Next, Edit_UnDo, Edit_ReDo, Edit_Paste, Edit_Cut, Edit_Copy, Edit_SelectAll, View_File_Property, View_Highlight_Syntax, View_Line_Number, View_RuleBar, Line_Wrap, Font_Type, Font_Style, Font_Size, Font_Preview, Exit, Help_About, Binary_SearchAlgorithm, Multiple_OpenedFiles

Figure 6.8: A-base (Part 2 of 3) with footprint.

A-base (Attribute Instances)			
TypeID: Memory_Consumption			
1	ProductName/FeatureName:	Product: Lite	
	Data:	23.1MB	
	Metadata:	Creation_Time:	28.07.2014-20:30
		Version:	1.0
		Source:	Measurement
		Source_Tool:	Windows Task Manager
		Complexity:	Low
	ValidityConditions:	Platform:	Operational_Sytem: Windows 8.1 Enterprise 64bits CPU: Intel Core i5-2410M 2.3GHz RAM: 6GB
2	ProductName/FeatureName:	Product: Standard	
	Data:	27.3MB	
	Metadata:	Creation_Time:	28.07.2014-21:34
		Version:	1.0
		Source:	Measurement
		Source_Tool:	Windows Task Manager
		Complexity:	Low
	ValidityConditions:	Platform:	Operational_Sytem: Windows 8.1 Enterprise 64bits CPU: Intel Core i5-2410M 2.3GHz RAM: 6GB
3	ProductName/FeatureName:	Product: Ultimate	
	Data:	29MB	
	Metadata:	Creation_Time:	28.07.2014-21:35
		Version:	1.0
		Source:	Measurement
		Complexity:	Low
	ValidityConditions:	Platform:	Operational_Sytem: Windows 8.1 Enterprise 64bits CPU: Intel Core i5-2410M 2.3GHz

Figure 6.9: A-base (Part 3 of 3) with memory.

6.2.3 Research Questions

After the observation activities, parts 1 and 2, it was performed a 20-minutes interview, in order to contribute with the evaluation of the approach and answer the research questions. The interview was focused on the application engineer, who was responsible for performing the product derivation process. The transcription of the interview lasted over 38 minutes.

The documentation created during the observation activities, electronic spreadsheets, the perceptions of the researcher, and the document with the transcript interview served as the basis to answer the research questions.

6.2.3.1 Q1: What is necessary to learn to start using the approach?

The application engineer characterized the NFPs Framework as simple and of easy understanding. There was no major difficulties to use the approach. However, in spite of the definitions of each element from the framework were clear, the engineer pointed out the importance of providing a more direct guide, in the style of "How to do" steps. In this guide, it could be defined in a few words the required fields to be filled out to create, for the first time, an Attribute Type and an Attribute Instance. The activity required to start using the reuse approach is to get a good understanding of the NFPs Framework. For the engineer, the initial 30-minutes activity was enough to learn the goals and motivations of the proposed approach.

6.2.3.2 Q2: Does the NFPs Reuse Approach avoid unnecessary (re)analysis of NFPs values?

The reuse of NFPs analysis, as stated by the application engineer, can be avoided in cases where estimated values are sufficient. Because, if only exact values are required, an estimated value is not enough and the engineer will have to measure it. But sometimes, it is possible to analyse upper and lower bounds regarding to the product in analysis, where these values are so close that the estimation can be further exploited. This was the case of the first product derived during the observation activity part 2, where NFPs were filtered by footprint. However, this kind of analysis only can be made by experts on the SPL in study. It is also encouraged that they have a minimum of knowledge on quality attributes.

Another point is related to the fact that not every SPL needs a depth knowledge in quality attributes. Applications addressed to devices with limitations on, for example, memory and CPU, take more advantage of the NFPs Reuse Approach than systems addressed to desktop domain, where there is not such a strong limitation. In devices like tablets and smartphones,

individual applications cannot consume a huge amount of resources, otherwise they can slow down and not work rightly.

An interesting point raised during the interview was about other utility for the A-base (repository of attribute instances). It serves as a derivation guide, which means that the reuse approach is not only focused on NFPs reusability, but it also can be attractive as a way to know more about the SPL. Before or after deriving a new product, it may be worth finding out the strengths and weaknesses of features and products, by taking an overall look on the A-base. This is possible due to the standardization provided by the NFPs Framework related to the description of each instance, which include metadata with specific information of each instance facilitating a comparison among the them.

This situation happened during the derivation of the second product in the observation activity part 2, where only the best values of each Attribute Type were searched. From the information filtered, the engineer could revise and change the product in order to improve it in some aspect. The aspect chosen was about to insert a feature, “Multiple_OpenedFiles”, in the product such a way to improve the product usability.

6.2.3.3 Q3: What is the effort spent to fill the A-Base with new instances of attributes?

When it is necessary to measure new values of NFPs, the effort spent to document it in the way of Attribute Instances is, according to the engineer, irrelevant. The engineer took less than 3 minutes to create an instance. However, when it was necessary to create for the first time a new classification type of NFPs, i.e., a new Attribute Type, the engineer took over 8 minutes. According to him, it was imperative a review on the NFPs Framework concepts before creating the required type, which justifies the time spent.

The greatest effort is not to specify, but to measure NFPs. Thus, compared to the development process of an SPL and also the process of measuring a NFP, specifying NFPs in terms of Attribute Type or Attribute instances is not a big deal. Nevertheless, in the beginning of the SPL development, where there is no NFP documented, depending on the size of the product-line, many Attribute Types must have to be specified, which entails an effort a little longer.

For the case study presented here, the domain engineer was responsible for this initial task of specifying the first Attribute Types. She had a good knowledge on the framework and reuse approach. In other cases, when this situation does not occur, the maturing process to deploy the framework might need a longer time than that one of the “Initial phase activity”. This activity was responsible for the brief explanation of the artifacts and lasted only 30 minutes.

6.2.3.4 Q4: What are the drawbacks and benefits of the NFPs Reuse Approach?

As a positive point of the NFPs Framework, the engineer characterized it as intuitive. According to him, the fields necessary to create a new Attribute Type, such as TypeID, Attributables, Variability and Description, are simple. From the name, it is possible to understand what it means. The same happens to the fields of Attribute Metadata and Attribute Instance.

The framework task regarding the values selection, performed by the Configuration Filter, was a little more complicated to understand, especially the "matching conditions", which needed more attention. However, during the observation activity, despite of the three filters had been perfectly described, they were not used as they should. The Configuration filter serve to filter the values from a set of instances, named in the NFPs Reuse Approach of A-base. But, for this case study two situations did not allow to properly run the filter: (i) there was only a small set of values; and (ii) the values were described in common spreadsheets. The filter in the way it was defined only creates a pattern to search for values, but it does not have usability since there is not a system, like a repository or database to search for these values. It is missing a tool that implements the search by the filter.

The idea of the NFPs Reuse Approach is as simple as the framework. Through the approach is possible to guide a product derivation aware of NFPs. The advantage is not only to reuse NFPs values, but to know more about the SPL in terms of quality. Stakeholders can start the derivation process by looking into the A-base to search for values that could be similar to that ones of the product they intend to derive. The validity conditions and metadata of each value can be useful in this process.

In spite of the effort of specifying types and instances of attributes to be small, the application of the reuse approach for large scale SPL may take a longer time. But, compared to the entire process of development of a product-line, this effort is minimal.

In cases of SPL that are not new, where there are a large number of NFPs that were never documented and only exists, for example, in informal documents or in the minds of developers, there may be necessary to measure these properties again in order to find out how they were measured, and only after that they can be documented.

6.3 Threats to Validity

The case study validity presents the trustworthiness of the results, besides to what extent the results are not biased by the researchers subjective point of view (Runeson *et al.*, 2012). Thus, the threats to validity of this work are described next.

Construct Validity. We identified the following threat to construct validity and the strategy to mitigated it:

As the main researcher of this study also developed the SPL Notepad, she had a strong influence on the conclusions. To mitigate this threat and allow a different view of the framework and approach, another participant played the role of an engineer application, who was responsible for the main tasks of the product derivation process aware of NFPs.

Internal Validity. In our case study, we identified different threats, as follows:

The research questions defined in this case study may not focus on the most important aspects regarding application of both framework and reuse approach in the text editor desktop context. We mitigated this risk through discussions with SPL experts, reviewing important papers in the topic, and the case study (Da Silva *et al.*, 2014).

Both NFPs Framework and NFPs Reuse Approach have a couple of steps and tasks, which explain how to proceed to ensure that activities are performed correctly. However, it is possible some concepts may have been misinterpreted. To mitigate this risk, the researcher was, during the observation activity, all the time near the SPL engineer .

The text editor SPL used in this study may not be the most appropriate. It was not an example of a product-line where the analysis of NFPs were strongly recommended, since it is an academic small project addressed to the desktop context. In order to mitigate this threat, we intend to further investigate the application of our approach on other contexts, as for example in an mobile SPL, which was also developed by the RiSE Labs group.

External validity. As the case study was executed in one small academic SPL, which has a small set of features and products, it is difficult to make generalizations. The findings and discussions in this study are delimited for this SPL context. Thus, although the findings and discussions could be generalized for large SPL, as for example that ones of big companies, we should not, because the challenges and problems could be different.

Despite the limitations, researchers can extend the study by replicating it in different SPL contexts following the design of this study. The case study protocol was minutely elaborated following specific guidelines for software engineering (Runeson *et al.*, 2012).

6.4 Chapter Summary

This Chapter presented important details about the case study performed: objective, the case, units of analysis, data collection and analysis, results and the main findings of this activity.

The exploratory case study detailed on this Chapter aimed to investigate the applicability of

the NFPs Reuse Approach, and also consequently the NFPs Framework, in a text editor SPL. To do so, two SPL engineers were observed during the derivation process aware of NFPs. The output documents of the approach (registries and A-base), the activity of observation and the interview served as input to answer the case study research questions.

Next chapter presents a replicated case study applied in a different domain. Through this new study we intend to follow the same procedures adopted to perform the present exploratory study in order to obtain and compare results.

7

A Replicated Case Study

This Chapter presents the final step of this dissertation performed through a replicated exploratory case study from the case study presented in the previous Chapter. The study is conducted with an SPL in a different domain. Replications are a way to understand how much context influences the results, which allow that generalizations regarding the research questions can be made (Runeson *et al.*, 2012). They are based on the design and results of a previous study in which the main goal is to verify or broaden applicability of the results of the initial study (Shull *et al.*, 2002).

Although there are no specific guidelines for replications, as reported by Juristo and Gómez (2012), they suggest four types of information to include in a replication report: (i) information about the original study to provide enough context for understanding the replication; (ii) information about the replication to support readers on understanding its specific details; (iii) comparison of replication results with original study results in order to make clear the commonalities and differences in the results obtained; and (iv) conclusions across studies to provide significant insights that can be drawn from the series of studies that may not be obvious from a single study.

The replicated case study presented on this Chapter aims at investigating the NFPs Framework and NFPs Reuse Approach applicability in a product line of emergency applications for the mobile domain. This chapter presents the information about the replicated case study, as well as the comparison between the studies and conclusions. This remainder of this chapter is organized as follows. Section 7.1 presents the replicated research design. The results and findings are described on Section 7.2. Section 7.3 presents the case studies comparison. The threats to validity are discussed on Section 7.4; and Section 7.5 presents the chapter summary.

7.1 Case Study Protocol

According to Runeson *et al.* (2012), the protocol is an important source of information when the case study is subsequently reported, to demonstrate quality assurance or to support a replication by other researchers.

Replicated studies are commonly used as a way to generalize the results of the original case study for other contexts. In order to do this, this kind of replication uses exactly the same protocol as was used for the original case. For this reason, the description of a good and detailed case study protocol is of crucial importance, as well as making it available to other researchers.

The previous chapter presented an exploratory study that aimed to investigate the NFPs Framework and the NFPs Reuse Approach applicability in a text editors product line for the desktop domain. From that case study results, we were able to draw initial conclusions, lessons learned and improvements on the reuse process. In this chapter, we present a case study which aims to replicate the procedures conducted in the previous study. However, this replicated study has the mobile domain as SPL context. The purpose is to provide more evidence and discussions about the applicability of the reuse approach and also compare both study cases considerations.

Shull *et al.* (2002) explain that, on one hand, the replication where the same exact study is run can be used to verify results of an initial study. On the other hand, if the researchers want to understand the applicability of the results in a different context, then slightly modifications on the design of the original study are acceptable, which does not invalidate the study as a replication. For this replication in discussion, we performed the study as similar as possible to the original one, respecting the same protocol and guidelines.

A case study protocol has the research design of a case study, which is composed by (Runeson *et al.*, 2012): rationale and objective of the study, the bounded system or case, units of analysis, case study research questions, data collection instruments and data analysis procedures.

7.1.1 Rationale and Objective

As the original case study, the replicated one has an exploratory purpose. We undertook this study to further investigate the SPL product derivation process, taking into account the reuse of NFPs values. The objective is to evaluate the NFPs Framework and NFPs Reuse Approach in a different SPL context to provide more evidence on the applicability of them. In this way, the replication study intends to increase the generalizability of the initial results by providing similar protocol characteristics, including similar rationale and objective.

For the present study, we selected a sample SPL of a mobile domain, the RescueMe SPL. We

are particularly interested in investigating whether the results found in the text editor / desktop domain are similar for the emergency / mobile domain. The information of the (GQM) template (Basili *et al.*, 1994; Wohlin *et al.*, 2000) is the same for both studies, in which presents that the replicated study aims to:

- Analyze the **NFPs Reuse Approach**,
- for the purpose of **investigating**,
- with respect to its **applicability**,
- from the view point of **domain and application engineers**,
- in the context of a **small SPL**.

7.1.2 The Case

The product-line under study is an SPL for mobile applications that assists users in emergency situations. The RescueMe SPL was developed for the iOS[1] platform applied to smartphone devices, using the Objective-C language. This SPL is based on the Savi application[2], both developed in the RiSE Labs. The development team was composed of one post-doctoral researcher, five Ph.D. students, two M.Sc. students and two B.Sc. students.

The RescueMe products aim to help its users in emergency and dangerous scenarios. The main screen of all products presents a red button, pressed by the users to send messages for all RescueMe contacts in the list. Optionally, contacts can be reached through social networks, and they can track the RescueMe user by checking the updated location in a map (Vale *et al.*, 2014). Figure 7.1 shows two RescueMe screenshots: the main screen (red button) and the import contacts screen.

The major features were identified by analyzing two sources: the single system previously developed in the RescueMe project and the similar products in the Apple Store[3]. The similar products are: Help Me![4], RescueMe Now[5], Rescue Button[6], and Red Panic[7]. A set of nine

[1] iOS platform - https://www.apple.com/ios/
[2] Savi on Apple Store - https://itunes.apple.com/us/app/savi/id590385285
[3] Apple Store - http://store.apple.com/
[4] Help Me! - https://itunes.apple.com/au/app/help-me/id510561786
[5] RescueMe Now - https://itunes.apple.com/us/app/rescuemenow/ id330785171
[6] Rescue Button - http://linskyapps.webs.com/
[7] Red Panic - http://www.redpanicbutton.com/

Figure 7.1: RescueMe screenshots.

major features emerged: `Access_Control`, `Contact`, `Destination`, `Emergency_Numbers`, `Tracking`, `Location`, `Language`, `User_Info` and `About`.

Then, it was identified five products using the criteria of incorporating only basic features for simple products and providing more complex features for the other ones. The RescueMe SPL has 29 features spread over 2,504 lines of code, divided into 28 files and 28 classes. The SPL is composed by 5 products, from 10 to 28 features each. The products are: *RescueMe Lite*, *RescueMe Standard*, *RescueMe Social*, *RescueMe Pro* and *RescueMe Ultimate*. Figure 7.2 shows this relation among the products, where the *Lite* is the simplest version, with 10 features, and the *Ultimate* version is the most complete one, with all the 28 features. Moreover, Appendix C.1 shows the ResecueMe SPL feature model, and Appendix C.2 shows the product map, which present in more details the features of each product.

The RescueMe SPL was implemented using Objective-C language with XCode IDE[8] version 4.6.2, using a Model-View-Controller (MVC) architectural pattern. The chosen variability implementation technique was conditional compilation, since XCode provides support for pre-processor directives through macro definitions (each macro has the same name of the feature). Only optional and alternative features had been implemented with pre-processor directives. An example of conditional compilation in RescueMeSPL is showed in the Figure 7.3.

[8]XCode - http://developer.apple.com/xcode/

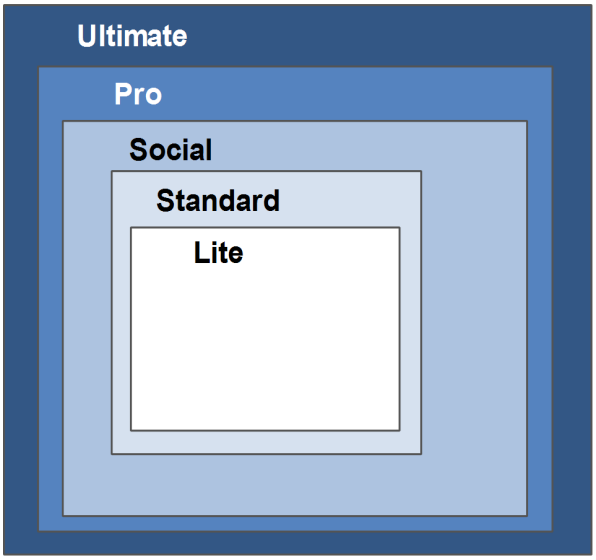

Figure 7.2: Relation among products in RescueMe SPL.

```
1  ⋮
2  @interface ImportContactViewController ()
3  #ifdef FACEBOOK_IMPORT
4  @property (retain, nonatomic)
       FBFriendPickerViewController *
       friendPickerController;
5  #endif
6  ⋮
```

Figure 7.3: Excerpt code from the *ImportContactViewController.m* (feature `Facebook_Import`) (Vale *et al.*, 2014).

The product derivation process performed for the RescueMe SPL is supported by the preprocessor directives implemented in the source-code for the RescueMe-SPL features. In XCode, the products are represented by targets. The SPL engineers configure the targets by selecting the macro definitions related to the features included in the respective products.

7.1.3 Units of Analysis

The units of analysis for this replicated study are the same of the original study: the NFPs Framework and the NFPs Reuse Approach. For the reuse approach, we are mainly interested in evaluate two artifacts: *A-base* and *Configuration Filter*, which were specially created to support the reuse approach.

7.1.4 Case Study Research Questions

The main objective of the original study was to investigate the application of the NFPs Reuse Approach inside an SPL context. This replication has the same purpose, but it also aims at providing more evidence and discussions about the applicability of the reuse approach and compare both study cases results. The most significant difference is the SPL under study, which is an SPL for emergency situations for the mobile domain.

In order to address this objective, small goals based on it can also be defined. These small goals are:

- to apply the reuse approach to the SPL product derivation process of the RescueMe SPL;
- to analyze the effort to specify NFPs values according the NFPs Framework guidelines;
- to assess the results in order to provide the lessons learned and improvements on the reuse process; and
- to compare the results of the replicated study with the results of the original study.

Since the objective is similar to the initial study, the research questions are exactly the same, which evaluate the case under the domain and application engineers perspective. In this replicated case study, engineers are responsible for applying the NFPS Reuse Approach concepts into life-cycle of the RescueMe SPL. In this way, the set composed by four research questions can be found at Section 6.1.4 from previous chapter.

Table 7.1 has the summary of the research questions. For more details, the GQM template of the original study, Table 6.1 (previous chapter), presents more details about these questions and it is totally applied for both original and replicated study.

Table 7.1: Research Questions

Question	Q1	What is necessary to learn to start using the approach?
Question	Q2	Does the NFPs Reuse Approach avoid unnecessary (re)analysis of NFPs values?
Question	Q3	What is the effort spent to fill the A-Base with new instances of attributes?
Question	Q4	What are the drawbacks and benefits of the NFPs Reuse Approach?

7.1.5 Data Collection

In spite of the two SPLs being of different domains, desktop (original case) and mobile (replicated case), they have similarities. The replicated case study researcher has also a face-to-face contact with the SPL engineers and a free contact with the SPL in study. Both are academic SPLs developed by the same researcher group, RiSE Labs. In this way, thanks to those similarities and the replication characteristics, this replicated study adopted the same three data collection methods: documentation analysis, observation, and interviews.

It is worth highlighting that the product derivation process using the reuse ideas from the NFPs Reuse Approach was also observed during over 1 hour. Furthermore, the interview followed the same protocol, including the same interview questions (Appendix B.3).

7.1.6 Data Analysis

In order to perform the direct qualitative data analysis on the application of the reuse approach inside the RescueMe SPL (mobile) context, the SPL engineers were submitted to the same three activities of the original study: initial phase, observation and interview.

Activity 1 - Initial Phase. A brief explanation on the NFPs Framework key tasks and the flow of steps of the NFPs Reuse Approach were performed.

Activity 2 - Observation. This activity aimed to observe the SPL engineers in practice, i.e., using the NFPs Reuse Approach to derive the SPL products. In order to analyze how engineers conduct the reuse approach steps, a workflow similar to the original study was used. Thus, the RescueMe SPL engineers were asked to:

1. choose a couple of NFPs for the RescueMe products and specify them in the way of Attribute Types. Documentation used: Attribute Types Registry, Attribute Metadata Registry;

2. measure those NFPs for two SPL products, such as RescueMe Lite and RescueMe Social;

3. specify and document the measured NFPs values in the way of Attribute instances, according to the NFPs Framework. Documentation used: A-base (repository with the instances);

4. derive a new product with a specific NFP need. In this case, the other three RescueMe SPL products were generated, one at a time. They derived a product and created a Configuration filter with the specific need chosen, as presented on Figure 5.2. Documentation used: Feature Model and Product Map.

5. analyze the instances retrieved from A-base according to the filter that has been drawn, process referred as “Reuse Diagnosis activity”.

Activity 3 - Interview. The interview was applied just after the Observation Activity, following the form of Appendix B.3. The audio file has fifteen minutes, and it was transcript to facilitate the analysis process.

7.2 Results and Findings

This section discusses the findings of the NFPs Reuse Approach application on the RescueMe SPL. The replicated case study was based on three main activities: Initial Phase, Observation and Interview. Table 7.2 and Sections 7.2.1 and 7.2.2 present the details of the two first activities and Section 7.2.3 discusses about the interview results.

Table 7.2: Replicated Study General Data

Activity	Addressed to	Time
Initial activity: brief explanation on - NFPs Framework - NFPs Reuse Approach	Application Engineer	60 min.
Observation 1: NFPs Reuse Approach domain phase - Populating A-base	Domain Engineer	35 min.
Observation 2: NFPs Reuse Approach application phase - Product Derivation	Application Engineer	60 min.

The RescueMe SPL was developed by ten researchers of the RiSE Labs group, and two of them participated on this study. They are M.Sc. students with three years of experience in SPLE. Similar to the original case study, one of the researchers played the role of a domain engineer and the other was the application engineer. The *Initial Activity* shown on Table 7.2 was applied to the application engineer, since the domain engineer was the author of this work. This step was a 60-minutes-activity and followed the specification of Appendix B.3.

The *Observation Activity* was performed with the same characteristics of the observation applied on the initial case study. The activity has two different parts, the first one for the domain engineer and the second for the application engineer.

7.2.1 Observation activity part 1

This activity was responsible by the NFPs Reuse Approach application during the domain engineering SPL phase, and comprised the three first points of the observation workflow (shared in five points) presented on previous section. Following the reuse approach guidelines described on Chapter 5.3, the first step to perform is the population of the A-base with the NFPs of interest for the SPL in study. A-base is the NFPs Reuse Approach repository where NFPs values are stored. The other steps are played during the application engineering SPL phase.

In this way, the domain engineer specified two NFPs, `Usability` and `Social_Network-Interaction`, and documented them using the Attribute Type Registry defined on the NFPs Framework (Chapter 4.3.2). In addition, the metadata with the information used to describe the context in which NFPs are analyzed were also documented using the Metadata Type Registry (Chapter 4.3.4). The specification of these registries represents the observation workflow point 1. Figure 7.4 shows the Attribute Type Registry, where the Attribute Types `Usability` and `Social_Network_Interaction` were defined. For the metadata registry, the Metadata Types were the same used in the original case study (Figure 6.6 on previous chapter).

Attribute Type Registry	
TypeID:	Usability
Attributables:	Product
Variability:	Optional
DataFormat:	{Qualitative, Low/Medium/High}
Description:	Ease of use of products specified by experts
Documentation:	no
TypeID:	Social_Network_Interaction
Attributables:	Product
Variability:	Optional
DataFormat:	{Qualitative, Low/Medium/High}
Description:	Level of social network interaction of the product specified by experts
Documentation:	no

Figure 7.4: Attribute Type Registry for RescueMe SPL (part 1 of 2)

The domain engineer specified the two Attribute Types of Figure 7.4 based on the RescueMe

SPL feature model, product model and also on her expertise about the development of the product line. Both attributes are optional and related to products. For each Attribute Type, the domain engineer defined qualitative values analyzing the NFPs per product of the product model:

- Product: RescueMe-Lite -> Usability: Low.
- Product: RescueMe-Ultimate -> Usability: High.
- Product: RescueMe-Lite -> Social_Interaction: Low.
- Product: RescueMe-Social -> Social_Interaction: High.

Even without the concrete products, experts can evaluate NFPs in terms of percentages, approximated values or qualitative means. This process of analysis represents the observation workflow point 2.

From this qualitative analysis, Attribute Instances were created and documented, two for `Usability` and other two for `Social_Interaction` (workflow point 3), shown on Figure 7.7. As can be seen in this figure, the instances have validity conditions. For example, the product RescueMe-Ultimate has a *High* usability because of the presence of feature `How_to_Use`, and the product RescueMe-Social has a *Low* social interaction since it does not have the features associated to the social networks, such as `Facebook_Import` and `Twitter_Import`. These instances form the A-base for the RescueMe SPL.

In order to document Attribute Types and Instances, electronic spreadsheets were used. The time needed to perform all these tasks, qualitative analysis and documentation, was in average 35 minutes.

7.2.2 Observation activity part 2

This part of the observation activity lasted over 60 minutes and aimed at observing the product derivation process under the NFPs Reuse Approach perspective, similar to the original case study. During this activity, the application engineer was asked to choose a couple of NFPs important to the RescueMe SPL and derive the products thinking on what NFPs the products could be associated. This engineer is not the same person that performed the *Observation Activity part 1*. In that activity, the domain engineer was the author of this work. In order to provide to the application engineer enough knowledge to perform this activity, he was firstly submitted to a brief explanation on the framework and reuse approach, which corresponded to the *Initial Activity* of Table 7.2.

This part 2 of the observation included all the five points of the observation workflow presented on previous section (7.1.6). Hereafter, it will be presented the results of each workflow point.

1. He chose two NFPs of relevance to the mobile domain and RescueMe products, `Cpu_Usage` and `Memmory_Consumption`. These NFPs had not been documented in the way of Attribute Types yet. Thus, two new entries were added in the Attribute Types Registry, as shown on Figure 7.5. From here, the Attribute Types Registry can be seen as a set composed by the types from Figure 7.4 and Figure 7.5. During this process, he was also asked to analyze the Metadata Registry to verify if other metadata could be added. However, no new Metadata Type was included.

Attribute Type Registry	
TypeID:	CPU_Usage
Attributables:	Product
Variability:	Optional
DataFormat:	{Quatitative, percentage}
Description:	The percentage of the CPU used by the process.
Documentation:	no
TypeID:	Memory_Consumption
Attributables:	Product
Variability:	Optional
DataFormat:	{Quantitative, MegaBytes (MB)}
Description:	The amount of real memory used by the process.
Documentation:	no

Figure 7.5: Attribute Type Registry for RescueMe SPL (part 2 of 2)

2. In order to perform this study, the application engineer had previously generated only two RescueMe products: RescueMe-Lite and RescueMe-Social. Thus, he measured the NFPs on these two available SPL products:

   - Product: RecueMe-Lite

     -> Cpu_Usage: 25.6%,

     -> Memmory_Consumption: 39.28 MB.

   - Product: RecueMe-Social

     -> Cpu_Usage: 27.2%,

     -> Memmory_Consumption: 22.67 MB.

3. At this stage, he specified and documented the measured NFPs values in the way of Attribute instances, according to the NFPs Framework, as shown on Figure 7.8. Thus, from here the A-base is composed by the instances from Figure 7.7, and Figure 7.8.

4. At this moment of the observation, the application engineer was asked to attribute NFPs analysis to the other three RescueMe products: RescueMe-Standard, RescueMe-Pro and RescueMe-Ultimate. In this way, each of these products were derived and analyzed, one at a time. Appendix C.2 shows the product map of this SPL.

   For each product a specific Configuration Filter was also specified:

   - Product: RescueMe-Pro
     Configuration Filter: (Source = Measurement) AND (TypeID = CPU_Usage)
   - Product: RescueMe-Ultimate
     Configuration Filter: (TypeID = Memory_Consumption)
   - Product: RescueMe-Standard
     Configuration Filter: (Source = Measurement)

5. The application engineer evaluated the results of each filter one at a time:

   (a) Until this stage of the replicated case study, the application engineer had derived only two products: RescueMe-Lite and RescueMe-Social. His motivation to firstly derive the RescueMe-Pro was due to the similarity between the Lite and Pro versions. This last product only has 1 more functionality than the former: feature `Web_Tracking`. Thus, the configuration filter was created to attend this purpose: to search for measurement values that might be similar to reuse on the Pro version.

   Since the engineer chose to analyse only *CPU_Usage*, this first Configuration Filter is able to filter from the A-base the first two instances from the Figure 7.8. These values were:
   -Cpu_Usage (RescueMe-Lite): 25.6%;
   -Cpu_Usage (RescueMe-Social): 27.2%.

   As the engineer is an expert on the RescueMe SPL, from these instances, he realized that the RescueMe-Pro version would have a value of *CPU_Usage* less than 30%. This analysis fits to the Reuse Diagnosis case 1 (NFPs Reuse Approach), since the engineer classified the filtered values as useful, where no other measurement needs to be computed.

(b) The second Configuration filter aimed to filter all the values that were analyzed for the Attribute Type `Memory_Consumption`. These values were:
-Memmory_Consumption (RescueMe-Lite): 39.28 MB;
-Memmory_Consumption (RescueMe-Social): 22.67 MB.

The engineer is interested in find out if these values can be used for the RescueMe-Ultimate product. However, he realized that there were few values specified on the A-base (Figure 7.8), and that was not possible to perform any analysis or comparison to the target product. According to the Reuse Diagnosis activity, this analysis fits to case 3, where no values can be reused and the SPL engineer has to compute the NFP value for the product.

(c) The third Configuration Filter retrieved the instances of the two previous filters. Thus, the engineer can visualize all the values that were measured for the RescueMe SPL. After analyzing the values retrieved from the filter, the application engineer realized that for the RescueMe-Ultimate version, which is the most complete version of this SPL, another NFP would be interesting to be calculated. The engineer would like to compute the required space on disk to install this product on a smartphone. Since, he is providing a new measured value, this case fits to the case 3 of the Reuse Diagnosis Activity.

In order to perform this analysis, the first step was to document the new Attribute Type, which was named `Space_on_Disk`. Figure 7.6 shows the Attribute Registry with this specification, and Figure 7.9 presented the Attribute Instance created from this type for the RescueMe-Ultimate.

Attribute Type Registry	
TypeID:	Space_on_Disk
Attributables:	Product
Variability:	Optional
DataFormat:	{Quantitative, MegaBytes(MB)}
Description:	The amont of secondary memory that the product uses to be installed.
Documentation:	no

Figure 7.6: New Attribute Type added to the Attribute Registry for RescueMe SPL

A-base (Attribute Instances)			
TypeID: Usability			
1	**ProductName/FeatureName:**	Product: RescueMe-Lite	
	Data:	Low	
	Metadata:	Creation_Time:	10.09.2014-16:16
		Version:	1.0
		Source:	Estimation
		Complexity:	Low
	ValidityConditions:	Feature	The low usability is justified because this product does not have the feature "How_to_use"
2	**ProductName/FeatureName:**	Product: RescueMe-Ultimate	
	Data:	High	
	Metadata:	Creation_Time:	10.09.2014-16:17
		Version:	1.0
		Source:	Estimation
		Complexity:	Low
	ValidityConditions:	Feature	This variant has a guide (feature "How_to_use") that supports the user on using the application
TypeID: Social_Network_Interaction			
1	**ProductName/FeatureName:**	Product: RescueMe-Lite	
	Data:	Low	
	Metadata:	Creation_Time:	10.09.2014-09:45
		Version:	1.0
		Source:	Estimation
		Complexity:	Low
	ValidityConditions:	Feature	The low interactiom is justified because this product does not have the features "Facebook_Import" and "Twitter_Import"
2	**ProductName/FeatureName:**	Product: RescueMe-Social	
	Data:	High	
	Metadata:	Creation_Time:	10.09.2014-09:45
		Version:	1.0
		Source:	Estimation
		Complexity:	Low
	ValidityConditions:	Feature	This variant has the features that interact with facebook and twitter

Figure 7.7: A-base (Part 1 of 2) with usability and social network interaction.

A-base (Attribute Instances)			
TypeID:	**CPU_Usage**		
1	**ProductName**	Product: RescueMe-Lite	
	Data:	25.6%	
	Metadata:	Creation_Time:	11.09.2014-12:21
		Version:	1.0
		Source:	Measurement
		Source_Tool:	Intruments Apple tool
		Complexity:	Medium
	ValidityConditions:	Platform:	Iphone 5, iOS 8
		Execution Flow:	After opening the app, the red button was pressed.
2	**ProductName**	Product: RescueMe-Social	
	Data:	27.2%	
	Metadata:	Creation_Time:	11.09.2014-12:29
		Version:	1.0
		Source:	Measurement
		Source_Tool:	Intruments Apple tool
		Complexity:	Medium
	ValidityConditions:	Platform:	Iphone 5, iOS 8
		Execution Flow:	After opening the app, the red button was pressed.
TypeID:	**Memory_Consumption**		
1	**ProductName**	Product: RescueMe-Lite	
	Data:	39.28MB	
	Metadata:	Creation_Time:	11.09.2014-12:21
		Version:	1.0
		Source:	Measurement
		Source_Tool:	Intruments Apple tool
		Complexity:	Medium
	ValidityConditions:	Platform:	Iphone 5, iOS 8
2	**ProductName**	Product: RescueMe-Social	
	Data:	22.67MB	
	Metadata:	Creation_Time:	11.09.2014-12:31
		Version:	1.0
		Source:	Intruments Apple tool
		Complexity:	Medium
	ValidityConditions:	Platform:	Iphone 5, iOS 8

Figure 7.8: A-base (Part 2 of 2) with CPU usage and memmory consumption.

A-base (Attribute Instances)			
TypeID:	**Space_on_Disk**		
1	**ProductName/FeatureName:**	Product: RescueMe-Ultimate	
	Data:	4.7MB	
	Metadata:	Creation_Time:	11.09.2014-12:12
		Version:	1.0
		Source:	Measurement
		Source_Tool:	Settings Apple tool
		Complexity:	Low
	ValidityConditions:	Platform:	Iphone 5, iOS 8

Figure 7.9: New Attribute Instance: space on disk.

7.2.3 Research Questions

The interview was performed just after the observation activities parts 1 and 2. The RescueMe application engineer was interviewed for 18 minutes. The interview-audio and the observation activities served as a basis to answer the case study research questions. The transcription of the interview lasted over 30 minutes.

7.2.3.1 Q1: What is necessary to learn to start using the approach?

For the RescueMe application engineer responsible to derive the products during the Observation Activity part 2, the framework is detailed enough to understand its concepts and guidelines. Each framework keyword was defined in different tasks of the framework, and according to the engineer, this organization facilitated the understanding of each part separately. For the application engineer, the initial 60-minutes activity was enough to learn the goals and motivations of the proposed approach.

However, when he had to specify the first Attribute Instance, he pointed out some difficulties. For example, since the metadata had been defined by the other engineer (domain engineer during the Observation Activity part 1) he had some difficulty to remember what he should specify. But, once the engineers have a free and easy access to the Metadata Registry, this situation can be solved. In the RescueMe case, the registries were specified in common electronic spreadsheets, so that anyone from RescueMe project can view the information.

Other aspect highlighted by the engineer was regarding the process of assembling the configuration filter. He knew what he would like to set in the filter, but how to do it and which words to use was not clear. In order to accomplish this task, the engineer had to first take a look on the Task 5 of the NFPs Framework (Chapter 4.3.5). He also reported that, for the products derived during the part 2, just simple filters were mounted, and perhaps, in cases where more elaborate filters are needed, he would probably have more difficulties to describe them. Although the engineer reported this difficulty, he stated that it does not sound complicated to define a filter based on its *matching conditions*, just requires a little more dedication to understand the conditions to apply them in more complex filters.

7.2.3.2 Q2: Does the NFPs Reuse Approach avoid unnecessary (re)analysis of NFPs values?

According to the application engineer, the reuse provided by the approach is useful in cases such as that one performed between the RescueMe-Social and RescueMe-Pro. For that case,

the information about the CPU consumption was reused and the engineer judged that it was not necessary to measure it again. The products were very similar and the reuse was implemented due to the expert knowledge on the RescueMe project and SPL artifacts, mainly the product map. Still according to the engineer, the reuse is possible but limited to this kind of case, where there are very similar products in the SPL. Considering large SPLs with a significant number of products, this situation is possible to occur.

For mobile applications, as for example the RescueMe products, the engineer reported that the analysis of NFPs might be essential to produce high quality products. Smartphones, like IPhones which are the targets of this SPL, are devices with limited physical resources and the developers must present guarantees that the products would not fail due to such limitations. The NFPs Framework is an alternative to standardize non-functional properties and facilitate the NFPs analysis.

7.2.3.3 Q3: What is the effort spent to fill the A-Base with new instances of attributes?

In cases of large SPLs, the engineer believes that the initial effort spent to document Attribute Types and Metadata Types might be relevant. For him, there is an initial effort, however, it is low when compared to the gain that it would provide, especially if the re-analysis of NFPs can be avoided. For the RescueMe SPL, this effort was low, since it is an small SPL with a couple of features and products. The engineer took less than 5 minutes to create an instance for the first time, and less than 3 minutes to create a new Attribute Type.

Since this SPL is focused on mobile applications and specifically on Apple applications, which has its own development constraints[9], many NFPs are measured at runtime, spending time and effort. For example, to measure runtime NFPs, such as those ones used during the observation activity (Cpu_Usage and Memmory_Consumption), a developer needs a MacBook with the OS X (Apple operating system) updated and compatible to the iOS (Apple mobile operating system) installed on the IPhone, and also at least two Apple development programs: XCode[10] and Instruments[11], to visualize the NFPs values.

According to the application engineer, reusing NFPs values of a product in other product is an alternative to not waste time in measuring then again. In this way, since the process of measuring NFPs can be avoided, the effort of the documentation can be considered low.

[9] iOS Developer Library - https://developer.apple.com/library/IOs/navigation/
[10] https://developer.apple.com/xcode/
[11] https://developer.apple.com/library/mac/documentation/DeveloperTools/Conceptual/InstrumentsUserGuide

7.2.3.4 Q4: What are the drawbacks and benefits of the NFPs Reuse Approach?

The main benefit pointed out by the application engineer was the possibility of avoiding NFPs measurements when there are products similar to the one in the product derivation process. This possibility makes the analysis of NFPs faster and effortless. In respect to the NFPs framework, he characterized it as simple, besides providing a detailed way of documenting NFPs, through the Attribute Type, Attribute Instance and Metadata.

The engineer reported three points as drawbacks. The first one has already been mentioned: the definition of a configuration filter. For him, as the number of instances increases, the filters scope will now become more limited and thus needs a previous and deeper study of the *matching conditions* for creating valid filters. However, for the RescueMe SPL, the filter was defined only to demonstrate its use, since there was only a couple of instances documented on electronic spreadsheets.

The second point was related to the initial effort to document the registries, *Attribute Type registry*, *Metadata registry* and *A-base*. Despite this effort be small, it does exist. But, as the engineers are becoming familiar with the framework, it becomes increasingly smaller. The benefits of avoiding NFPs re-analysis outweigh the effort.

The last point reported by him was about disagreements that may exist among SPL project members regarding the values of NFPs. For example, an engineer can specify a value in which another engineer disagrees in how it was analyzed (estimated/simulated /measured). However, the *NFPs Framework* defines the *metadata* to help in this situation. The information on the metadata helps to understand the context in which the value was analyzed. But, if even so the engineer does not agree, he can create another instance with another value, and presents how it was analyzed through the metadata, where it is possible to see the differences between then. The metadata are an important part of the Attribute Instance and must be carefully described to minimize questions of how the value was analyzed.

7.3 Comparative Analysis

In this section, we aim at comparing the results obtained in applying the NFPs Framework and NFPs Reuse Approach guidelines in both Notepad and RescueMe SPLs.

General Comparison. The Notepad SPL version 2 is an academic product line developed by two master students from the RiSE Labs group. The SPL is focused on the desktop domain, implemented in JAVA language using Model-View-Controller (MVC) as architectural pattern. The Notepad SPL has 1,803 lines of code and 14 java class. The Conditional compilation (CC)

was the technique used to perform the variability among the SPL products, through an Eclipse plugin named CIDE. Notepad SPL contains 40 features and 3 main products: Notepad Lite, Notepad Standard and Notepad Ultimate, each one with 10, 31 and 35 features respectively.

The RescueMe SPL is also an academic SPL developed by the RiSE group, but implemented by a group of 10 researchers. This SPL has 5 applications for the mobile domain, which aim to help the users in dangerous situations. The product line was developed for Apple devices, using the Objective-C language. The RescueMe SPL has 2,504 lines of code and 28 class. Similar to the Notepad V2 SPL, the RescueMe was also implemented based on the MVC architectural pattern and using the CC as the variability technique, but in this case the CIDE plugin was not used. The Objective-C support conditional compilation by use of pre-processor directives, IFDEFs. The RescueMe SPL is composed by 29 features and 5 products. Table 7.3 compares the general information of both SPLs.

Table 7.3: Comparative table with the Notepad and RescueMe SPLs.

	Notepad V2 SPL	***RescueMe SPL***
Origin	Academic, RiSE group	Academic, RiSE group
Domain	Desktop	Mobile
Language	Java	Objective-C
Development Environment	Eclipse IDE	XCode
Variability Technique	Conditional Compilation (CIDE tool)	Conditional Compilation (IFDEFs)
Architectural pattern	MVC	MVC
Line of code	1,803	2,504
Number of Class	14	28
Number of features	40	29
Products	3	5
Features - Mandatory - Alternative - Optional	40 20 4 15	29 11 0 17
Qualitative NFPs analyzed	Performance and Usability	Usability and Social Network Interaction
Quantitative NFPs analyzed	Footprint and Memory Consumption	CPU Usage, Memory Consumption and Space on Disk

Execution of the *Initial Activity*. The data analysis of both studies, the original and the

replicated one, was based on three main activities: *Initial Activity* and *Observations parts 1 and 2*, as presented on Table 6.2 for the Notepad SPL and Table 7.2 for the RescueMe SPL. From the results of the first case study, we realized that the *Initial Activity* could be more detailed. In this way, the time spent to explain the framework and reuse approach was duplicated for the replicated study. During the documentation of the first attributes by the RescueMe application engineer, it was observed that he performed the tasks more easily than the Notepad application engineer.

We realized that even both engineers had reported that the NFPs Framework and NFPs Reuse Approach are of easy understanding, it is recommended to devote time to study them so that there are no doubts in their application. Another aspect commented by the Notepad SPL engineer was regarding the importance of providing a guide in the way of "How to do" steps to support during the creation of the Attribute Types, Metadata and Attribute Instances. In order to do this, for the replicated study, we provide the engineer printed documents showing the fields that need to be filled in during each moment of the NFPs analysis.

Execution of the *Observation Activities*. For both case studies, the researcher took over the role of the domain engineer. However, the Notepad SPL was developed only by two people, in which the engineer had a major participation and influence. This aspect was a facilitating factor in the application of the approach. The RescueMe project was developed by a larger group of people, where each person had a well-defined role. Although she participated in the development of the two product lines, in the second she had a slightly more limited knowledge.

Thus, the NFPs analysis for the RescueMe SPL products was a bit more complicated than for the first case study. This was caused due to the RescueMe project had already been finalized, and in this case, the development environment had to be reset (the programs necessary to derive the products). The engineers had problems, for example, in generating more than those five products pre-designed to the SPL, and they had to adequate the case study to use only them. Even those responsible for the project and the researcher being from the same research group, there were difficulties in finding them and arrange meetings.

Effort in applying the approach. The application engineers of the two case studies, which were responsible for deriving the products aware fo NFPs, during the Observation Activity part 2, reported being concerned with the initial effort to use the framework and reuse approach in large product lines. This point, which was common between the studies, addresses the effort spent in understanding the NFPs Framework in order to document the first *Attribute Types* and *Attribute Instances*. Although there is effort, the engineers classified it as low because of the benefits the framework can bring. According to them, the framework is clear and divided into tasks that are cohesive enough to facilitate their understanding.

Regarding the NFPs Reuse Approach, the engineers highlighted two main points: (i) it might support the reuse of NFPs avoiding unnecessary re-analysis for very close products, and (ii) it serves as a derivation guide such a way the SPL engineers can also have access to the NFPs information before and after the derivation of a product.

Non-functional properties. The NFPs used during the Observation Activities of the original case study (Figure 6.5) and the replicated one (Figures 7.4 and 7.5) were chosen at the same way. Firstly, the domain engineer thought in NFPs that could be general for any product. Then, she supported the application engineers in finding out which NFPs were important to the SPL products. This task of defining the NFPs by the application engineers was simple and took little time, since the engineers had a deep understanding of the particularities of each product line.

7.4 Threats to Validity

Like the original case study, this replicated study presents some threats to validity, which are described as follow.

Construct Validity. Similar to the first case study of previous chapter, the main researcher of this study also developed the SPL in study. In this way, she had a strong influence on the conclusions of this replicated study too. To mitigate this threat, besides other engineer having participated in the study in the main role of deriving products (the application engineer), we intend to further investigate the application of our approach on other kinds of SPL. In this case, the SPL would have different engineers, where the researcher would not participate in the development of the product line and would not assume the role of the domain engineer.

Another aspect related to the case study researcher is about her knowledge on the SPLs in study. The researcher participated in the developed of both Notepad SPL and RescueMe SPL, but under different conditions. For the second SPL, which was developed by more engineers than the first SPL, she had a bit more limited knowledge. Differences in the engineers knowledge do not prevent the implementation of a study, it is only essential the responsible to define quality attributes are part of the SPL development group and be aware of requirements and products of the product line.

Internal Validity. For this replicated case study, we identified different threats, as follows:

The replicated study research questions are the same of the original study. For that study, there was a threat related to the possibility of the research questions do not focus on the most important aspects regarding application of both framework and reuse approach. We mitigated this risk for the replicated study through discussions with SPL software engineers on how to

conduct a replicated study and reviewing important papers that approaches this topic. The execution of a replication is also a way to reinforce that the research questions are consistent with the object under study.

However, both SPLs are small academic projects from the same research group. In this way, there are not buyers who asked for specific NFPs not evaluated by the engineers. Thus, we could not evaluate the case where real clients buy for products with required NFPs. To mitigate this threat, we simulated this situation asking the application engineers to choose a couple of NFPs they believe be important to end users. In addition, due to the second study be aimed to mobile applications, before running the study, the RescueMe application engineer were already aware of at least what are the main properties that they would like to assess and that could be of interest to customers.

Regarding to these properties, the NFPs, there is a threat with respect to the different NFPs used in the SPLs studied. For the Notepad SPL, four NFPs were identified, and the RescueMe SPL case study used five. Among them, only two were common between both SPLs: *Usability* and *Memory Consumption*. In order to better mitigate this threat, we intend to perform other case studies with the same set of NFPs to analyze and compare the results.

External validity. A common threat towards external validity is related to generalizations. In order to minimize this threat, we performed two case studies aiming at providing more findings and discussions on the NFPs Framework and NFPs Reuse Approach. The case studies were strategically chosen because they are from different domains. The first was applied on a text editor SPL for desktop, and the second was applied on emergency applications of a mobile domain.

Another way to provide further generalizations is to replicate this same study on large SPLs to analyze whether the findings would be similar to those which were discussed in the results of small academic product lines. To mitigate this threat, we intend to apply our approach on real software companies.

7.5 Chapter Summary

This Chapter presented a replicated exploratory case study based on the same protocol of the study presented on previous chapter. We aimed to extend the original study by replicating it in a different SPL context following the same original design. That original protocol followed important case study guidelines specified by Runeson *et al.* (2012), which facilitated its replication.

The replicated case study detailed on this Chapter also aimed at investigate the applicability of the NFPs Reuse Approach, and consequently the NFPs Framework, but now, in an SPL for mobile applications. The case study analysis was based on two SPL engineers that were responsible to apply the approach on the RescueMe SPL. A new contribution was a comparison performed between the two studies, original and replicated ones.

Next Chapter presents the research contributions, future work and concluding remarks of this dissertation.

8

Conclusions

This dissertation presented the conduction of a systematic literature review, from which we were able to understand how NFPs are handled in the SPL context. The objective of the review was to present a holistic overview of the existing studies that had been reported regarding the analysis of NFPs.

From this literature review, where it was possible to obtain an overall view of the analysis of NFPs in the product lines area, we observed the management of NFPs still is a remaining challenge. Each review primary study had its own way of describing the NFPs, and for the same attribute it was possible to find, for instance, different concepts and units of measure. This lack of a systematic and uniform specification of NFPs for SPL was identified as an interesting gap that could be addressed.

Thus, a framework that aims at providing this systematic quality attribute specification was proposed. It allows a clearer integration process of NFPs with features and SPL products into the product derivation process. The NFPs Framework was initially based on a work for component-based embedded systems (Sentilles, 2012).

The NFPs Framework was described throughout five tasks in order to provide the specification of: (i) the formal definition of an attribute (Task 1); the type of an quality attribute, Attribute Type (Task 2), ; (ii) its values, Attribute Instances (Task 3); (iii) the context in which the values were obtained, Metadata Type (Task 4); and (iv) a way to select a value from a large amount of possible instances, Configuration Filter (Task 5).

In order to allow a product derivation process aware of NFPs, a reuse approach was also proposed. The NFPs Reuse Approach intends to define a systematic way of reusing previous NFPs values in order to minimize the effort of performing a new analysis. The main artifact of this approach is a repository where it is possible to store the attribute values/instances, namely A-base.

Further, the NFPs Reuse Approach was evaluated through a case study performed in an SPL text editor desktop context. Such a way to analyse the applicability of the approach, and consequently, of the framework, an activity of observation and an interview were performed. The case study highlighted that the approach can also be used as a derivation guide. From the A-base, stakeholders can find quality characteristics of the general SPL, through the quality of (sub)products and features, before, during or after the derivation of a product.

A replicated case study was also performed in a different domain. The objective was to provide more evidences and discussions about the applicability of the framework and reuse approach, and also present a comparison between the original and replicated study. This replicated study was applied on a product line of emergency applications for the mobile domain. From these results, it was possible to increase the generalizability of the initial results, since the approach was now performed in two different kinks of domain: desktop and mobile.

8.1 Published Work

The work reported in this dissertation has resulted in one publication (Soares *et al.*, 2014) at the 40th IEEE EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), in the year of 2014. Additional papers concerning the framework, reuse approach and the case study have been written to submit for other conferences on the next months.

8.2 Future Work

From the results and findings obtained in this dissertation, we have identified a number of aspects that can be investigated in the future, as described next.

- **Extended literature review.** The review should be extended in order to get a deeper understanding on the differences and similarities among the three categories of approaches: prediction, estimation and feature selection focused. Also, during the research method definition, a couple of other research questions could be investigated, but due to a restriction of scope, we chose to reduce it.

- **Case studies applied in different scenarios.** The case study should be replicated for other domains and scenarios. We intend to further investigate the application of our reuse approach and framework on an industrial scenario, in order to perform a cross case analysis among the industrial SPL and the other two academic SPL, the text editor SPL

and the mobile SPL. In this way, more interviewees may provide additional data, which may improve the assessment of the approach too.

- **Definition of an ontology for the NFPs Framework.** Ontology presents the set of basic terms and relations comprising the vocabulary of a topic of an area, as well as the rules for combining terms and relations to define extensions vocabulary. As a future work, we intend to evaluate the use of ontologies for formally defining the NFPs Framework, creating domain ontologies for different SPL applications.

- **Reuse Approach Tool.** A tool to support the NFPs Reuse Approach should be implemented to allow an automatic configuration of SPL products, where the feature model can be annotated with NFPs information. The CIDE tool, aforementioned, is a plugin to Eclipse IDE which extends the FeatureIDE tool (Leich *et al.*, 2005). The latter is also an Eclipse plugin for Feature-Oriented Software Development with a feature model graphic editor. The reuse approach tool should also be based on it, where the application engineer can associate features and products with NFPs, and thus facilitate the derivation process with customer's preferences.

- **Automatic NFPs selection.** The selection of NFPs values can be manually performed by setting the matching conditions, which form the framework Configuration Filter. As the number of NFPs values (Attribute Instances) for a specific product line grows, this search for instances in the A-base may become a complex task. Thus, an automatic NFPs selection could facilitate the engineers work. For example, a database could be used to store the Attribute Types and Attribute Instances facilitating the filter process.

- **Automatic NFPs measurements.** A way to allow a NFPs analysis process faster is to provide tools that could automatically measure NFPs values inside the Reuse Approach Tool. These values could be immediately attached to the documentation of Attribute Instances, the A-base. In addition, the validity conditions of each value could be automatically attached too.

8.3 Concluding Remarks

SPL in practice has been very successful in managing features that comprise functional properties, but a much smaller number of studies are related to non-functional ones. In addition, there are many NFPs that cannot be expressed and then realized in form of features, but require different approaches. How to deal with them is still not well established, neither in theory nor in practice.

Thus, the goal of this work was to further investigate the non-functional properties inside the software product lines area. In order to contribute and encourage discussion in this area, we proposed an approach to reuse NFPs values on SPL products. Once SPL engineering is focused on the reuse of SPL artifacts, NFPs values may also be reused.

Our findings from this work comes from the knowledge leveraged on the systematic literature review and from the NFPs Reuse Approach evaluations. Thus, they helped to conclude that additional investigations regarding the analysis of NFPs in the context of SPLs would be valuable for both practitioners and researchers. In addition, from evidence of the case studies, it also would be interesting to further investigate this phenomenon in industrial contexts.

References

Alves, V., Niu, N., Alves, C. F., and Valença, G. (2010). Requirements engineering for software product lines: A systematic literature review. *Information & Software Technology*, **52**(8), 806–820.

Aoyama, M. and Yoshino, A. (2008). Aore (aspect-oriented requirements engineering) methodology for automotive software product lines. In *Proceedings of the 2008 15th Asia-Pacific Software Engineering Conference*, APSEC '08, pages 203–210, Washington, DC, USA. IEEE Computer Society.

Apel, S. and Beyer, D. (2011). Feature cohesion in software product lines: an exploratory study. In *Software Engineering (ICSE), 2011 33rd International Conference on*, pages 421–430.

Asadi, M., Soltani, S., Gasevic, D., Hatala, M., and Bagheri, E. (2014). Toward automated feature model configuration with optimizing non-functional requirements. *Information and Software Technology*, **56**(9), 1144 – 1165. Special Sections from Asia-Pacific Software Engineering Conference (APSEC), 2012 and Software Product Line conference (SPLC), 2012.

Basili, V. R., Selby, R. W., and Hutchens, D. H. (1986). Experimentation in software engineering. *IEEE Trans. Softw. Eng.*, **12**(7), 733–743.

Basili, V. R., Caldiera, G., and Rombach, D. H. (1994). *The Goal Question Metric Approach*, volume I. John Wiley & Sons.

Bass, L., Clements, P., and Kazman, R. (2003). *Software Architecture in Practice*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2 edition.

Baumgart, S., Froberg, J., and Punnekkat, S. (2012). Towards efficient functional safety certification of construction machinery using a component-based approach. In *Product Line Approaches in Software Engineering (PLEASE), 2012 3rd International Workshop on*, pages 1–4.

Becker, S., Grunske, L., Mirandola, R., and Overhage, S. (2004). Performance Prediction of Component-Based Systems - A Survey from an Engineering Perspective. In *Architecting Systems with Trustworthy Components*, volume 3938 of *LNCS*, pages 169–192. Springer.

Benavides, D., Trinidad, P., and Ruiz-Cortés, A. (2005). Automated reasoning on feature models. In *Proceedings of the 17th International Conference on Advanced Information Systems Engineering*, CAiSE'05, pages 491–503, Berlin, Heidelberg. Springer-Verlag.

Benbasat, I., Goldstein, D. K., and Mead, M. (1987). The case research strategy in studies of information systems. *MIS Q.*, **11**(3), 369–386.

Bosch, J. (2000). *Design and Use of Software Architectures: Adopting and Evolving a Product-line Approach*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.

Chen, L. and Babar, M. A. (2011). A systematic review of evaluation of variability management approaches in software product lines. *Information & Software Technology*, **53**(4), 344–362.

Chung, L. and Prado Leite, J. C. (2009). Conceptual modeling: Foundations and applications. chapter On Non-Functional Requirements in Software Engineering, pages 363–379. Springer-Verlag, Berlin, Heidelberg.

Cicchetti, A., Ciccozzi, F., Leveque, T., and Sentilles, S. (2011). Evolution management of extra-functional properties in component-based embedded systems. In *Proceedings of the 14th International ACM Sigsoft Symposium on Component Based Software Engineering*, CBSE '11, pages 93–102, New York, NY, USA. ACM.

Clements, P. and Northrop, L. (2001). *Software Product Lines: Practices and Patterns*. Addison-Wesley.

Conradi, R. and Westfechtel, B. (1998). Version models for software configuration management. *ACM Comput. Surv.*, **30**(2), 232–282.

Couto, M., Valente, M., and Figueiredo, E. (2011). Extracting software product lines: A case study using conditional compilation. In *Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on*, pages 191–200.

Crnkovic, I., Larsson, M., and Preiss, O. (2005). Concerning predictability in dependable component-based systems: Classification of quality attributes. In *Architecting Dependable Systems III*. Springer-Verlag.

Czarnecki, K. and Eisenecker, U. W. (2000a). *Generative Programming: Methods, Tools, and Applications*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.

Czarnecki, K. and Eisenecker, U. W. (2000b). *Generative programming: methods, tools, and applications*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.

Czarnecki, K., Helsen, S., and Eisenecker, U. W. (2005). Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, **10**(1), 7–29.

Da Silva, I. F., Da Mota Silveira Neto, P. A., O'leary, P., De Almeida, E. S., and Meira, S. R. D. L. (2014). Software product line scoping and requirements engineering in a small and medium-sized enterprise: An industrial case study. *J. Syst. Softw.*, **88**, 189–206.

Deelstra, S., Sinnema, M., and Bosch, J. (2005). Product derivation in software product families: A case study. *Journal of Systems and Software*, **74**(2), 173–194.

Dijkstra, E. W. (1972). Structured programming. chapter Chapter I: Notes on Structured Programming, pages 1–82. Academic Press Ltd., London, UK, UK.

Etxeberria, L. and Sagardui, G. (2005). Product-line architecture: New issues for evaluation. In *SPLC*, pages 174–185. Springer-Verlag.

Etxeberria, L., Sagardui, G., and Belategi, L. (2008). Quality aware Software Product Line Engineering. *J. Braz. Comp. Soc.*, **14**(1), 57–69.

Fabbri, S., Hernandes, E., Thommazo, A., Belgamo, A., Zamboni, A., and Silva, C. (2012). Using information visualization and text mining to facilitate the conduction of systematic literature reviews. In *ICEIS*, pages 243–256.

Glinz, M. (2007). On non-functional requirements. In *RE*, pages 21–26. IEEE.

Greenfield, J. and Short, K. (2003). Software factories: assembling applications with patterns, models, frameworks and tools. In *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, OOPSLA '03, pages 16–27, New York, NY, USA. ACM.

Harman, M., Langdon, W. B., Jia, Y., White, D. R., Arcuri, A., and Clark, J. A. (2012). The GISMOE challenge: Constructing the pareto program surface using genetic programming to find better programs. In *ASE*.

Higgins, J. and Green, S., editors (2011). *Cochrane Handbook for Systematic Reviews of Interventions*. The Cochrane Collaboration. Version 5.1.0.

ISO/IEC 25000 (2011). Software product Quality Requirements and Evaluation (SQuaRE), Guide to SQuaRE.

ISO/IEC 9126 (2001). International standard iso/iec 9126, information technology - product quality - part1: Quality model.

Jha, M. and O'Brien, L. (2009). Identifying Issues and Concerns in Software Reuse in Software Product Lines. In *Proc. of the 11th International Conference on Software Reuse (ICSR)*, pages 181–190. Springer-Verlag.

Juristo, N. and Gómez, O. S. (2012). Empirical software engineering and verification. chapter Replication of Software Engineering Experiments, pages 60–88. Springer-Verlag, Berlin, Heidelberg.

Kang, K., Cohen, S., Hess, J., Nowak, W., and Peterson, S. (1990a). *Feature-Oriented Domain Analysis (FODA) Feasibility Study*.

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 CMU/SEI-90-TR-21, Pittsburgh, PA, USA.

Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., and Peterson, A. S. (1990c). Feature-oriented domain analysis (foda) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute.

Kästner, C. (2010). Virtual separation of concerns: Toward preprocessors 2.0. Logos Verlag Berlin, isbn 978-3-8325-2527-9.

Kauppinen, R. and Taina, J. (2003). Rita environment for testing framework-based software product lines. In P. Kilpeläinen and N. Päivinen, editors, *SPLST*, pages 58–69. University of Kuopio, Department of Computer Science.

Kitchenham, B. and Charters, S. (2007). Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report.

Lee, J., Kang, S., and Lee, D. (2012). A Survey on Software Product Line Testing. In *Proc. of the 16th International Software Product Line Conference (SPLC)*, pages 31–40. ACM.

Leich, T., Apel, S., Marnitz, L., and Saake, G. (2005). Tool support for feature-oriented software development: Featureide: an eclipse-based approach. In *Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology eXchange*, eclipse '05, pages 55–59, New York, NY, USA. ACM.

Linden, F. J. v. d., Schmid, K., and Rommes, E. (2007). *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Liu, Y., Ma, Z., and Shao, W. (2010). Integrating non-functional requirement modeling into model driven development method. In *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*, pages 98–107.

Lobato, L. L., Bittar, T. J., da Mota Silveira Neto, P. A., do Carmo Machado, I., de Almeida, E. S., and de Lemos Meira, S. R. (2013). Risk management in software product line engineering: a mapping study. *International Journal of Software Engineering and Knowledge Engineering*, **23**(4), 523–558.

Lohmann, D., Spinczyk, O., and SchrÃ¶der-preikschat, W. (2005). On the configuration of non-functional properties in operating system product lines. In *Proc. of the 4th AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software*.

Mairiza, D., Zowghi, D., and Nurmuliani, N. (2010). An investigation into the notion of non-functional requirements. In *SAC*, pages 311–317.

Mari, M. and Eila, N. (2003). The impact of maintainability on component-based software systems. In *EUROMICRO Conference*, pages 25–32. IEEE.

McCabe, T. (1976). A complexity measure. *Software Engineering, IEEE Transactions on*, **SE-2**(4), 308–320.

McGregor, J. D., Northrop, L. M., Jarrad, S., and Pohl, K. (2002). Guest editors' introduction: Initiating software product lines. *IEEE Software*, **19**(4), 24–27.

Montagud, S., Abrahão, S., and Insfrán, E. (2012). A systematic review of quality attributes and measures for software product lines. *Software Quality Journal*, **20**(3-4), 425–486.

Myllärniemi, V., Raatikainen, M., and Männistö, T. (2012). A Systematically Conducted Literature Review: Quality Attribute Variability in Software Product Lines. In *SPLC*, pages 41–45. ACM.

Neto, P. A. M. S., Machado, I. C., Mcgregor, J. D., Almeida, E. S., and Lemos Meira, S. R. (2011). A systematic mapping study of software product lines testing. *Information and Software Technology*, **53**(5), 407–423.

Nguyen, Q. L. (2009). Non-functional requirements analysis modeling for software product lines. In *Proceedings of the 2009 ICSE Workshop on Modeling in Software Engineering*, MISE '09, pages 56–61, Washington, DC, USA. IEEE Computer Society.

NHMRC (2000). *How to use the evidence: assessment and application of scientific evidence*. Australian National Health and Medical Research Council.

O'Leary, P., Rabiser, R., Richardson, I., and Thiel, S. (2009). Important issues and key activities in product derivation: Experiences from two independent research projects. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 121–130, Pittsburgh, PA, USA. Carnegie Mellon University.

Parnas, D. (1976). On the design and development of program families. *Software Engineering, IEEE Transactions on*, **SE-2**(1), 1–9.

Peng, X., Yu, Y., and Zhao, W. (2011). Analyzing evolution of variability in a software product line: From contexts and requirements to features. *Information & Software Technology*, **53**(7), 707–721.

Petticrew, M. and Roberts, H. (2006). *Systematic Reviews in the Social Sciences: A practical guide*. Oxford: Blackwell Publishing.

Pohl, K., Böckle, G., and Linden, F. J. v. d. (2005). *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag.

Rabiser, R., Grünbacher, P., and Dhungana, D. (2010). Requirements for product derivation support: Results from a systematic literature review and an expert survey. *Information and Software Technology*, **52**(3), 324–346.

Rabiser, R., O'Leary, P., and Richardson, I. (2011). Key activities for product derivation in software product lines. *Journal of Systems and Software*, **84**(2), 285–300.

Robertson, S. and Robertson, J. (1999). *Mastering the requirements process*. ACM Press Books. Addison-Wesley.

Robson, C. (2002). *Real World Research*. Blackwell, 2nd edition.

Rosa, N., Cunha, P. R. F., and Justo, G. R. R. (2002). Processnfl: a language for describing non-functional properties. In *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*, pages 3676–3685.

Runeson, P., Höst, M., Rainer, A., and Regnell, B. (2012). *Case Study Research in Software Engineering - Guidelines and Examples*. Wiley.

Sentilles, S. (2012). *Managing Extra-Functional Properties in Component-Based Development of Embedded Systems*. Ph.D. thesis, Mälardalen University, Västerås, Sweden.

Shull, F., Basili, V., Carver, J., Maldonado, J. C., Travassos, G. H., Mendonça, M., and Fabbri, S. (2002). Replicating software engineering experiments: Addressing the tacit knowledge problem. In *Proceedings of the 2002 International Symposium on Empirical Software Engineering*, ISESE '02, pages 7–, Washington, DC, USA. IEEE Computer Society.

Shull, F., Singer, J., and Sjoberg, D. I., editors (2008). *Guide to Advanced Empirical Software Engineering*. Springer.

Siegmund, N., Rosenmuller, M., Kuhlemann, M., Kastner, C., and Saake, G. (2008). Measuring non-functional properties in software product line for product derivation. In *Software Engineering Conference, 2008. APSEC '08. 15th Asia-Pacific*, pages 187–194.

Siegmund, N., Kuhlemann, M., Pukall, M., and Apel, S. (2010). Optimizing Non-functional Properties of Software Product Lines by means of Refactorings. In D. Benavides, D. S. Batory, and P. Grünbacher, editors, *Fourth International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS'10)*, volume 37 of *ICB-Research Report*, pages 115–122. Universität Duisburg-Essen.

Siegmund, N., Rosenmüller, M., Kuhlemann, M., Kästner, C., Apel, S., and Saake, G. (2012). SPL Conqueror: Toward optimization of non-functional properties in software product lines. *Software Quality Journal*, **20**(3-4), 487–517.

Sincero, J., Spinczyk, O., and Schröder-preikschat, W. (2007). On the Configuration of Non-Functional Properties in Software Product Lines. In *Software Product Lines*, pages 167–173.

Sincero, J., Schroder-Preikschat, W., and Spinczyk, O. (2009). Towards tool support for the configuration of non-functional properties in spls. In *System Sciences, 2009. HICSS '09. 42nd Hawaii International Conference on*, pages 1–7.

Sincero, J., Schroder-Preikschat, W., and Spinczyk, O. (2010). Approaching non-functional properties of software product lines: Learning from products. In *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*, pages 147–155.

Soares, L. R., Potena, P., Machado, I. C., Crnkovic, I., and Almeida, E. S. (2014). Analysis of non-functional properties in software product lines: a systematic review. In *IEEE 40th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*.

Stevens, S. S. (1946). On the Theory of Scales of Measurement. *Science*, **103**(2684), 677–680.

Thum, T., Kastner, C., Erdweg, S., and Siegmund, N. (2011). Abstract features in feature modeling. In *Software Product Line Conference (SPLC), 2011 15th International*, pages 191–200.

Vale, T., Cabral, B., Alvim, L., Soares, L. R., Santos, A., Machado, I., Souza, I., Silva, I. F., and Almeida, E. S. (2014). Splice: A lightweight spl development process for small and medium size projects. In *8th Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS)*.

Villela, K., Arif, T., and Zanardini, D. (2012). Towards product configuration taking into account quality concerns. In *Proceedings of the 16th International Software Product Line Conference - Volume 2*, SPLC '12, pages 82–90, New York, NY, USA. ACM.

Wagner, S. (2013). Software product quality control. pages I–XII, 1–210.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2000). *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Norwell, MA, USA.

Yin, R. K. (2009). *Case Study Research: Design and Methods*. SAGE Publications, 4th edition.

Zhang, H., Jarzabek, S., and Yang, B. (2003). Quality prediction and assessment for product lines. In *Proceedings of the 15th International Conference on Advanced Information Systems Engineering*, CAiSE'03, pages 681–695, Berlin, Heidelberg. Springer-Verlag.

Zhang, H., Babar, M. A., and Tell, P. (2011). Identifying relevant studies in software engineering. *Information and Software Technology*, **53**(6), 625–637.

Appendices

A

Systematic Literature Review - Primary Studies

This appendix lists the primary studies analyzed in the Systematic Literature Review of NFPs for SPL Engineering, earlier addressed in Chapter 3.

A.1 Primary Studies

(S1) D. Benavides et al., "Automated Reasoning on Feature Models,". in *CAiSE*, 2005.

(S2) M. Aoyama and A. Yoshino, "AORE (Aspect-Oriented Requir. Engin.) Methodology for Automotive Software Product Lines," in *APSEC*, 2008.

(S3) J. Bartholdt el al., "Integrating Quality Modeling with Feature Modeling in Software Product Lines," in *ICSEA*, 2009.

(S4) L. Etxeberria and G. Sagardui, "Variability Driven Quality Evaluation in Software Product Lines," in *SPLC*, 2008.

(S5) C. Ghezzi and A. M. Sharifloo, "Model-based verification of quantitative non-functional properties for software product lines," *Information and Software Technology*, 2013.

(S6) J. González-Huerta et al., "Non-functional Requirements in Model-driven Software Product Line Engineering" in *NFPinDSML*, 2012.

(S7) J. González-Huerta et al., "A Multimodel for Integrating Quality Assessment in Model-Driven Engineering," in *QUATIC*, 2012.

(S8) V. Guana and D. Correal, “Variability Quality Evaluation on Component-based Software Product Lines,” in *SPLC* 2011.

(S9) S.S. Kolesnikov et al., “Predicting Quality Attributes of Software Product Lines Using Software and Network Measures and Sampling,” in *VaMoS*, 2013.

(S10) T.M. Dao et al., “Problem Frames-Based Approach to Achieving Quality Attributes in Software Product Line Engineering,” in *SPLC*, 2011.

(S11) B. Mohabbati, “Development and Configuration of Service-oriented Systems Families,” in SAC 2011.

(S12) V. Nunes, “Variability Management of Reliability Models in Software Product Lines: An Expressiveness and Scalability Analysis,” in *SBCARS*, 2012.

(S13) Q.L. Nguyen, “Non-functional requirements analysis modeling for software product lines”,in *MISE*, 2009.

(S14) R. Olaechea et al., “Modelling and Multi-objective Optimization of Quality Attributes in Variability-rich Software,” in *NFPinDSML*, 2012.

(S15) F. Roos-Frantz et al., “Quality-aware analysis in product line engineering with the orthogonal variability model,” *Software Quality Journal*, 2012.

(S16) N. Siegmund et al., “Measuring Non-Functional Properties in Software Product Line for Product Derivation,” in *APSEC*, 2008.

(S17) N. Siegmund et al., “Optimizing Non-functional Properties of Software Product Lines by means of Refactorings,” in *VaMoS*, 2010.

(S18) N. Siegmund et al., “Predicting performance via automated feature-interaction detection,” in *ICSE*, 2012.

(S19) N. Siegmund et al., “SPL Conqueror: Toward optimization of non-functional properties in software product lines,” *Software Quality Journal*, 2012.

(S20) N. Siegmund et al., “Interoperability of non-functional requirements in complex systems,” in *SEES*, 2012.

(S21) N. Siegmund et al., “Scalable prediction of non-functional properties in software product lines: Footprint and memory consumption,” *Information and Software Technology*, 2013.

(S22) J. Sincero et al., “Approaching Non-functional Properties of Software Product Lines: Learning from Products,” in *APSEC*, 2010.

(S23) J. Sincere et al., “Towards Tool Support for the Configuration of Non-Functional Properties in SPLs,” in *HICSS*, 2009.

(S24) S. Soltani et al., “Automated Planning for Feature Model Configuration Based on Functional and Non-functional Requirements,” in *SPLC*, 2012.

(S25) R. Tawhid and D.C. Petriu, “Automatic Derivation of a Product Performance Model from a Software Product Line Model,” in *SPLC* 2011.

(S26) R. Tawhid and D. Petriu, “User-friendly Approach for Handling Performance Parameters During Predictive Software Performance Engineering,” in *ICPE*, 2012.

(S27) V. Karina et al., “Towards Product Configuration Taking into Account Quality Concerns,” *SPLC* 2012.

(S28) H. Zhang et al., “Quality Prediction and Assessment for Product Lines,” in *CAiSE*, 2003.

(S29) G. Zhang et al., “Quality attribute modeling and quality aware product configuration in software product lines,” *Software Quality Journal*, 2013.

(S30) V. Guana and D. Correal, “Improving software product line configuration: A quality attribute-driven approach,” *Information and Software Technology*, 2013.

(S31) E. Bagheri et al., “Configuring Software Product Line Feature Models Based on Stakeholders’ Soft and Hard Requirements”, *SPLC*, 2010.

(S32) J. Bosch and J. Lee, “Usage Context as Key Driver for Feature Selection,” in *SPLC*, 2010.

(S33) J. White et al., “Automating Product-Line Variant Selection for Mobile Devices,” *SPLC*, 2007.

(S34) J. White et al., “Selecting highly optimal architectural feature sets with Filtered Cartesian Flattening,” *Journal of Systems and Software*, 2009.

(S35) S. Jarzabek et al., “Addressing quality attributes in domain analysis for product lines,” *IEEE Proc.-Soft.*, 2006.

(S36) G. Jianmei et al., “A genetic algorithm for optimized feature selection with resource constraints in software product lines,” *Journal of Systems and Software*, 2011.

B

Case Study

This appendix presents more details about the case study addressed in Chapter 6. Section B.1 shows the forty features of Notepad SPL through its feature model. Section B.2 describes the product map of Notepad SPL, which lists the features of each product: Notepad Lite, Notepad Standard and Notepad Ultimate. Section B.3 presents the asked questions in the interview of this case study. And Section B.4 shows the details of three new products derived for the Notepad SPL.

B.1 Notepad SPL Feature Model

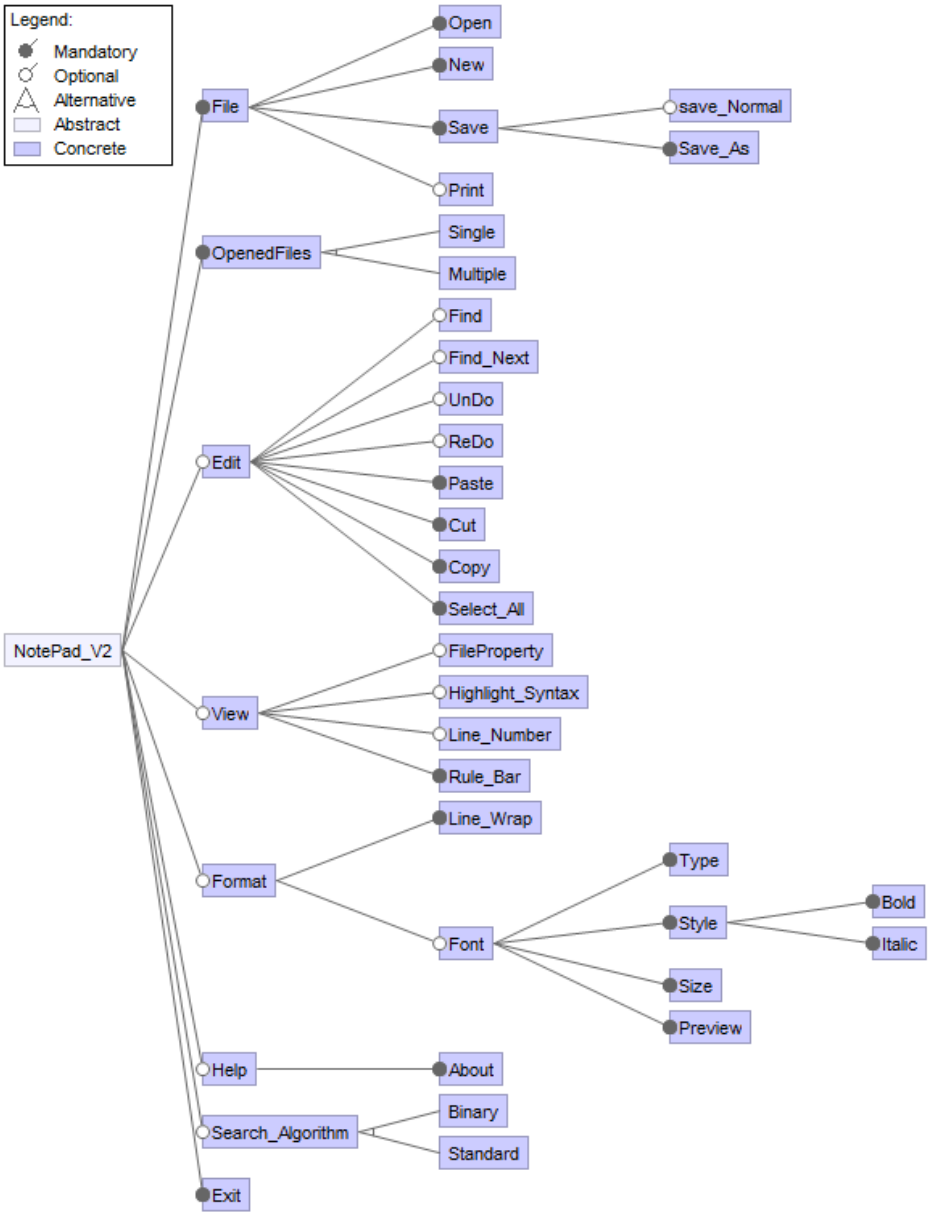

Figure B.1: Notepad SPL Feature Model

B.2 Notepad SPL Product Map

Features / Products			NotePad Lite	NotePad Standard	NotePad Ultimate
File			X	X	X
	Open		X	X	X
	New		X	X	X
	Save		X	X	X
		Save_As	X	X	X
		Save_Normal	-	X	X
	Print		-	X	X
Edit			-	X	X
	Find		-	X	X
	Find_Next		-	X	X
	UnDo		-	-	X
	ReDo		-	-	X
	Paste		-	X	X
	Cut		-	X	X
	Copy		-	X	X
	Select_All		-	X	X
View			-	X	X
	File_Property		-	X	X
	Highlight_Syntax		-	X	X
	Line_Number		-	X	X
	RuleBar		-	X	X
Format			X	X	X
	Line_Wrap		X	X	X
	Font		-	X	X
		Type	-	X	X
		Style	-	X	X
		Size	-	X	X
		Preview	-	X	X
Exit			X	X	X
Help			-	-	X
	About		-	-	X
Search_Algorithm			-	X	X
	Satandard		-	X	-
	Binary		-	-	X
OpenedFiles			X	X	X
	Single		X	X	-
	Multiple		-	-	X

Figure B.2: Notepad SPL Product Map

B.3 Case Study Interview

INITIAL PHASE

1. Motivation for using the NFPs Framework
2. Presentation of the NFPs Framework
   - Attribute Type
   - Attribute Instance
   - Attribute Metadata
3. Motivation for using the NFPs Reuse Approach
4. Presentation of the NFPs Reuse Approach
   - A-base
   - Configuration Filter
5. Interview proposal

INTERVIEW QUESTIONS

1. Are the Attribute Type and Attribute Instance definitions enough to (respectively) specify and analyse a measured/estimated/simulated value?
2. Are the additional tasks from the NFPs Reuse Approach clear?
3. Do you have difficulties to start using the reuse approach?
4. Do you think the reuse approach can avoid unnecessary (re)analysis of NFPs values? How?
5. Compared to a conventional product derivation process, what is the effort spent to specify analysed NFPs values in the way of Attribute instances?
6. What are the benefits and drawbacks of the reuse approach?

FINAL

Thanks for your attention.

B.4 Notepad SPL Product Map for three new products

Features / Products	NotePad Lite+2	NotePad Standard+1	NotePad Ultimate-1
File	X	X	X
Open	X	X	X
New	X	X	X
Save	X	X	X
Save_As	X	X	X
Save_Normal	X	X	X
Print	X	X	X
Edit	X	X	X
Find	-	X	X
Find_Next	-	X	X
UnDo	-	X	X
ReDo	-	X	X
Paste	X	X	X
Cut	X	X	X
Copy	X	X	X
Select_All	X	X	X
View	-	X	X
File_Property	-	X	X
Highlight_Syntax	-	X	X
Line_Number	-	X	X
RuleBar	-	X	X
Format	X	X	X
Line_Wrap	X	X	X
Font	X	X	X
Type	X	X	X
Style	X	X	X
Size	X	X	X
Preview	X	X	X
Exit	X	X	X
Help	-	X	X
About	-	X	X
Search_Algorithm	-	X	X
Satandard	-	X	-
Binary	-	-	X
OpenedFiles	X	X	X
Single	X	X	X
Multiple	-	-	-

Figure B.3: ProductMap for 3 new products

C

Replicated Case Study

This appendix presents some details about the exploratory replicated case study addressed in Chapter 7. Section C.1 shows the features of ResecueMe SPL through its feature model. Section C.2 describes the product map of ResecueMe SPL, which lists the features of each product.

C.1 RescueMe SPL Feature Model

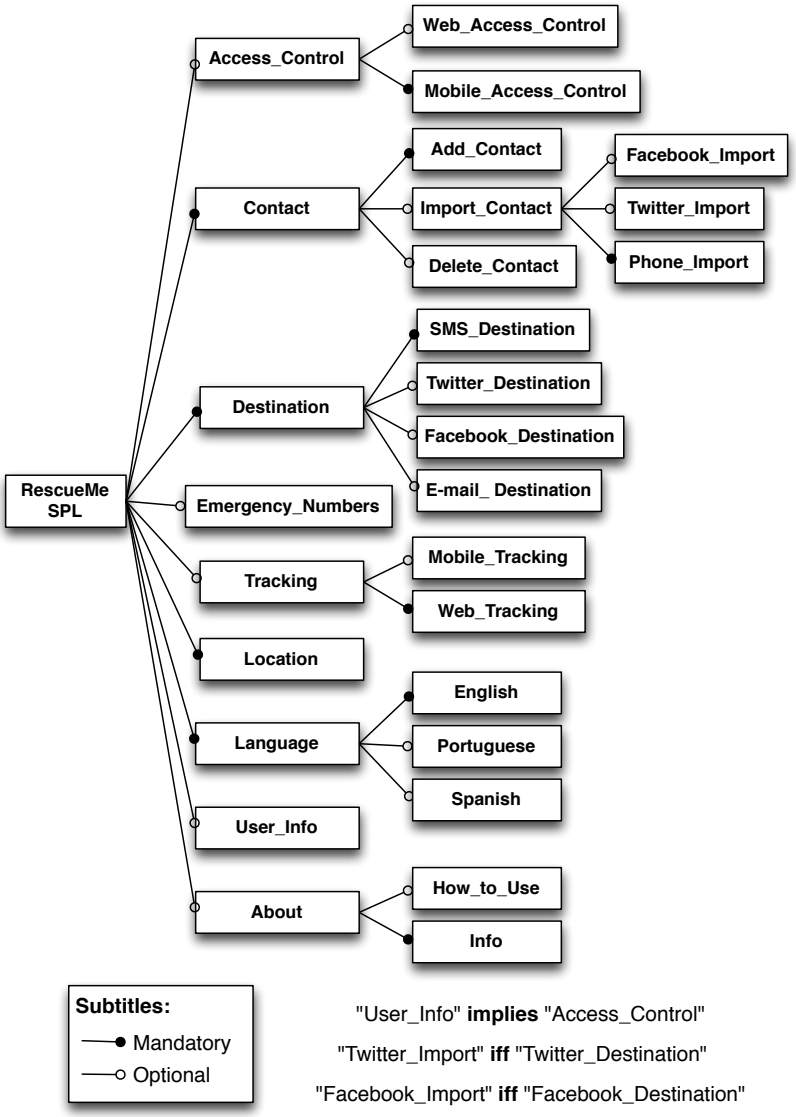

Figure C.1: Part of the RescueMe Feature Model (Vale *et al.*, 2014)

C.2 RescueMe SPL Product Map

Table C.1: A fragment of the RescueMeSPL Product Map.

#	RescueMe Features	Lite	Standard	Social	Pro	Ultimate
1	Destination	✔	✔	✔	✔	✔
2	SMS_Destination	✔	✔	✔	✔	✔
3	Twitter_Destination			✔	✔	✔
4	Facebook_Destination			✔	✔	✔
5	Email_Destination		✔	✔	✔	✔
6	Access_Control			✔	✔	✔
7	Web_Access_Control			✔	✔	✔
8	Mobile_Access_Control			✔	✔	✔
9	Contact	✔	✔	✔	✔	✔
10	Add_Contact	✔	✔	✔	✔	✔
11	Import_Contact		✔	✔	✔	✔
12	Phone_Import		✔	✔	✔	✔
13	Twitter_Import			✔	✔	✔
14	Facebook_Import			✔	✔	✔
15	Delete_Contact	✔	✔	✔	✔	✔
16	Emergency_Numbers		✔	✔	✔	✔
17	Tracking				✔	✔
18	Mobile_Tracking					✔
19	Web_Tracking				✔	✔
20	Location	✔	✔	✔	✔	✔
21	Language	✔	✔	✔	✔	✔
22	English_Language	✔	✔	✔	✔	✔
23	Portuguese_Language					✔
24	Spanish_Language					✔
25	About	✔	✔	✔	✔	✔
26	Info	✔	✔	✔	✔	✔
27	How_to_Use		✔	✔	✔	✔
28	User_Info		✔	✔	✔	✔

✔: Selected feature.