Non-Functional Properties in Software Product Lines: A Reuse Approach

Larissa Rocha Soares[*]
Federal University of Bahia
Salvador, Bahia, Brazil
larissars@dcc.ufba.br

Ivan do Carmo Machado
Federal University of Bahia
Salvador, Bahia, Brazil
ivanmachado@dcc.ufba.br

Eduardo Santana de Almeida[†]
Federal University of Bahia
Salvador, Bahia, Brazil
esa@dcc.ufba.br

ABSTRACT

Software Product Line Engineering (SPLE) emerges for software organizations interested in customized products at reasonable costs. Based on the selection of features, stakeholders can derive programs satisfying a range of functional properties and non-functional ones. The explicit definition of Non-Functional Properties (NFP) during software configuration has been considered a challenging task. Dealing with them is not well established yet, neither in theory nor in practice. In this sense, we present a framework to specify NFP for SPLE and we also propose a reuse approach that promotes the reuse of NFP values during the product configuration. We discuss the results of a case study aimed to evaluate the applicability of the proposed work.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques; D.2.8 [**Software Engineering**]: Metrics; D.2.13 [**Software Engineering**]: Reusable Software

General Terms

Design, Theory, Measurement

Keywords

Software Product Line, Quality Attributes, Empirical Software Engineering

1. INTRODUCTION

SPLE exploits the commonalities among products to achieve economies of scale by developing core assets, entities that will be reused in multiple product instances, such as requirements, architectures, components, test cases and so on [4].

*Corresponding author.

†Fraunhofer Project Center for Software and Systems Engineering, Brazil

VaMos 2015 Hildesheim, Germany
Copyright 2015 ACM 978-1-4503-3273-6/15/01 ...$15.00.
http://dx.doi.org/10.1145/2701319.2701326

SPLE allows creating a number of related products aimed at satisfying market demands. For the expected economic benefits, SPLE has gained an increasing attention by software companies, as these are looking for strategies to handle an even increasing market demand for better products, shorter time-to-market, and decreased production costs.

Product-line variants are distinguished in terms of features, i.e., end-user visible characteristics of products [5]. Based on features selection, stakeholders can derive tailor-made products satisfying a range of functional properties (FP) and non-functional properties (NFP).

FP implement the tasks/functionalities of a software. On the other hand, NFP also play an important role in SPLE, and are those that impose special conditions and qualities on the system [6]. For instance, if a software system runs slower than expected, users may not be interested in using it, regardless its provided functionalities. FP and NFP should work synchronized for a product to be viable in the market.

As opposed to single-system software development, an SPL typically covers a wide variety of application scenarios, environments and customer demands. Hence, an SPL may hold several different products with many types of NFP. The explicit definition of NFP during software configuration has been considered as a challenging task [10, 11]. This definition may vary from author to author, with different ways of NFP analysis and measurement process, which hinders the understanding of their meaning or even to reuse them in other contexts.

This work aims at investigating NFP in SPLE. We present a framework to specify NFP and propose an approach to reuse NFP values on SPL products, named NFP-RA. Once SPLE promotes the reuse of SPL artifacts, we believe that NFP values may be reused as well. Besides, we report on a case study to evaluate the applicability of both framework and reuse approach in a text editors SPL.

The framework describes the characteristics of an NFP, and systematically provide additional information about features and products inside the SPL process. The framework supports the NFP-RA, where time consuming tasks related to the measurement of NFP values do not have to be repeated for very similar systems. The NFP-RA presents how to configure product-line variants aware of NFP, and also shows how to reuse NFP values previously analysed, avoiding unnecessary reanalysis.

The remainder of this paper is structured as follows: Section 2 shows the related work; Section 3 presents the Framework; Section 4 details the NFP-RA; approaches evaluation is presented in Section 5; and Section 6 draws concluding remarks.

2. RELATED WORK

The literature presents a number of studies dealing with NFP in SPL. Some of them stand out. The work in [8] focused their investigation on the problem of modeling variability in quality concerns for SPL. Specifically, the primary studies at the Software Product Line Conference (SPLC) have been reviewed in order to understand why, how, and which quality attributes should vary among product instances.

The systematic review presented in [12] investigated how researchers and practitioners have been looking for improvements in the inclusion process of runtime NFP in the SPL approach. The authors categorized the primary papers in predictions, estimations and features selection-focused approaches. A different review was presented in [7], which proposes a catalog of measures for quality attributes found in the SPL lifecycle. The study found 165 measures related to 97 different quality attributes.

There are some other studies handling specification and reuse of NFP information. A framework for component-based embedded systems was proposed in [9], where NFP can be attached to architectural entities of component models, such as ports, interfaces and connectors. Furthermore, concrete NFP values can be compared and specified during the development process.

That work aimed to provide an efficient support for analyzing selected properties. Our proposal used the same framework described in [9], but with slight changes to the SPL context, in order to provide a way of properly specifying and documenting NFP for the SPL development, specially on the product derivation process.

Some techniques to reuse NFP values in a new context are discussed in [2, 3]. Authors in [3] introduce a novel mechanism focused on evolution management of NFP for component-based embedded systems. Their goal was to discover whether an NFP value is still valid when components related to it evolve. Conversely, the work in [2] proposed a way to reuse functional software certification for the automotive domain, since the safety certification process of vehicles critical functionalities takes too much time. We elaborate on these work in order to discuss an NFP reuse approach for SPLE aimed to support the reuse of NFP, and avoid time consuming tasks regarding measurements of NFP values.

3. AN NFP FRAMEWORK FOR SPL

In a preceding investigation, we observed that the management of NFP is still a key challenge for SPLE [12]. Similar observations were also made in [10, 11]. The specifications of NFP regarding features/products is neither defined nor uniform. Most publications in the literature handles NFP in a different and not systematic way [12]. Along this Section, we present the NFP Framework that aims at a clearer integration process of NFP with features and products. The Framework supports the reuse approach, NFP-RA, later discussed in this paper.

As we earlier discussed in the paper, our proposal elaborates on the framework proposed in [9], and extend it to incorporate SPLE particularities, such as the notion of features and variability. Such a tailoring task included an arrangement of activities spread over five key tasks. The main goal is to support the product derivation process with additional information about features and products. These tasks are discussed next.

3.1 NFP Framework Key Tasks

Task 1: **Definition of an Attribute.** This is the starting point of the approach. The purpose is to present the definition of a Quality Attribute (hereinafter, Attribute).

Task 2: **Definition of Attribute Type.** An Attribute is composed of two main parts: Attribute Type and Attribute Instance. This task aims to show the specifications of Attribute Types, besides presenting their repository, Attribute Registry.

Task 3: **Definition of Attribute Instance.** This task presents the other part of an Attribute, the Attribute Instance, that represents the concrete value of an Attribute.

Task 4: **Definition of Attribute Value Metadata.** The context of Attribute Instances helps understanding how and under which conditions a given instance was analyzed. Attribute Value Metadata represents this context, and their repository is the Metadata Registry.

Task 5: **Definition of Value Selection.** As an SPL can have many different Attributes, and a same Attribute can assume several values, this task shows how to select only the Attribute Instances of interest for a given product.

Task 1. The concept of attribute is defined as: `Attribute = (Type, Value*)` [9]. An NFP, also called *Attribute*, is based on two definitions: (i) the *Attribute Type*, that define a type of NFP, and (ii) the attribute instance, *Attribute Value*, that refers to a given NFP value associated with a feature or product of an SPL-based design.

An NFP is represented through **only one** *Attribute Type* and **at least one** *Attribute Value*. Therefore, an attribute can have more than one value, where the difference can be on different metadata and/or validity conditions. For example, an NFP defined as *CPU Consumption* can have an 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 other for a measured value.

Task 2. This task is responsible to define two primary elements from NFP Framework: *Attribute Type* and *Registry*. The *Attribute Type* specifies all the commonalities shared by the instances of a given attribute. It is defined as `Attribute Type = (TypeID, Description, Attributable*, Variability, DataFormat, Documentation)` [9].

TypeID is a unique identifier for the *Attribute Type*. The name of the property is used as its unique identifier. *Description* offers the meaning of the corresponding *Attribute Type*. *Attributable* represents the element kinds to which an NFP of type *TypeID* can be attached to. Attributable elements are features and products. But, in the case of features,

Table 1: Attribute Registry with four Attribute Types

#	TypeID	Description	Attributable	Variability	DataFormat	Documentation
1	*Footprint*	Binary size of compiled files	Product	Optional	Quantitative, Kilobytes (KB)	foot.pdf
2	*Performance*	Performance specified by experts	Feature	Optional	Qualitative, Low/Medium/High	perf.doc
3	*Memory Consumption*	Consumed memory during the product execution	Product	Optional	Quantitative, Megabytes (MB)	mem.doc
4	*Usability*	Ease of use of features	Feature	Optional	Qualitative, Low/Medium/High	usa.pdf

only leaf features are important to attach NFP information[1]. Regarding the *Variability*, a feature can be part of a system or not, i.e., it can be either mandatory or optional. Similarly, some NFP are mandatory (common across the SPL) whereas others are optional. *DataFormat* corresponds to the data type used to represent the values of an *Attribute Type*. *Documentation* describes the NFP in natural language.

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 ensure the uniqueness of each one. Thus, we may keep a *registry* of *Attribute Types* [9]. Table 1 presents an *Attribute Registry* for a text editors SPL, named Notepad SPL, with four *Attribute Types*: Footprint, performance, memory consumption and usability.

Task 3. The instance of an attribute represents its concrete value, so that *Attribute Value* and *Attribute Instance* hold similar meaning. Thus, an instance is defined as `Instance = (TypeID, AttributableName, Data, Metadata, ValidityConditions)`. A specific instance of an attribute must have a *TypeID*, which is an identification for its type; *Data*, concrete value for the NFP; *AttributableName*, name of the feature or product that the instance is related to; *Metadata*, complementary information on data that allows to distinguish among them; and the *ValidityConditions*, conditions under which the value is valid. These conditions correspond to a set of restrictions of the applicability context of *Attribute Instances*. For instance, feature interaction and platform are examples of *ValidityConditions*. Table 2 shows two instances for different *Attribute Type* of Table 1.

Task 4. 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 a metadata that follows the specifications defined by the given *Metadata Type*.

A *Metadata Type* (MetaType) is defined as `MetaType = (MetaID, ValueFormat, Variability, Description)`, where *MetaID* is a unique identifier for the *Metadata Type*; *ValueFormat* specifies types used to represent the values; the *Variability* specifies whether the *Metadata Type* is either mandatory or optional for an Attribute Value; and *Description* corresponds to a simple description of the *Metadata Type*.

For example, the *Metadata Type*, which ID is "Source" can be defined as *Source* = {"Measurement" or "Estimation" or "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 2 shows, for example, for the *Attribute Type* "CPU Consumption", two *Attribute instances*, where one has *Source* = "Estimation" and the other has *Source* = "Measurement". Table 3 shows the *Metadata Registry* with examples of *Metadata Types* and its specifications.

Task 5. As a same attribute can have many different values, it is possible that a large amount of attribute instances can be associated to a given SPL. This way, Attribute instances can be obtained through the use of a *Configuration Filter* [9]. It provides a better control over the values to retrieve by using one or more conditions. In the values selection point of view, *Metadata* and *ValidityConditions* are equivalent. The Configuration Filter defines constraints over them in the same way. An example of *Configuration Filter* can be expressed as follows (where ELSE conditions are neatly tested until a subset of attribute instances is selected):

(Source = Estimation) **AND** (Platform Operation System = Ubuntu 12.10) **ELSE**
(TypeID = "Memory Usage")

Table 3: Metadata Registry

MetaID	ValueFormat	Variability	Description
CreationTime	TimeStamp	Mandatory	Creation data and time of the Attribute Value
Source	{"Estimation", "Measurement", "Simulation"}	Mandatory	The way a value is analyzed
SourceTool	String	Optional	Tool responsible to analyze the Attribute Value
Complexity	{"Low", "Medium", "High"}	Mandatory	Complexity to get an Attribute Value

4. THE REUSE APPROACH FOR NFP

The systematic reuse of assets is at the core of an SPLE approach. Since features can be reused among many different SPL products, the analysis of NFP could also be reused among them, as a way to avoid unnecessary re-analysis. If two products are very similar according to their selected set of features and application context, it is possible that they

[1]Leaf features are at the end of feature models and cannot be abstract features. For reasoning about NFP in SPL, abstract features are not relevant [13]

Table 2: Attribute Instances

TypeID	#	Attributable Name	Data	Metadata	Validity Conditions
Footprint	1	Product: Notepad Lite	116kB	CreationTime = “10.06.14” Source = “Measurement” Complexity = “Low”	Feature Interactions: {F1, F5}
	2	Product: Notepad Standard	232kB	CreationTime = “11.06.14” Source = “Measurement” Complexity =“Low”	Feature Interactions: {F5, F12, F17}
Memory Consumption	3	Product: Notepad Lite	23.1MB	CreationTime = “12.06.14” Source = “Measurement” Source Tool = “Windows Task Manager” Complexity =“Low”	Platform: SO: Windows 8.1, CPU: Intel Core i5, RAM: 6GB.
	4	Product: Notepad Standard	27.3MB	CreationTime = “13.06.14” Source = “Measurement” Source Tool = “Windows Task Manager” Complexity =“Low”	Platform: SO: Windows 8.1, CPU: Intel Core i5, RAM: 6GB.

have very similar NFP values too. Thus, NFP values may not need to be calculated again, for the next product instances.

In order to promote the reusability of NFP values, we propose the *NFP-RA*, an NFP Reuse Approach for SPL, aimed to reuse previous NFP analysis in order to minimize the effort of performing a new analysis. This approach integrates the *NFP Framework* inside the SPL life-cycle, providing a systematic reuse of NFP values during the product derivation process.

The proposed approach supports a product derivation NFP-aware process, where stakeholders consider requirements over FP and NFP. Engineers and customers can formulate requirements in terms of constraints and preferences and make configuration decisions to select proper features based on the feature model and also in a repository (A-base) of NFP values (instances). These NFP values provide engineers with a means to generate the best product aware of the properties of that particular product, such as their strengths and weaknesses.

The NFP-RA is explained through four main steps, as listed next, and further detailed.

- **Step 1:** populate the A-base with the attribute instances;
- **Step 2:** configure the NFP Filter in order to search for values in the A-base;
- **Step 3:** analyze the filtered values;
- **Step 4:** create new instances if there are no appropriate values for the derived product.

4.1 Step 1: Populating the A-base

Figure 1 shows the proposed approach, by considering the two main SPL processes: 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. We consider that NFP can be identified in two moments: (i) for the entire SPL during domain engineering, where NFP are common for all products; and (ii) per product, during the application engineering, where new NFP can be required for a derived product.

During the domain phase, we propose the creation (and population) of the registries defined in the *NFP Framework*: Attribute Registry (Task 2) and Metadata Registry (Task 4). Even in this phase, when products had not been assembled yet, the **Attribute Values Base (A-base)** can be created. It is possible to analyze the set of NFP through estimations by, for example, qualitative measures. But, if there are experts with a deep knowledge of analyzing software quality aspects, they can estimate NFP values with percentage or more accurate quantitative means.

Population activities of registries and A-base can be visited during any moment of the SPL life-cycle. Not only during the domain engineering, but also during the application engineering, insertions or modifications can be made.

4.2 Step 2: Configuring the NFP Filter

Through the **FM** (feature model) a new product starts to be configured by selecting the features that better fits into the target requirements, represented by the **Feature Conf.** (feature configuration) activity, as Figure 1 shows. 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 NFP-RA, the **Conf. Filter** (configuration filter), needs to be defined to obtain the NFP values. The filter should reflect customer demands on the specified product. The main idea, as detailed on Task 5, is to search for NFP 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.

4.3 Step 3: Reuse Diagnosis activity

Although the filter defined by the user (developer, engineer, etc.) selects only the values that conforms to the conditions, many values can be selected. The user must analyze and decide if they are applicable to the product or not. For this decision activity we named **Reuse Diagnosis**, and it has three different cases:

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.

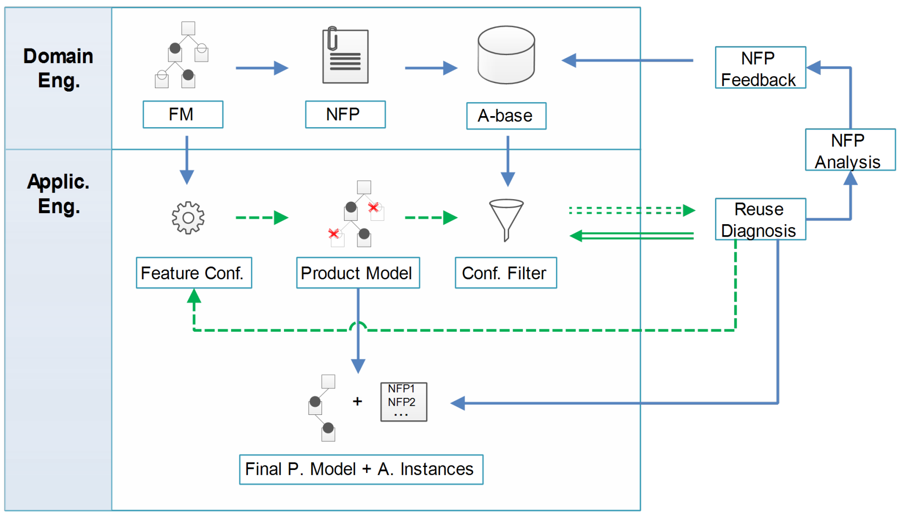


Figure 1: Approach to derive a product aware of NFP

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. Corresponding to the features means that all the necessary conditions (*Metadata* and *Validity Conditions*) for using that filtered NFP values were accepted by users for the derived product. In this case, the reuse of NFP analysis was performed. Thus, the **Final P. Model** (model with the features chosen to the current product) is generated and can be annotated with attribute values from the **Conf. Filter**, the **A. Instances** (Attribute Instances).

Case 2 may occur 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 to change the set of features to obtain that desired good values of NFP. 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. This is a cyclic process and is represented by the dashed arrows of Figure 1.

In Case 3, the filtered values are not applicable to the product at all, i.e., it is not possible to comply with the conditions requested by the *Metadata* and *Validity Conditions* of the filtered instances. Thus, it is not possible to reuse the values filtered from the A-base. The user has two options:

- Option 1: change the filter. The user can search either for *Metadata* or *Validity Conditions*, or directly specify on the filter the NFP of interest. The two double-line arrows on Figure 1 represents the user can change the filter as much as necessary.

- Option 2: feed the A-base with new NFP values. If the A-base does not have the NFP values searched by the user, he needs to measure the NFP for that product, and add these new information (*Attribute Instances*) inside the A-base, making them available to others.

4.4 Step 4: Creation of new Attribute Instances

The main goal of the *NFP-RA* is to avoid unnecessary analysis of NFP values during the configuration of a new product, making the derivation process faster. Since it maintains a repository (*A-base*) with NFP values that have already been analyzed (estimated, measured, simulated, etc.) for other products of the same SPL, it is possible that if the user wants to generate a new product similar to one already derived before, adequate NFP values can be found for the new product, avoiding to calculate them again.

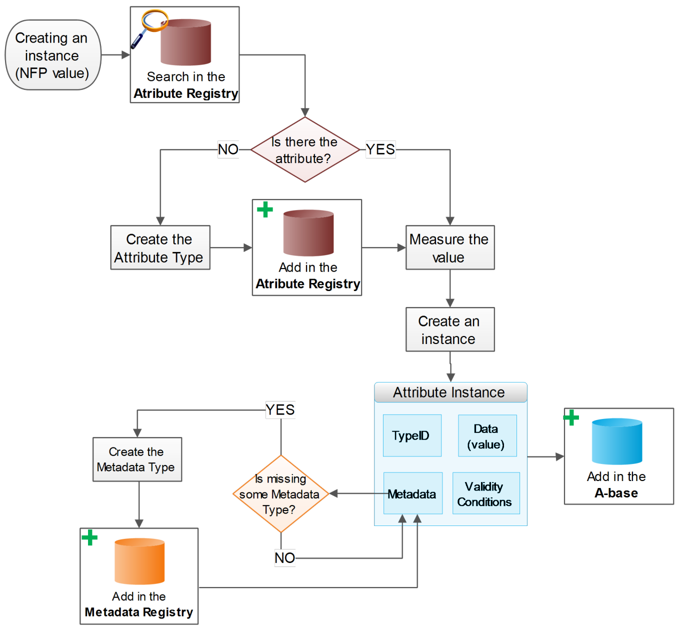
However, the reuse of NFP values (or attribute instances) cannot always be done. For example, when: either (i) the SPL is really new, with a few or no product derived; or (ii) even with an SPL that has many products, no value may be appropriated for the new product to be generated. In such cases, the NFP of interest has to be analyzed for the first time. The reuse approach activity from Figure 1, named **NFP Analysis**, represents this NFP value analysis process for a target product.

After analyzing the required NFP, the user must feed the **A-base** with the newer values. This activity is named **NFP Feedback**. To do so, some points should be highlighted according to the *NFP Framework*. For example, it is necessary to verify if the NFP to be measured has already been specified as an *Attribute Type* in the *Attribute Registry*. In case the user did not find it, he has to create a new one, and after that, an *Attribute Instance* can be created. Besides the value, an instance has also the *TypeID*, *Metadata* and *Validity Conditions*. The *Metadata Registry* also needs to be checked. If the metadata presents on the *Metadata Registry* are not enough to describe the conditions under which the value is analyzed, a new category of metadata can be added to the registry. Figure 2 shows this process of creating a new *Attribute Instance*.

The **A-base** is a repository of *Attribute Instances*, and whenever a new instance is created it must be added to the base. Therefore, this new instance will be available to other users during the derivation of other products instances. This process is of great importance to ensure the efficiency and usefulness of the reuse approach.

5. EVALUATION

During product derivation, when the features from a core set are selected to assemble a given product instance, both



Creating an instance (NFP value)
Search in the Atribute Registry
Is there the attribute?
NO
YES
Create the Attribute Type
Add in the Atribute Registry
Measure the value
Create an instance
Attribute Instance
TypeID
Data (value)
Metadata
Validity Conditions
Add in the A-base
YES
Create the Metadata Type
Is missing some Metadata Type?
NO
Add in the Metadata Registry

Figure 2: Workflow of Step 4

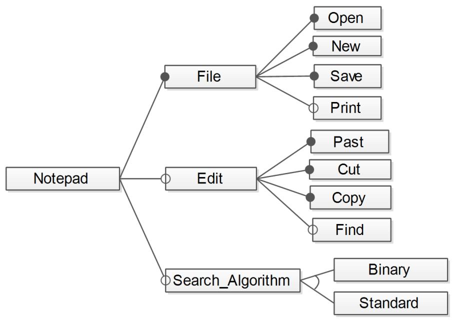


Figure 3: Part of Notepad SPL

FP and NFP should be taken into account. A repository containing NFP previously analyzed, e.g., for already delivered product instances, should provide product derivation task with actual and reliable information to aid reusing some property values. It may lead to effort reduction, as it will not be necessary to analyze known properties again.

Hence, instead of performing analysis/measurements for each new product instance, there is an opportunity to simply reuse such knowledge from similar products. To accomplish such a goal, we present an NFP Framework and defined a reuse approach, the NFP-RA. In order to investigate both approach and framework applicability, we performed an exploratory case study in a product line of text editors, as discuss next.

5.1 Case Study

The empirical evaluation aims at analyzing the NFP Framework and NFP-RA from the point of view of domain and application engineers working in SPLE.

The SPL used in this study is named Notepad SPL, a small-sized product line in the domain of text editors, implemented by the RiSE Labs group[2]. 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. The study in [1] earlier analyzed the same programs to explore feature cohesion characteristics[3].

The Notepad SPL consists of a desktop application implemented in JAVA. It contains 1,803 lines of code and 14 classes. The project has 40 features and 3 main product instances: Notepad Lite, Notepad Standard and Notepad Ultimate, with 10, 31 and 35 features respectively. Figure 3 shows part of the Notepad SPL feature model.

In order to observe the applicability of the NFP-RA within the Notepad SPL, two SPL engineers acted as subjects in this study. Their role included extracting NFP of importance to the Notepad products and features. The SPL had already been completely implemented by the time of this study. Thus, we considered three main activities to be carry out by the engineers: initial phase, observation, and interview. These activities are discussed next.

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

Activity 1 - Initial Phase. A 30-minute-explanation was performed to the SPL engineers about the NFP Framework and NFP-RA responsible to present its motivations and details.

Activity 2 - Observation. SPL engineers from the Notepad development team were observed when using the NFP-RA to derive SPL products. They were observed during about one hour. The engineers followed a workflow with five points. They were asked to: (i) choose some NFP for the Notepad SPL, and specify them as *Attribute Types*; (ii) measure those NFP for the three available SPL products, Notepad Lite, Standard and Ultimate; (iii) specify and document the measured NFP values in the way of *Attribute Instances*; (iv) derive new products with specific NFP needs. According to the NFP-RA, they derived a product and created a *Configuration filter* with the specific needs chosen (NFP-RA Step 2 as earlier detailed in Section 4.2); and (v) analyze the instances retrieved from *A-base* according to the filter that has been drawn. This process of analysis is referred to as “Reuse Diagnosis activity” and it is composed of three cases (NFP-RA Step 3).

Activity 3 - Interview. After completing the previous workflow, we conducted an interview aiming to answer a set of research questions, as follows:

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

Q2: Does the NFP-RA avoid unnecessary (re)analysis of NFP values?

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

Q4: What are the drawbacks and benefits of the NFP Framework and NFP-RA?

5.2 Execution

The Notepad SPL was developed by two 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 one was the application engineer. The domain engineer was the first author of this work. The observation activity was shared in two parts, one for each engineer.

The first part was performed by the domain engineer, and included only the three first points of the observation workflow (shared in five points presented on previous section, Activity 2). As the Notepad SPL has already been implemented with three different products and there were no NFP documented, the domain engineer could start performing the Step 1 of the NFP-RA, *Populating the A-base*, creating the *Attribute Registry, Metadata Registry* and *A-base.*

The domain engineer chose four NFP, `Footprint`, `Memory Consumption`, `Performance`, and `Usability`), specified them as *Attribute Types* (workflow point 1), analyzed through estimations (workflow point 2), and created instances (workflow point 3). Table 1 showed on Section 3, represents the *Attribute Registry* with the specified NFP.

For the second part of the observation, assumed by the application engineer, he had particular differences. He did not know the guidelines of the NFP-RA in a deep way as the domain engineer did, besides he did not work on domain engineering phase, but on application engineering phase, which is responsible for the product derivation process.

This second part included the five points of the observation workflow. The application engineer chose two NFP from Table 1, `Footprint` and `Memory Consumption` (workflow point 1), measured them for the three Notepad products (workflow point 2), and created the instances (workflow point 3). Some of these instances are showed on Table 2.

After that, the application engineer derived three new products, each one with its corresponding *Configuration Filter* (workflow point 4). For example, for the new product `Notepad Lite+2`, the *Configuration Filter* was designed as `(Source = Measurement) AND (TypeID = Footprint)`. In this case, the engineer wants to know more about the values of *footprint* that had already been *measured* for this SPL.

From the three filters, he analyzed the instances retrieved from each one (workflow point 5). Since, the engineer is an expert on the Notepad SPL, he classified the filtered values as useful for one case, where no other measurement needed to be computed, and for the others cases either the product was modified to attend NFP restrictions or new NFP were measured and added in the *A-base.*

5.3 Results and Findings

After the observation activities, we performed a 20-minute interview, in order to contribute with the approach evaluation and answer the research questions. The interview was applied to the application engineer, as he was responsible for performing the product derivation process.

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

The application engineer characterized the NFP Framework as simple and of easy understanding. However, despite 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" steps. The activity required to start using the reuse approach is to get a good understanding of the NFP Framework.

Q2: Does the NFP-RA avoid unnecessary reanalysis of NFP values?

The reuse of NFP analysis, as stated by the application engineer, can be avoided in cases where estimated values are enough. Whether only exact values are required, an estimated value is not enough. As a consequence, the engineer will have to measure it. However, sometimes it is possible to analyze upper and lower bounds regarding the product under analysis, where these values are so close that the estimation can be further exploited.

Not every SPL needs to be completely NFP-aware. Applications addressed to devices with limitations in memory and CPU consumption, for instance, take more advantage of the NFP-RA than systems addressed to the desktop domain. For devices, such as tablets and smartphones, individual applications must not consume a huge amount of resources, otherwise they can slow down and not work properly.

The A-base also serves as a derivation guide. The reuse approach is not only focused on NFP reusability, but it 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.

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

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

Q4: What are the drawbacks and benefits of NFP Framework and the NFP-RA?

According to the engineer, the simplicity of the fields necessary to create a new *Attribute Type*, such as *TypeID*, *Attributables*, *Variability* and *Description*, is a positive point. The same occurs to *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 because of the conditions which needed more attention. However, during the observation activity, despite the three filters had been perfectly described, they were not used as they should. 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 electronic spreadsheets.

The NFP-RA is as simple as the framework. Through the approach is possible to guide a product derivation aware of NFP. The advantage is not only to reuse NFP values, but to know more about the SPL in terms of quality. In spite of the effort of specifying types and instances of attributes to be small, the application of the NFP-RA for large scale SPL may take a longer time.

5.4 Threats to Validity

Construct Validity. As the main researcher of this study also developed the SPL Notepad, she had a strong influence on the conclusions. To mitigate this threat, another participant played the role of an application engineer, who was responsible for the main tasks of the product derivation process aware of NFP.

Internal Validity. Both NFP Framework and NFP-RA have a couple of steps and tasks. It is possible that some concepts may have been misinterpreted. To mitigate this threat, the researcher was, during the observation activity, all the time close to the SPL engineer.

The text editor SPL used in this study may not be the most appropriate. It was not an example of a SPL where the analysis of NFP were strongly recommended, since it is an academic small project addressed to desktop context. In order to mitigate this threat, we intend to further investigate the proposed approach in other contexts.

External validity. As the case study was executed in one small academic SPL, it is difficult to make generalizations. The findings and discussions in this study are delimited for this SPL context. Despite the limitations, researchers can extend the study by replicating it in different SPL contexts following the design of this study.

6. CONCLUSIONS

This paper presented the *NFP Framework*, which aims at providing a systematic quality attribute specification for SPL through five main tasks. An NFP Reuse Approach (*NFP-RA*) was proposed to support the product derivation NFP aware process. This approach aims to define a systematic way of reusing previous NFP values in order to minimize the effort of performing a new analysis for each new product. In an SPL, similar products are derived and sometimes, the NFP values that are measured for other products can be adequate or adaptable to new products.

We carried out an empirical evaluation to assess both the framework and the approach applicability. The evaluation consisted of observation activities and an interview with practitioners. Among the observations, we examined the role of the approach 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.

The SPL engineers pointed out that the reuse of NFP values provided by the NFP-RA may be useful where estimated values are enough. If they need an exact value for a particular product, it should be measured. However, whether other values have already been measured, the information on the A-base can guide the new measurements.

It may be difficult to create valid Configuration Filters, as the case study results pointed out. In this effect, as a future work, we plan to implement an infrastructure to support the filter automation. It will support the NFP-RA to allow an automatic configuration of SPL products. An extension of the FeatureIDE plugin[4], where the feature model can be annotated with NFP information, is being planned.

7. ACKNOWLEDGMENTS

This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES[5]) funded by CNPq and FACEPE, grants 573964/2008-4 and APQ-1037-1.03/08 and CNPq grants 305968/2010-6, 559997/2010-8, 474766/2010-1 and FAPESB.

References

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

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

[3] A. Cicchetti, F. Ciccozzi, T. Leveque, and S. Sentilles. 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*, pages 93–102. ACM, 2011.

[4] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns.* Addison-Wesley, 2001.

[5] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Pittsburgh, PA, USA, 1990.

[6] D. Lohmann, O. Spinczyk, and W. SchrÃűder-preikschat. 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*, 2005.

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

[8] V. Myllärniemi, M. Raatikainen, and T. Mannistö. A systematically conducted literature review: Quality attribute variability in software product lines. In *Proceedings of the 16th International Software Product Line Conference - Volume 1*, pages 41–45. ACM, 2012.

[9] S. Sentilles. *Managing Extra-Functional Properties in Component-Based Development of Embedded Systems.* PhD thesis, Mälardalen University, Västerås, Sweden, June 2012.

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

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

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

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

[4]http://wwwiti.cs.uni-magdeburg.de/iti_db/research /featureide/
[5]INES - http://www.ines.org.br