Requirements Engineering In Feature Oriented Software Product Lines: An Initial Analytical Study

Mohsen Asadi[1], Ebrahim Bagheri[2],Bardia Mohabbati[1], Dragan Gašević[2]
[1]Simon Fraser University, Canada, [2]Athabasca University, Canada
{ masadi, mohhabati}@sfu.ca, dgasevic@acm.org, ebagheri@athabascau.ca

ABSTRACT
Requirements engineering is recognized as a critical stage in software development lifecycle. Given the nature of Software Product Lines (SPL), the importance of requirements engineering is more pronounced as SPLs pose more complex challenges than development of a 'single' product. Several methods have been proposed in the literature, which encompass activities for capturing requirements, their variability and commonality. To investigate the maturity and effectiveness of the current requirements engineering approaches in software product lines, we develop an evaluation framework containing a set of evaluation criteria and assess feature oriented requirements engineering methods based on the proposed criteria. As a result of this initial study, we find out the majority of approaches lacks proper techniques for supporting the validation of family requirements models as well as dealing with delta requirements. Additionally, capturing stakeholders' preferences and applying them during the course of software feature configuration have not been taken into account and addressed in the proposed approaches.

Categories and Subject Descriptors
D.2.1 [**Software**]: Software Engineering – Requirements/ Specification*, methodologies, tools.*

General Terms
Algorithms, Languages, Software Engineering

Keywords
Software Product Line Engineering, Requirements Engineering, Evaluation Criteria.

1. INTRODUCTION
Software Product Line (SPL) engineering is a paradigm in software engineering that aims at improving the quality of products, decreasing the cost of development, and reducing time to market [1]. 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 are developed from a common set of core assets in a prescribed way" [2]. Key concepts in software product lines are reusability (a set of core assets designed and developed for reuse) and variability management (i.e., the commonalities and dissimilarities between products) [3]. Feature-oriented software product line development is one of the most well-known approaches, which relies on the notion of features for identifying variability and commonality between the members of a product line. A feature is defined as a distinguishable aspect, quality or characteristic of a software system or systems [4]. Features are modeled in a tree representation called a *feature model*.

Software product line processes encompass two lifecycles namely the domain and the application engineering [1]. Domain engineering (development for reuse) aims at understanding the target domain and developing reusable artifacts via performing domain requirements analysis, domain design, and domain implementation phases [5]. On the other hand, the application engineering receives the reusable artifacts developed in the domain engineering phase and creates an appropriate application instance for the given requirements. To this end, application engineers identify the requirements of a target application, and design and realize the application by reusing the existing domain concepts and developed artifacts.

Requirements engineering is at the core of software product line engineering. Similar to requirements engineering for a single system [6][7] the success of a software product line highly depends on the correct understanding of the context in which the software product line will be used, the understanding of the domain requirements, and also the proper modeling, and analysis of the requirements [8]. Additionally, due to the new development principles (i.e., variability management and extensive reuse) in software product lines, requirements engineering is more challenging and critical than requirements engineering for a single system. Requirements engineering in product lines can be divided into domain requirements engineering and application requirements engineering lifecycles. In the domain requirements engineering, requirements are developed with the purpose of incorporating reusability and variability into requirements artifacts. On the other hand, in the application requirements engineering phase, target requirements are developed by reusing reference requirements.

SPLC - Vol. II, September 02 - 07 2012, Salvador, Brazil

Several approaches have been developed in the product line engineering community to deal with requirements engineering activities [11][12][13][14][15][16]. In this paper, we concentrate on feature-oriented approaches and analyze them against a set of evaluation criteria. These evaluation criteria were developed in top-down (investigating existing evaluation frameworks) and bottom-up (investigating feature-oriented approaches) manners. In order to ensure the quality of the criteria set, we applied a set of meta-criteria, criteria for evaluating the criteria set [17]. As a result of the evaluation, existing challenges in the requirements engineering for software product lines are highlighted.

The initial results of our study show that there have not yet been enough works on the verification and validation of requirements models. Additionally, stakeholders' preferences have been neglected in many of the approaches along with delta requirements [26], i.e., the requirements that are not covered with the product line.

The paper is organized as follow: Section 2 consists of process-centered description of feature-oriented methods for domain and application requirements engineering. The evaluation criteria are explained in Section 3, which is followed by analysis of the results in Section 4 and conclusion in Section 5.

2. REQUIREMENTS ENGINEERING TECHNIQUES IN FEATURE ORIENTED SOFTWARE PRODUCT LINE

This section reviews a number of prominent feature-oriented software product line requirements engineering approaches. The criteria for selecting these approaches were: 1) The approach proposes a process and defines the steps for developing family requirements models and/or application requirements models; 2) The approach adopts feature driven strategy for variability management; 3) the approach covers the most of requirements engineering activities.

In order to find requirements engineering approaches in software product lines that satisfy abovementioned criteria, we investigated existing literature review papers on the software product line requirements engineering [8][9]and variability management [10]. Using these sources, we selected Feature-Oriented Domain Analysis (FODA) [4], Feature-Oriented Reuse Method (FORM) [18], Cardinality-based Feature Modeling (CBFM) [5], Goal and Scenario Based Domain Requirements Engineering [15], FeatuRSEB[11], and Product Line Use-case for Software and System engineering (PLUSS)[12]. Afterwards, three other approaches i.e. AMPLE[14], Goal-Driven Product Line engineering [16], and AoURN-based Software Product Line [20] were selected based on the knowledge of authors on existing approaches on software product line requirements engineering. Since our work is still work-in-progress, this is not meant to be a comprehensive list.

For each of the approaches we provide a brief overview of its characteristics and description of its processes. Of course, a detail description of these approaches is beyond the scope of this paper and interested readers can find the description in the relevant citations.

2.1 Feature-Oriented Domain Analysis (FODA)

FODA [4] is the first approach developed based on the notion of features and defines a process for developing domain artifacts. The domain requirements engineering in FODA consists of *information analysis, feature analysis,* and *operational analysis* [4]. *Information analysis* captures the domain knowledge in the form of domain entities and relations between them and produces artifacts such as semantic networks, object models, and entity-relationship models; *Feature Analysis* activity identifies visible features – i.e., the capabilities of a domain which are visible to stakeholders, and then variability analysis is performed based on the notion of features in order to identify and extract which features are common or variable between different products of a family. Next, a feature model is constructed and developed. The feature model represents the variability types and relations among features, i.e., Optional, Alternative, and Or relations. *Operational analysis* captures the relationships between objects in the information system and features in the feature model.

2.2 Feature-Oriented Reuse Method (FORM)

FORM [18] extends FODA [4] with activities for the design and implementation of a family and refines the requirements engineering activities.

FORM defines two engineering processes namely domain engineering and application engineering [18]. Domain engineering aims at analyzing systems in a domain and creating reference architectures as well as reusable components based on the analysis results. Domain engineering consists of context analysis, domain (or feature) modeling, and architecture (and component) modeling phases. The context analysis phase determines the scope of a domain and the intended use of the domain applications. The domain modeling phase identifies the features of the domain and their relations and develops a feature model. Finally, architecture modeling produces a reference architecture model including subsystem models, process models, and module models, and establishes mappings between the feature and architecture models. The application engineering phase aims at developing applications using artifacts created in domain engineering. The application engineering process encompasses the analysis of user requirements, the selection of appropriate and valid domain features from a feature model, and the identification of the corresponding reference model.

2.3 Cardinality-based Feature Modeling (CBFM)

Cardinality-based feature modeling proposes extending the original FODA notation with UML-like multiplicities (so-called cardinalities). The main motivation has been derived by practical application and "conceptual completeness" [32]. The cardinality-based feature resembles to the original FODA proposal, where each feature has associated a *feature cardinality* which determines the number of instances of the features that can be part of a product and how many clones of the feature are allowed in a specific configuration. Furthermore, features can be organized in feature groups, which also can feature group cardinality enabling for the restriction of the minimum and maximum number of group members that can be selected for

configuration. A feature is also extended by attribute. This extension enables to add more information about features and include non-functional requirements into feature model, which are used during the configuration process. These types of feature models where feature models extended by attributes and additional information are also called extended, advances or attributed feature model[27][33]. The requirements engineering process in CBFM is similar to the process in FORM approach.

2.4 Goal and Scenario Based Domain requirements analysis

This approach proposes the use of goals and scenarios for capturing requirements in the domain engineering process and representing them as variable use-cases for a product family [15]. Goals are defined as high-level objectives of a business, an organization or a system and are generally categorized into four types including business goals, service goals, interaction goals, and internal goals. Scenarios are also defined as possible behaviors limited to a set of purposeful interactions which take place among several agents. Accordingly, for scenarios and requirements, four abstraction levels are considered namely business, service, interaction, and internal levels.

In the domain requirements engineering, business goals are defined by the organization, and then the business goals are refined into service goals whose combination leads to the achievement of the business goals. From the service goals scenarios showing the functions of products are identified and classified into common, alternative, and optional variant types. From the service level scenarios, interaction goals and interactions scenarios are identified and categorized into proper variation types. The process continues for internal level goals and scenarios. This approach uses variable use case models – a use case model with optional and alternative relations for modeling variable requirements. A number of transfer guideline rules are described to derive use case models from goals and scenarios. To the best of our knowledge, no process for application requirements engineering has been described.

2.5 Goal-driven software product line engineering

This approach integrates goal-oriented requirements engineering into software product lines [16]. Goal models and feature models are used for representing intentional and software variability in product line engineering. Goal-oriented techniques are applied for capturing objectives in a domain and for modeling composition and variability relations between the objectives. In this context, the i* modeling framework was used widely [19].

This approach defines both domain requirements engineering and application requirements engineering processes and provides a pre-configuration algorithm based on the stakeholders' objectives. The domain requirements engineering starts with identifying the high-level objectives of products and refining them using mechanisms provided by goal modeling. Afterwards, by investigating tasks and plans in the goal model, candidate features are extracted. Then, features are composed to form a feature model and variability relations are captured and represented. Simultaneously, the features are mapped to the tasks in the goal model. Application engineering process identifies high-level functional and non-functional objectives of stakeholders and performs backward reasoning to select related tasks. Afterward, the corresponding features from feature models are selected.

2.6 AoURN-based Software Product Line

Mussbacher et al. [20] proposed and developed the Aspect-oriented User Requirements Notation (AoURN) for the context of software product lines. Features, stakeholders, and products are considered as concerns. This approach covers both the domain requirements engineering and application requirements engineering life-cycles.

Domain requirements engineering process starts by creating a goal model and follows by developing a feature model using AoURN profile. Afterwards, AoURN scenario models are used to defined the behavior and structure of each feature in the feature model. Finally, the impacts of features on the stakeholders' goals are formalized in order to create feature impact model. During application engineering, after analyzing the satisfaction level of business objectives of the users in a top-down or bottom-up manner, the final product configuration is created.

2.7 FeatuRSEB

Griss et al. [11] proposed the integration of feature-oriented domain analysis (FODA) with Reuse-driven Software Engineering Business (RSEB) to form FeatuRSEB. RSEB uses reference architecture to provide a reuse-oriented model. In RSEB, the architecture and reusable subsystems are described by use-cases and then are transformed into object models. Finally, traceability links are established between the use-cases and the object models. RSEB manages variability by structuring use-case and object model using explicit variation points and variants.

The FeatuRSEB process includes constructing a use-case model for the product line and simultaneously developing a feature model. After developing the feature model and the use-case model, commonality and variability analysis is performed for the use-case model and the feature model. The use-case construction process includes: 1) constructing a domain actor model; 2) constructing a domain use-case model; 3) performing robustness analysis on the domain use-case model. The process of extracting a feature model from the domain use-case model performs the following steps: 1) identifying mandatory and optional features; 2) decomposing features into sub-features; 3) identifying variants and variation points; 4) performing robustness analysis on the feature model. In the next stages of FeatuRSEB when other products such as object models and implementation models are created, their variability is incorporated into the feature model.

2.8 Product Line Use-case for Software and System engineering (PLUSS)

PLUSS is an approach developed based on FeatuRSEB [11] and aims at managing natural-language requirements specifications for software product lines [13]. The main artifacts developed in PLUSS are 1) a use-case model which shows the whole requirements of the family and 2) a feature model that visualizes variability in the abstract product family use-case model. PLUSS extends the existing notions in the feature model

proposed by FODA and adds "at-least-one-must-be-selected" relation called ''multiple adaptor features". Moreover, alternative features in the feature model were renamed into "single adaptor features". With respect to the use-case model, "black box flow of evens" is adopted for describing use-case scenarios which provides tabular descriptions of use-case scenarios in natural language. Using these notions, the authors were able to relate non-functional requirements to use-cases.

Domain requirements engineering encompasses activities for modeling scenarios in the use-case model and activities for representing variabilities and commonalities in the feature model. Application requirements engineering is performed by introducing new requirements (i.e., requirements of the target application) into the feature model and selecting the proper variants to address the new requirements.

2.9 AMPLE

This approach is based on model driven development of requirements and provides a traceability model between requirement engineering products [14]. Use-case models and feature models are employed to represent requirements in the family and their variability commonality relations, respectively.

It defines activities for domain requirements engineering and application requirements engineering. Domain requirements engineering encompasses: identifying requirements, grouping requirements into features, refactoring requirements and features, modeling SPL features and use-cases, relating features to use -cases, generating SPL use-cases annotated with features, and modeling use-cases as activity diagrams, and specifying composition rules between requirements. The application engineering process includes: defining a software product line configuration, generating a use-case model from the SPL configuration, and producing activity diagrams from a software product line configuration.

The artifacts developed in the domain requirements engineering are family use-case models, activity models, feature models, and traceability models.

3. CRITERIA BASED EVALUATION REQUIREMENTS ENGINEERING APPROACHES

In this section, we evaluate requirements engineering approaches for feature-oriented product line engineering according to a set of the proposed criteria. The criteria set is extracted by investigating research literature on requirements engineering [6][7][8] and software product line engineering [23] [24][25] [26]. In order to ensure about the quality of the criteria set, we have applied meta-criteria notion, criteria for evaluating a criteria set [17]. We defined three meta-criteria: coverage of requirements engineering, coverage of variability and commonality analysis, and coverage of tooling support. The first meta-criterion investigates if the criteria-set contains the required aspects for evaluating techniques with respect to requirements engineering principles and processes. Variability and commonality analysis - the key principles in the success of software product line engineering- forms the second meta-criterion. Adoption of technique by software developers in addition to detailed guidelines (processes) and explicit representations requires tooling support. The third meta-criterion investigates whether the criteria-set contains required criteria to evaluate this aspect.

For the first meta-criterion, we further defined three subcategories based on the knowledge on software development methods formulated in Software Process Engineering Meta-Model (SPEM) [38] and principles related to the context of meta-criterion (i.e. requirements engineering). According to SPEM, two important aspects of software development methods are process and artifact. Therefore, we considered both process and artifact aspects for evaluating the proposed approaches. With respect to process aspect, the generic requirements engineering process needs to encompass requirements elicitation, requirements modeling, requirements validation and verification, and requirements management [6][7]. Hence, these steps form criteria for evaluating process aspects of requirements engineering in the SPL. Several artifacts have been used in the requirements engineering literature to represent requirements. Among the existing artifacts, we selected more commonly used artifacts i.e. goal model, use-case model, scenario based model, and non-functional model as criteria. Also, according to requirements engineering papers, the needs of stakeholders include functional and non-functional requirements as well as preferences over the functional and non-functional requirements.

For the second meta-criterion, similarly we defined both process and artifacts aspects. Due to the nature of software product lines, a variability and commonality management process is distributed in both the domain engineering and the application engineering lifecycles. In the domain engineering lifecycle, the purpose mainly is to capture and model the variability and commonality while in the application engineering the purpose is to reuse the variability and commonality artifacts. According to the variability management literature[34][10], the variability process includes identifying, analyzing, modeling, and binding (configuring) the variability. Hence, requirements engineering approaches should cover these steps to manage variability in requirements models. Managing delta requirements (i.e. requirements that are not covered by the product line) is an important step in the application engineering lifecycle. With respect to artifacts, variability can be represented in a variability dimension (i.e. feature models) or in software development artifacts (e.g. use-cases) or combination of the variability dimension and the development artifacts. Regarding principles in variability management, we considered types of variability and a strategy adopted for developing a product line.

For the third meta-criterion, modeling support, support for traceability, and automatic validation criteria were selected based on criteria defined in [17] for evaluating tooling support.

Table 1 shows the criteria set for evaluating requirements engineering in the software product line domain. We do not claim that the criteria set is complete, but it provides a proper set of criteria to highlight some challenges in the existing requirements modeling techniques in product lines. The results of analysis are shown in table 2. The results are achieved by reading publications related to each method and exploring for coverage of criteria in the publications.

Table 1: A set of criteria for evaluating requirements engineering methods in software product line

Meta-Criteria	Criteria			Description
Coverage of Requirements Engineering	Requirements Types	Functional Requirements		Ability of the technique to manage functional requirements of stakeholders.
		Non-functional Requirements		Ability of the method to manage non-functional requirements of stakeholders.
		Preferences		Ability of the method to manage preferences of stakeholder in terms of prioritization of both functional and non-functional requirements.
	Process Aspect	Requirements Elicitation		Method explicitly defined an activity or mentioned reusing traditional requirements elicitation techniques for requirements elicitation in SPL
		Requirements Modeling and analysis		Method explicitly defined an activity or mentioned reusing traditional requirements modeling and analysis techniques for requirements modeling in SPL
		Requirements Validation and Verification		Method explicitly defined an activity or mentioned reusing traditional requirements validation and verification techniques for requirements validation in SPL
		Requirements Management		Method explicitly defined an activity or mentioned reusing traditional requirements management techniques for requirements management in SPL
	Artifact Aspect	Goal-Model		A representation for objectives of stakeholders
		Use-case model		Representation for functional requirements in the family
		Scenario based model		Representation for behavior part of the systems in family
		Non-functional model		Representation for non-functional requirements in family
Coverage of Variability and commonality	Variability Types	Optional Variability		special case of single variant when only available variant set consists of one variant
		Alternative Variability		from a set of variants in the binding time a single variant is picked
		Multi-parallel variability		From set of variants in the binding time one or more variants can be picked
	Process	Domain Engineering	Identification	A method is explicitly defined for identifying variability and commonality in the approach
			Analyzing	A method explicitly is defined for identifying types of variation and dependency between them
			Modeling	A method is explicitly defined for representing the variability.
		Application engineering	Configuring	A method is explicitly defined about how to bind the variation points and when to bind them.
			Reusing	A method is explicitly defined about how to instantiate reusable requirements artifacts
			Identify Deltas	A method is explicitly defined about how to manage application requirements not covered in the family.
	Product	Variability Dimension		If there is a separate variability dimension for visualizing variability in the family
		Artifact Dimension		If the existing artifacts were extended to incorporate variability into them.
	Adoption Strategy	Proactive Strategy		whether method considers developing the product line from scratch
		Extractive Strategy		Whether method considers developing the product line from set of existing products
		Reactive Strategy		Whether method considers evolving product line for new requirements.
Coverage of tooling support	Automatic Validation			Whether method describes a validation approach for checking inconsistency between variability in variability dimension and other requirements artifacts
	Support for traceability			If method provides vertical and horizontal traceability
	Modeling support			If the method explicitly develop a tooling support

4. ANALYSIS OF THE RESULTS

All the requirements engineering methods under study consider functional requirements in their processes and provide techniques and guidelines to model them and their variability. However, non-functional requirements have received less attention by these methods. Feature-Oriented Domain Analysis (FODA) did not explicitly deal with non-functional requirements, but some extensions such as Benavides et al. [27] and Bagheri et al.[28] extended feature models to capture non-functional requirements and encounter with their variability. PLUSS mentions representing non-functional requirements in use-case notation, but no explanation about capturing and modeling their variability is given. Due to the nature of non-functional requirements, they may conflict with each other and trade-off decisions are required. For example, in many cases security is in conflict with performance and higher security leads to lower performance. Hence, stakeholders should specify their preferences over non-functional requirements. Stakeholders' preferences might be defined in the form of standard priorities (A is more important than B) or in the form of conditional relative priorities (A is more important than B if C holds, otherwise, B is more important). None of existing approaches except AoURN considers

Table 2: the evaluation result of requirements engineering method - method - √ covered with method, × not covered with method, ⊕ partially covered with some extensions of the method

Criteria			FODA[4]	FROM[18]	CBFM [5]	Goal-Driven SPLE[16]	Goal and Scenario [15]	AoURN Based	PLUSS [13]	FeatuRSEB [11]	AMPLE[14]
Requirements Types	Functional Requirements		√	√	√	√	√	√	√	√	√
	Non-functional Requirements		⊕	×	×	⊕	×	√	√	√	×
	Preferences		×	×	×	×	×	×	×	×	×
Process Aspect	Requirements Elicitation		√	√	√	√	√	√	√	√	√
	Requirements Modeling and analysis		√	√	√	√	√	√	√	√	√
	Requirements Validation and Verification		×	×	√	×	×	×	×	×	×
	Requirements Management		×	×	×	×	×	×	√	√	√
Artifact Aspect	Goal-Model		×	×	×	√	√	√	×	×	×
	Use-case model		⊕	×	√	×	√	×	√	√	√
	Scenario based models		×	×	×	×	√	√	×	√	×
	Non-functional representation		×	×	×	√	×	√	√	×	×
Variability Types	Optional Variability		√	√	√	√	√	√	√	√	√
	Alternative Variability		√	√	√	√	√	√	√	√	√
	Multi-parallel variability		√	√	√	√	√	√	√	√	√
Process	Domain Engineering	Identification	√	√	√	√	√	√	√	√	√
		Analyzing	√	√	√	√	√	√	√	√	√
		Modeling	√	√	√	√	√	√	√	√	√
	Application engineering	Configuring	×	√	√	√	×	√	⊕	√	√
		Reusing	×	√	√	√	√	√	⊕	√	×
		Identify Deltas	×	×	×	×	×	×	√	×	×
Product	Variability Dimension		√	√	√	√	×	√	√	√	√
	Artifact Dimension		×	×	×	√	√	×	×	√	√
Adoption Strategy	Proactive Strategy		√	√	√	√	√	√	√	√	√
	Extractive Strategy		⊕	×	√	×	×	×	×	√	√
	Reactive Strategy		⊕	×	×	×	×	×	√	√	×
Tooling Support	Automatic Validation		⊕	×	√	⊕	×	√	×	×	×
	Support for traceability		√	√	√	√	×	√	√	√	√
	Modeling support		√	√	√	√	√	√	√	√	√

capturing stakeholders' preferences. Bagheri et al [28] and Ognjanovic et al [29] introduced the notion of relative importance between non-functional requirements and proposed algorithms to rank requirements based on stakeholders' preferences over non-functional requirements.

Requirements elicitation and analysis and modeling were covered by all the approaches, but a few of the reviewed approaches such as CBFM defined techniques for the validation and verification activities. Additionally, Benavides et al [36] provided verification techniques for feature models using SAT solvers and CSPs. Since inconsistent and incomplete requirements models cause further problems for later stages, i.e., design and implementation, hence it is essential to develop proper techniques to manage validity of a family requirements model. Requirements management activities such as requirements evaluation were covered by featuRSEB, PLUSS, and AMPLE.

In many approaches, goal models or use-cases are used for representing family requirements besides feature models, which represents variability between the features of the product line.

The activities related to identifying and modeling variability and commonality are covered by all techniques and modeling

languages employed in the discussed methods cover three variability types (i.e., optional, alternative, and multiple parallel). Due to diversity of the stakeholders in software product lines, a wide range of functional and non-functional requirements may exist whose variability should be modeled. Proposed methods mainly discuss functional variability and non-functional variability has been neglected.

With respect to application engineering, it is important that in the configuration process when critical decisions need to be made, the stakeholders' judgments and opinions are taken into account and properly addressed. Most of the approaches except FODA and [15] provides mechanisms for configuring feature models and reusing reference requirements models for developing the application requirements model. Moreover, in many cases some requirements for target applications cannot be satisfied through instantiating domain requirements model and the application engineers need to identify delta requirements. All reviewed methods except PLUSS provided no support for this case.

Regarding development strategies, most of the approach adopted a proactive strategy which is most expensive and a high risk strategy with respect to the other strategies. Some extensions to FODA like Chen et al [34] and Alves et al [35] cover reactive and extractive strategies. Also She et al [37]provided technique to extract feature model for the existing product lines. Extractive strategy is the second most adopted strategy and only FeatuRSEB contains all three development strategies.

To summarize, according to initial results of our study, major shortcomings of requirements engineering approaches in software product lines encompass: lack of techniques for validating and verifying requirements models; lack of mechanisms for capturing stakeholder preferences and applying them for the selection of features; and managing delta requirements during application engineering life-cycle. Additionally, most of techniques emphasize on proactive strategy for developing requirements models, which has the highest cost among the other strategies.

5. CONCLUSION

In this paper, we explored requirements engineering in the software product line domain. Product line requirements engineering differs from single system requirements engineering, because 1) there is a set of software products instead of one system; 2) the purpose of requirements engineering in software product lines is the development for reuse and development by reuse; 3) new notions ,i.e., commonality and variability emerged in product lines that must be considered in the process of requirements management; 4) the range and number of involved stakeholders is more than those involved with a single system. Product line requirements engineering is divided into two sub-processes, domain requirements engineering and application requirements engineering.

We identified a set of criteria for evaluating existing requirements engineering approaches. Afterwards, we systematically applied the criteria set into reviewed methods and showed the results in tabular format. The results revealed that non-functional requirements, validation and verification, and preferences have been neglected by researchers. Additionally, delta requirements should be handed during the application engineering process.

6. REFERENCES

[1] K. Pohl, G. Böckle, and F. J. van der Linden. 2005. Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[2] P. Clements, L., and Northrop. 2001. Software Product Lines: Practices and Patterns. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[3] J. Van Gurp, J. Bosch, and M. Svahnberg, "On the notion of variability in software product lines," in Software Architecture, 2001. Proceedings. Working IEEE/IFIP Conference on, 2001, p. 45–54.

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

[5] K. Czarnecki, and U. Eisenecker, Generative Programming: Methods, Tools, and Applications. Addison-Wesley, 2000.

[6] B. H. C. Cheng and J. M. Atlee, "Research Directions in Requirements Engineering," in 2007 Future of Software Engineering, Washington, DC, USA, 2007, p. 285–303.

[7] B. Nuseibeh and S. Easterbrook, "Requirements engineering: a roadmap," in Proceedings of the Conference on the Future of Software Engineering, 2000, p. 35–46.

[8] V. Alves, N. Niu, C. Alves, and G. Valença, "Requirements engineering for software product lines: A systematic literature review," Information and Software Technology, vol. 52, p. 806–820, Aug. 2010.

[9] M. Khurum, T. Gorschek, A systematic review of domain analysis solutions for product lines, Journal of Systems and Software 82 (12) (2009) 1982–2003.

[10] L. Chen, M.A. Babar, N. Ali, Variability management in software product lines: a systematic review, in: Proceedings of the 13th Software Product Line International Conference (SPLC 2009), San Francisco, CA, USA, 2009, pp. 81–90.

[11] M. Griss, J. Favaro, M. d'Alessandro, Integrating feature modeling with the RSEB. Proceedings of the Fifth International Conference on Software Reuse. p. 76–85 (1998).

[12] I. John and D. Muthig, Modeling Variability with Use Cases. Fraunhofer IESE, Technical Report IESE Report No. 063.02/E, 2002.

[13] M. Eriksson, J. Börstler, and K. Borg, "The PLUSS approach-domain modeling with features, use cases and use case realizations," Software Product Lines, p. 33–44, 2005.

[14] M. Alférez, U. Kulesza, N. Weston, J. Araujo, V. Amaral, A. Moreira, A. Rashid, and M. C Jaeger., A Metamodel for Aspectual Requirements Modelling and Composition. Technical report. Universidade Nova de Lisboa, Portugal, 2008.

[15] J. Kim, M. Kim, and S. Park, “Goal and scenario based domain requirements analysis environment,” Journal of Systems and Software, vol. 79, p. 926–938, Jul. 2006.

[16] M. Asadi, E. Bagheri, D. Gašević, M. Hatala, and B. Mohabbati, “Goal-driven software product line engineering,” in Proceedings of the 2011 ACM Symposium on Applied Computing, New York, NY, USA, 2011, p. 691–698.

[17] M. Asadi and R. Ramsin, “MDA-Based Methodologies: An Analytical Survey,” in Proceedings of the 4th European conference on Model Driven Architecture: Foundations and Applications, Berlin, Heidelberg, 2008, p. 419–431.

[18] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh, “FORM: A feature-oriented reuse method with domain-specific reference architectures,” Annals of Software Engineering, vol. 5, no. 1, p. 143–168, 1998.

[19] E. Yu, and J. Mylopoulos, From E-R to “A-R” — Modelling strategic actor relationships for business process reengineering. In *Proc. ER '94 Conf.* (1994), 548-565.

[20] G. Mussbacher, J. Araújo, A. Moreira, and D. Amyot, AoURN-based Modeling and Analysis of Software Product Lines. Software Quality Journal (2011), doi:10.1007/s11219-011-9153-8

[21] S. Deelstra, M. Sinnema, and J. Bosch, “Variability assessment in software product families,” Information and Software Technology, vol. 51, no. 1, p. 195–218, 2009.

[22] A. Metzger, K. Pohl, P. Heymans, P.-Y. Schobbens, and G. Saval, “Disambiguating the Documentation of Variability in Software Product Lines: A Separation of Concerns, Formalization and Automated Analysis,” in 15th IEEE International Requirements Engineering Conference (RE 2007), Delhi, India, 2007, pp. 243-253.

[23] F. Bachmann, “Managing Variability in Software Architectures,” p. 126--132, 2001.

[24] J. Bosch, with M. Svahnberg, J. Van Gurp, and J. Van, Gurp, “A Taxonomy of Variability Realization Techniques,” SOFTWARE—PRACTICE AND EXPERIENCE, vol. 35, p. 705--754, 2001.

[25] G. Halmans and K. Pohl, “Communicating the variability of a software-product family to customers,” Software and Systems Modeling, vol. 2, no. 1, pp. 15-36, 2003.

[26] M. Sinnema and S. Deelstra, “Classifying variability modeling techniques,” Information and Software Technology, vol. 49, p. 717–739, Jul. 2007.

[27] D. Benavides, P. Trinidad, and A. Ruiz-Cortés, “Automated reasoning on feature models,” in LNCS, Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005, 2005, vol. 3520, p. 491–503.

[28] E. Bagheri, M. Asadi, D. Gasevic, and S. Soltani, “Stratified analytic hierarchy process: prioritization and selection of software features,” in Proceedings of the 14th international conference on Software product lines: going beyond, Berlin, Heidelberg, 2010, p. 300–315.

[29] I. Ognjanovic, D. Gašević, E. Bagheri, and M. Asadi, “Conditional preferences in software stakeholders’ judgments,” in Proceedings of the 2011 ACM Symposium on Applied Computing, New York, NY, USA, 2011, p. 683–690.

[30] J. Mylopoulos, L. Chung, B. Nixon “Representing and using nonfunctional requirements: a process-oriented approach,” Software Engineering, IEEE Transactions on, vol. 18, no. 6, pp. 483-497, Jun. 1992.

[31] K. Czarnecki, S. Helsen, and U. Eisenecker, “Staged configuration using feature models,” Software Product Lines, p. 162–164, 2004.

[32] K. Czarnecki, S. Helsen, and U. Eisenecker, “Formalizing cardinality-based feature models and their staged configuration,” Software Process Improvement and Practice, vol. 10, no. 1, pp. 7–29, 2005.

[33] D. Benavides, S. Segura, and A. Ruiz-Cortés, “Automated analysis of feature models 20 years later: A literature review,” Information Systems, Mar. 2010.

[34] K. Chen, W. Zhang, H. Zhao, and H. Mei, "An Approach to Constructing Feature Models Based on Requirements Clustering", in Proceedings of the 13th IEEE International Conference on Requirements Engineering, Paris, France, IEEE Computer Society, 2005, pp. 31-40.

[35] V. Alves, R. Gheyi, T. Massoni, U. Kulesza, P. Borba, and C. Lucena, "Refactoring Product Lines", in Proceedings of the 5th International Conference on Generative Programming and Component Engineering, Portland, Oregon, USA, ACM Press, 2006, pp. 201-210.

[36] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés, "FAMA: Tooling a Framework for the Automated Analysis of Feature Models", presented at First International Workshop on Variability Modelling of Software-intensive Systems, Limerick, Ireland, 2007.

[37] S. She, R. Lotufo, T. Berger, A. Wąsowski, and K. Czarnecki. 2011. Reverse engineering feature models. In Proceedings of the 33rd International Conference on Software Engineering (ICSE '11). ACM, New York, NY, USA, 461-470.

[38] R. Schuppenies and S. Steinhauer, (2001). Software Process Engineering Metamodel, OMG group, November 2002.