Non-functional Properties in Software Product Lines: A Taxonomy for Classification

Mahdi Noorian[1], Ebrahim Bagheri[1,2], and Weichang Du[1]
University of New Brunswick, Fredericton, Canada[1]
Athabasca University, Edmonton, Canada[2]
m.noorian@unb.ca, ebagheri@athabascau.ca, wdu@unb.ca

***Abstract*—In the recent years, the software product lines paradigm has gained interest in both industry and academia. As in traditional software development, the concept of quality is crucial for the success of software product line practices and both functional and nonfunctional characteristics must be involved in the development process in order to achieve a high quality software product line. Therefore, many efforts have been made towards the development of quality-based approaches in order to address non-functional properties in software product line development. In this paper, we propose a taxonomy that characterizes and classifies various approaches for employing non-functional properties in software product lines development. The taxonomy not only highlights the major concerns that need to be addressed in the area of quality-based software product lines, but also helps to identify various research gaps that need to be filled in future work in this area.**

I. Introduction

A. Software Product Lines

The Software Product Line (SPL) paradigm is a systematic reuse-based software development approach that is founded on the idea of identifying and capturing commonalities and variabilities of software products within a target domain [15]. Such an approach allows a new software product to be rapidly developed by exploiting a set of reusable assets, known as *core assets*, which support the management of commonality and variability. The SPL approach allows for improvements in software quality, time to market, and cost reduction [11]. The software product line approach consists of two main development lifecycles, namely *Domain Engineering* and *Application Engineering* [9]. Domain engineering involves analyzing and modeling the target domain as a whole and producing a set of reusable core assets. On the other hand, application engineering involves developing a domain-specific software product using and through the customization of artifacts that are developed in the domain engineering phase.

B. Non-functional Properties

As a part of the software development process, requirements engineering covers all activities that are involved in identifying, representing, documenting, and managing the set of needs, desired features and preferences of the stakeholders [10]. Requirements can generally be categorized into functional and non-functional. In software system requirement engineering [14], [19], the term Functional Properties (FPs) refer to the characteristics that specify the functions that the system must perform [1]; while, the term Non-functional Properties (NFPs) refers to the characteristics that are not related to the functionality of the software [4] but are essential for the operation and acceptance of the system. In general, FPs define the 'what of a software system whereas NFPs address questions pertaining to the 'how of the software system performance. Usually, NFPs are concerned with imposing a set of restrictions on the development process and so define the overall qualities of the product being developed [10]. NFPs are mostly known as system qualities. Some examples of non-functional properties include aspects such as performance, security, and reliability.

C. Objectives of This Research

The general purpose of our ongoing research is to provide a quality-aware framework for software product line development. To achieve this goal, it is required to understand how the NFPs can be managed and employed in the SPL development lifecycle. More specifically, we need to study how NFPs can be captured, modeled and integrated with functional properties. Also, it is important to investigate how quality-based approaches can be applied in both the domain and application engineering lifecycles. In order to cater quality-aware SPL development, there are several fundamental research questions that need to be answered first, such as:

- What are the main tasks/stages within the domain engineering and application engineering lifecycles that need to be cognizant of quality?
- What are the most frequent NFPs used by SPLs practitioners and how are they modeled and represented?
- How can existing literature on NFPs within the general area of software engineering be adopted in the context of software product lines?
- What are the different types of NFPs that are used in different stages of SPL development?
- How can NFPs be systematically defined and measured in the context of software product lines? (This is specially more complex as a product line has the potential to develop a vast number of individual applications with different quality levels, c.f. [13]).

To come up with a classification framework, it is important to systematically analyze the current state-of-the-research and find answers to some of the above-mentioned questions.

In the first step, we propose a taxonomy that can be used to classify the available research literature in intersection of NFPs and SPLs. The proposed taxonomy provides us with the opportunity to systematically investigate and extract the prominent information from existing research works in NFPs and SPLs and be able to draw valid conclusions about how to best pursue the incorporation of non-functional properties within software product lines.

D. Outline

The remainder of this paper is organized as follows: In Section II, the proposed taxonomy is presented. In addition, the dimensions of the taxonomy are discussed in detail. Section III is devoted to presenting some prominent work in the area of NFPs and SPLs. Then this set of representative work is classified according to our proposed taxonomy. We conclude the paper with conclusions and direction for future work in Section IV.

II. The proposed Taxonomy

The taxonomy characterizes and classifies the main aspects of quality-based approaches in the context of software product lines. In order to clarify the standpoint of NFPs in SPLs, we propose this taxonomy.

The dimensions of the taxonomy and the motivation for selecting them are as follows:

- (Dimension1) *Main lifecycle*: This dimension helps to understand and categorize the various quality-based tasks/steps that need to be performed in the SPL lifecycle in order to cater quality-aware SPL development.
- (Dimension2) *Class*: This dimension investigates the different classes of NFPs that have been identified by the software engineering community and additionally helps to find the most widely known NFPs that have been applied in the SPL development approach.
- (Dimension3) *Measurement*: This dimension aims to extract data about the various methods and techniques that are required for measuring NFPs in the current SPL research work.
- (Dimension4) *Scope of impacts*: This dimension focuses on identifying various types of interaction that might exist between NFPs and FPs or also among the NFPs. Considering and measuring these interactions can help to perform more accurate trade-off analysis in the SPL development process.

The dimensions and sub-dimensions of the proposed taxonomy are depicted in Fig. 1. In the rest of this section, we look at each dimension and its related sub-dimensions in more detail.

A. Dimension 1- Main life-cycle

As defined by Kang et al. [9], SPL development consists of two main lifecycles, namely domain engineering and application engineering. In order to fulfill a quality-based SPL development process, it is required to study the impact of NFPs in both life-cycles separately. This first dimension is devoted to the introduction of the major quality-based tasks/steps that need to be performed in both domain and application engineering lifecycles. In the following sub-sections, we first discuss the major tasks in domain engineering and then for application engineering as well.

Domain Engineering

Based on Kangs definition [9], domain engineering has three main phases, *1) domain analysis, 2) reference architecture development*, and *3) reusable component development*.

1) domain analysis

Domain analysis is the process of identifying, eliciting and modeling the requirements of a family of products. The major quality-based tasks that can be categorized in this phase are: *identification and elicitation* of functional and non-functional properties; *modeling*, which aims to model and represent NFPs in a structured format. In the context of feature models, which represent functional properties, the extended feature model [9] is an instance of a structured representation model for representing NFPs; *integration* of functional and non-functional models, which helps to have both models under one umbrella as it is useful to understand and identify the mutual impacts between them; and *model evaluation*, the derived model itself needs to be evaluated in terms of some expected quality attributes.

2) reference architecture development

The main purpose of the reference architecture development phase is to provide a common architecture or *reference architecture* for a particular domain. On the other hand, in quality-aware reference architecture in addition to FPs the developed architecture is influenced by predefined domain NFPs. In order to obtain quality-aware reference architecture it is required to consider: the *architecture design* stage in such a way that predefined NFPs are employed in the design phase, e.g., the components and connectors should be designed in such a way that support specific quality attributes; *trade-off analysis*, when designing the software architecture to meet any of the NFPs, it is necessary to consider the mutual impacts of the NFPs and analyze the tradeoffs between them. The importance or priority of each NFP differs from system to system; in *architecture assessment* it is required to assess the developed architecture in terms of its capability for supporting the expected NFPs.

3) reusable component development

The last phase of domain engineering is devoted to implementation. It is important to design a component and perform coding in such a way that the expected quality can be satisfied. For example, for achieving the expected security level, developers should follow certain secure coding standards. Therefore, in order to achieve quality-aware implementation, *component design* and *coding* sub-dimension are defined.

Application Engineering

In application engineering, there are also three phases [9], *1) user requirements analysis, 2) application architecture selection*, and *3) application software development*.

1) user requirements analysis

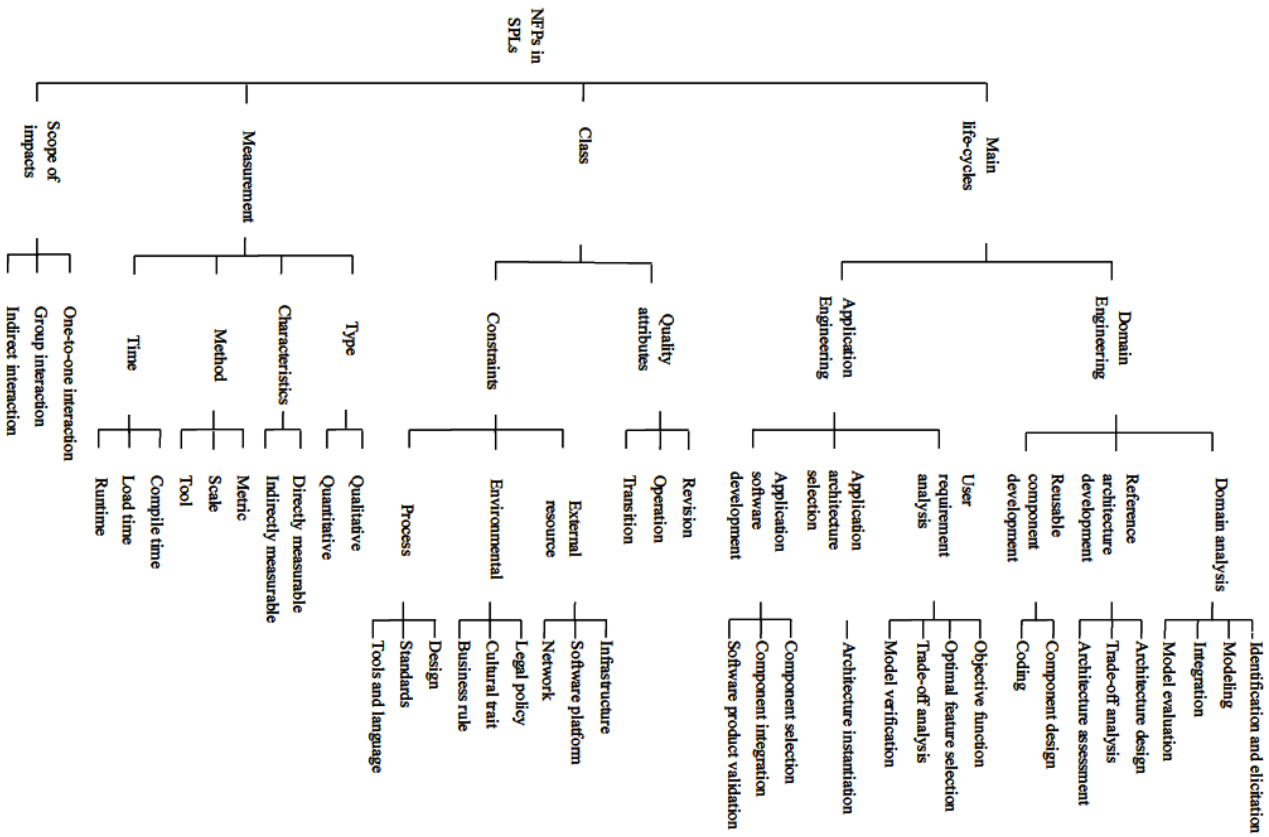

Fig. 1. A taxonomy for non-functional properties in software product lines.

User requirement analysis is concerned with capturing and managing the target product requirements. In this phase, the main task is to derive an instance model from the general domain model that is developed in the domain engineering lifecycle. In the context of feature models, the derived model can be employed in the software product configuration process.

In quality-aware software product configuration, it is required to consider: an *objective function*, which captures the end-user's desirable functional and non-functional properties; *optimal feature selection*, the feature selection process should be conducted according to the end-user's predefined requirements. This process leads to an optimized product. The optimized product is the one that satisfies all desirable FPs and NFPs optimally; *trade-off analysis*, during the configuration process various trade-off scenarios should be defined in order to resolve the potential conflicts and maximize the end-user's intended NFPs; *model verification*, after the target product is configured the final product needs to be assessed based on some verification process. The verification aims to confirm that the target product is consistent with respect to domain and end-users predefined functional and non-functional properties.

2) application architecture selection

As mentioned before, the reference architecture covers all possible software architectures in a products family. In this phase, the main task is to provide an instance from the reference architecture and use it to develop a concrete software product. Therefore, the quality-aware *architecture instantiation* sub-dimension is defined to identify the work that consider both user-defined and domain NFPs in developing the software architecture for a concrete product.

3) application software development

In *application software development*, the main task is to implement target products using the common assets such as reusable components. The role of quality in this phase can be seen in the following steps: *component selection*, *component integration*, and *software product validation*. In both *component selection* and *component integration*, the concern is to select and integrate components in such a way that the target NFPs can be satisfied. In *software product validation*, the purpose is to validate the final implemented product in terms of its predefined NFPs and observe how much the NFPs are satisfied.

B. Dimension 2- Class

The main goal of this dimension is to understand and categorize the NFPs and study the main NFPs that have been employed in SPL development. There are many classification frameworks for NFPs in the literature each of which have been proposed to classify NFPs in general [10], [6] or for a specific application domain [5]. Two classification frameworks were particularly influential to introduce Dimension 2, the McCall quality model [12] and the classification proposed in [18]. The non-functional properties in our proposed taxonomy are

divided into two main categories: *1) quality attributes* and *2) constraints*.

The *quality attributes* sub-dimension addresses quality attributes and based on McCall quality model it can be categorized as: *product operations*, which are concerned with attributes that address the product's operation characteristics, e.g. performance and usability ; *product revision*, which is concerned with the attributes that address the product's ability to undergo changes, e.g. maintainability and testability ; and *product transition*, which is concerned with the attributes that address the products ability to adopt to new environments, e.g. portability and reusability.

On the other hand, we introduce the *constraints* sub-dimension. The NFPs sometimes appear as constraints and impose restrictions on various stages of development. In our proposed taxonomy, *constraints* are categorized as: *external resource*, *environmental*, and *process*. The *external resource* can be categorized as: *infrastructure, software platform* and *network*. The *external resource* can impose specific quality constraints on the target software product, e.g., the minimum network bandwidth for the software product *P* is 1 Mb/s, or the product *P* is only compatible with Windows OS platform. In addition, *environmental* constraints can be identified as: *legal policy, cultural trait* and *business rule*, e.g., in order to obtain Canadian market, the software product *P* needs to support both French and English languages. Also, *process* constraint can be categorized as: *design, standards* and *tool and language*, e.g., the software product *P* needs to be developed based on Web 2.0 standards and using the Java language.

C. Dimension 3- Measurement

There are various methods that have been proposed for measuring NFPs in traditional software engineering. Recently, a number of methods and techniques have been proposed for measuring NFPs in the SPL context. This dimension is devoted to studying the approaches that have been employed to measure the impact of non-functional properties. The *measurement* dimension is classified as *1) type, 2) characteristic, 3) method,* and *4) time*. Measurement *type* can be classified as*quantitative* and *qualitative*. Quantitative measurement is concerned with numerically measuring the NFPs for a particular artifact with continuous values while qualitative measurement is concerned with NFPs with non-numeric values. Measurement *characteristic* can be classified as *directly measurable* and *indirectly measurable*. The *directly measurable* approach is concerned with measuring the basic attributes that cannot be divided into other attributes. On the other hand, *indirectly measurable* NFPs are those that are dependent on other basic attributes and need to be divided into several measurable attributes. In the measurement *method* sub-dimension, we study *metric* and *scale* sub-dimensions and how they pertain to measurement of each individual NFP. In addition, we take a look at possible *tool* support for NFP measurement. The measurement *time* sub-dimension is categorized as, *compile time, load time*, and *runtime*. This sub-dimension targets the measurement stage. In other words, we intend to realize what NFPs should be measured in what stage of execution time. For instance, the security attribute should be measured at runtime and footprint can be measured at compile time.

D. Dimension 4- Scope of impacts

Functional/nonfunctional properties are usually interdependent and one attribute can affect others. Dimension 4 is defined to identify the research that consider the mutual impacts between functional/non-functional properties and see how this interaction can be measured. We classified this dimension to *one-to-one interaction, group interaction* and *indirect interaction*. We consider *one-to-one interaction* when one functional/non-functional property has impact on a functional/non-functional property based on one-to-one interaction, e.g., security can have direct negative impact on usability. In *group interaction*, a group of functional/non-functional properties have impact on one or group of functional/non-functional property(s). In *one-to-one* and *group* interactions the properties directly impact each other while in *indirect interaction*, one property indirectly impacts others. For instance, assume that quality attribute *a* has direct impact on attribute *b* and attribute *b* has direct impact on attribute *c*. In this case, here attribute *a* has indirect impact on attribute *c*. Identifying and considering the well known interaction during the development process could be invaluable. For instance, assume that an application engineer wants to develop a product where both security and usability attributes are important but the usability attribute is preferred over security. Considering the negative impact that security attribute can have on usability, the application engineer should not select the security related features, which can lead to reduced usability.

III. Survey on Non-functional Requirements and Software Product Lines

In this section, some research work in the area of NFPs and SPLs has been selected for supporting the proposed taxonomy. In Section II, we discussed various dimensions of our proposed taxonomy. Note that, Dimension 1 has been chosen as the main dimension for classification purposes since this dimension is concerned with various potential quality-based tasks in the SPL lifecycle. We have chosen and briefly introduced four interesting works (due to space limitation) that have employed quality-based approaches in the software product line paradigm. In addition, we show how these four works address different stages of SPL lifecycle, what classes of NFPs are addressed, and how the measurement techniques are employed. In our future work, we will comprehensively classify the relevant related literature within our proposed taxonomy. The following four works are only introduced to show the potential of our taxonomy. A mapping of taxonomy to several selected work in NFPs and SPLs is shown in Table 1.

In [2], Bagheri et al. proposed an approach to predict the maintainability of a software product line by measuring the quality of the designed feature model in the early stages of the development process. For this purpose, a set of feature

Table 1 TAXONOMY MAPPING ON RESEARCH WORK IN NFPS AND SPLS

		Taxonomy Dimensions			
#	Research work	*Main life-cycle*	*Class*	*Measurement*	*Scope of impacts*
1	Bagheri et al. [2]	- Domain eng. (domain analysis) - Model evaluation	- Quality attributes (Maintainability: analyzability, changeability, understandability)	- Type (quantitative); Characteristics (indirectly measureable); Method (feature model structural metric)	N/A
2	Siegmund et al. [17]	- Domain eng. (domain analysis) -Identification & elicitation, Modeling -Application eng. (User requirements analysis) - Optimal feature selection	-Quality attributes (reliability, complexity, footprint, performance)	-Type (qualitative, quantitative); Method (McCabe's cyclomatic complexity); Time(compile ,runtime)	One-to-one
3	Jarzabek et al. [7]	- Domain eng. (domain analysis) - Integration	-All types of NFP under the concept of soft-goal	-Type (qualitative)	-One-to-one -Group -Indirect
4	Roos-Frantz et al. [16]	Domain eng. (domain analysis) -Modeling, Model evaluation Application eng (User requirements analysis) -model verification	-Quality attribute (Accuracy) -Constraints ,external resource, infrastructure (Memory consumption)	-Type (quantitative) -Metric (meters, milliseconds, kilobytes)	N/A

model structural metrics such as size and complexity are proposed. According to our proposed taxonomy, this work has been designed for the*domain analysis* phase and the quality-based task was *model evaluation*, which tried to evaluate the feature models quality of design. The non-functional property that is covered in this work is*maintainability* that can be categorized under the *revision* quality attributes. With respect to measurement dimension, *quantitative* measurement has been utilized by the authors. In addition, maintainability was measured *indirectly* using analyzability, changeability, and understandability sub-attributes. Furthermore, the feature model structural parameters such as feature model size, length and complexity are defined as metrics for measurement. The scope of impacts dimension is not applicable for this work since the NFPs mutual interaction is not considered by the authors.

For the work presented in [17], Siegmund et al. have proposed an approach named SPL Conqueror, that addresses the problem of automatically configuring a software product with respect to a set of predefined NFPs. The quality-aware tasks in this work can be categorized in both domain and application engineering. In *domain analysis*, the authors address two type of tasks: the *identification & elicitation* task through which the NFPs of the target domain can be elicited and specified by the domain expert; and the *modeling* task during which the feature model is enriched with the identified NFPs. In addition, the NFPs are measured and the results of the measurements are stored. On the other hand, in the *user requirements analysis* step, the optimal feature set is computed and the optimal software product is derived. Thus, this task would fall under the *optimal feature selection* sub-dimension. Regarding NFP *class* dimension, the selected NFPs by the authors are: reliability, complexity, footprint, and performance. With respect to *measurement* dimension, for *qualitative* measurement, the domain experts assign ordinal value to features. Also, for *quantitative* measurement the domain expert assign a proper metric to each NFP. A *tool* has been provided by the authors to measure the NFPs automatically. For measuring the complexity attribute, McCabe's cyclomatic metric is used. In addition, the Source Monitor tool is used by the authors in the measurement process. For the *time* sub-dimension, the footprint attribute measured is at*compile time* and the performance attribute at *runtime*. In the work by Siegmund et al. the concept of FP-FP interaction is employed which falls under the *one-to-one* interaction sub-dimension.

In the work presented in [7], the integrated modeling framework called feature-softgoal interdependency graph (F-SIG) is proposed. In order to support the concept of quality in SPL, the authors extend the Feature-Oriented Domain Analysis (FODA) [8] method with a goal-oriented approach. From the perspective of our proposed taxonomy, this work can be categorized in the *domain analysis* phase, which supports *integration* of two major modeling approaches namely, FODA and SIG [3]. In terms of the *class* dimension, the F-SIG model supports a wide variety of NPFs under the concept of soft-goal. Note that in this work, since the concept of NFPs is presented using the idea of soft-goals, there was no specific method for measuring the NFPs quantitatively and all NFPs are addressed qualitatively. With respect to the *scope of impacts* dimension, all types of interactions *one-to-one, group*, and *indirect* can be directly understood or inferred from the F-SIG model. The interaction can be considered as: feature-to-feature(FP-to-FP) structural interdependency, NFP-to-NFP structural interdependency, and FP-to-NFP interdependency (the influence of FP on specific NFP can implicitly be expressed).

In [16], Roos-Frantz et al. have proposed an approach for quality-based analysis in software product lines. For verification purposes, the quality-annotated variability model is mapped to a constraint satisfaction problem and using constraint solver the verification task is conducted automatically. Within our proposed taxonomy, this work can be classified under the *domain engineering* and *application engineering* lifecycles. The performed quality tasks in domain engineering are: *modeling*,where the orthogonal variability model (OVM) is extended with NFPs; *model evaluation*, during which the

developed variability model can be evaluated to find a void model, dead elements and false optionals. Both *modeling* and *model evaluation* are categorized in the *domain engineering* phase. On the other hand, in application engineering for the *user requirements analysis* phase, the *optimal feature selection* sub-dimension can be considered as a proposed quality-based task in the work by Roos-Frantz et al. In this case, the general model can be checked to find whether any software product can be identified with respect to the defined NFPs. In this work, the NPFs form both classes, *quality attribute* and *constraints*, e.g., accuracy, cost, latency, and memory consumption. For measurement purposes, this work only support *quantitative* measurement. Based on the application domain, latency is measured with milliseconds and memory consumption is measured with kilobytes.

IV. Conclusions and Future Work

In this work, we have presented a taxonomy regarding the role of NFPs in SPL. This taxonomy focuses on the main aspects that need to be addressed for developing a quality-aware products and product lines in the SPL context. The proposed taxonomy consists of four main dimensions, *main life-cycle*, *class*, *measurement*, and *scope of impacts*. We have discussed each dimension and introduced the related sub-dimensions. In addition, in order to discuss our taxonomy, we briefly survey some prominent research work in the field and appropriately classify them into different categories according to the proposed taxonomy.

As a result of this classification, we have identified some research gaps in the area that need to be addressed. For instance, in [17], the identification of quality attributes and proper metric assignment are dependent on the domain expert's knowledge. In order to reduce human expert interference, a systematic approach such as ontology-based knowledge management could be useful to identify the domain and product related NFPs. Furthermore, many measurement approaches are only focused on a limited set of NFPs while they need to also consider other significant NFPs such as security, usability which could not be measured with quantitative approaches. Furthermore, in [2] and [16], the authors do not consider FPs and NFPs interaction and in [17] interaction is limited to FP-FP. Considering various interactions and addressing them in the represented model would be helpful for analyzing trade-offs and also for performing decision making throughout the development process such as reference architecture development and product configuration stages.

Therefore, with respect to our proposed taxonomy and in order to achieve a quality-aware SPL development framework some major research direction could be identified as: developing a systematic approach for identifying and representing NFPs for specific target domain; introducing a formal modeling method for representing NFPs by considering the potential interactions between them; proposing different measurement methods for measuring various types of NFPs including qualitative non-functional properties.

Our direction for future research is to perform a comprehensive survey in the field and classify all current research work and report the status in the area of NFPs and SPL. The collected information from the classification process will assist us to identify the possible enhancements to fill the existing gaps between these two areas. Our initial probe has shown that our proposed taxonomy is quite strong in providing the means to capture various aspects of work in NFPs and SPLs.

References

[1] Ieee standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, page 1, 1990.

[2] Ebrahim Bagheri and Dragan Gasevic. Assessing the maintainability of software product line feature models using structural metrics. In *Software Quality Journal 19(3):579-612*. Springer, 2011.

[3] Lawrence Chung, Brian Nixon, Eric Yu, and John Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, New York, 2000.

[4] Lawrence Chung and Julio Cesar Prado Leite. Conceptual modeling: Foundations and applications. chapter On Non-Functional Requirements in Software Engineering, pages 363–379. Springer-Verlag, Berlin, Heidelberg, 2009.

[5] M. Galster and E. Bucherer. A for identifying and specifying non-functional requirements in service-oriented development. In *Services - Part I, 2008. IEEE Congress on*, pages 345 –352, july 2008.

[6] M. Glinz. On non-functional requirements. In *Requirements Engineering Conference, 2007. RE '07. 15th IEEE International*, pages 21 –26, oct. 2007.

[7] S. Jarzabek, B. Yang, and S. Yoeun. Addressing quality attributes in domain analysis for product lines. *IEE Proceedings - Software*, 153(2):61–73, 2006.

[8] 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, Carnegie-Mellon University Software Engineering Institute, November 1990.

[9] Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Gerard Jounghyun Kim, and Euiseob Shin. Form: A feature-oriented reuse method with domain-specific reference architectures. *Annals of Software Engineering*, 5:143–168, 1998.

[10] Gerald Kotonya and Ian Sommerville. *Requirements Engineering - Processes and Techniques*. John Wiley & Sons, 1998.

[11] Frank J. van der Linden, Klaus Schmid, and Eelco Rommes. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[12] James A. McCall. *Quality Factors*. John Wiley & Sons, Inc., 2002.

[13] Bardia Mohabbati, Dragan Gasevic, Marek Hatala, Mohsen Asadi, Ebrahim Bagheri, and Marko Boskovic. A quality aggregation model for service-oriented software product lines based on variability and composition patterns. In *The 9th International Conference on Service Oriented Computing (ICSOC 2011)*. Springer, 2011.

[14] Bashar Nuseibeh and Steve Easterbrook. Requirements engineering: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 35–46, New York, NY, USA, 2000. ACM.

[15] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[16] Fabricia Roos-Frantz, David Benavides, Antonio Ruiz-Cortes, Andre Heuer, and Kim Lauenroth. Quality-aware analysis in product line engineering with the orthogonal variability model. *Software Quality Journal*, pages 1–47. 10.1007/s11219-011-9156-5.

[17] Norbert Siegmund, Marko Rosenmuller, Martin Kuhlemann, Christian Kastner, Sven Apel, and Gunter Saake. Spl conqueror: Toward optimization of non-functional properties in software product lines. *Software Quality Journal*, pages 1–31. 10.1007/s11219-011-9152-9.

[18] Ian Sommerville and Pete Sawyer. *Requirements Engineering: A Good Practice Guide*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1997.

[19] P. Zave. Classification of research efforts in requirements engineering. In *Requirements Engineering, 1995., Proceedings of the Second IEEE International Symposium on*, pages 214 – 216, mar 1995.