REM4DSPL: A Requirements Engineering Method for Dynamic Software Product Lines

Amanda Sousa
UFC, Ceará, Brazil
amandas@alu.ufc.br

Anderson Uchôa
PUC-Rio, Rio de Janeiro, Brazil
auchoa@inf.puc-rio.br

Eduardo Fernandes
PUC-Rio, Rio de Janeiro, Brazil
emfernandes@inf.puc-rio.br

Carla I. M. Bezerra
UFC- Quixadá, Ceará, Brazil
carlailane@ufc.br

José Maria Monteiro
UFC, Ceará, Brazil
monteiro@dc.ufc.br

Rossana M. C. Andrade
UFC, Ceará, Brazil
rossana@ufc.br

ABSTRACT

Context: Dynamic Software Product Line (DSPL) is a set of software products capable of self-adapt and configure in run-time. DSPL products have common features (commonalities) and varying features (managed in run-time according to context changes). **Objective:** DSPL requirements engineering is challenging. Requirements engineers have to carefully plan self-adaptation while eliciting, modeling, and managing variability requirements. This paper introduces a method for DSPL requirements engineering. **Method:** We relied on empirically-derived activities of DSPL requirements engineering to build our method. We selected techniques and templates used in other domains such as SPL for refinement and incorporation into the method. We asked DSPL experts via a survey on the method applicability. **Result:** We introduced the Requirements Engineering Method for DSPL (REM4DSPL). Elicitation is guided by supervised discussions. Modeling relies on feature models. Variability Management is tool-assisted and validated via feature model inspection. DSPL experts agreed on the method applicability and suggested improvements. **Conclusion:** REM4DSPL relies on empirically-derived activities, techniques that have been successfully used by previous work, and templates adapted to the DSPL context. We expect our method to guide requirements engineers in practice.

CCS CONCEPTS

• **Computer systems organization → Reconfigurable computing**; • **Software and its engineering → Software product lines**; *Requirements analysis.*

KEYWORDS

Dynamic Software Product Lines, Requirements Engineering

ACM Reference Format:
Amanda Sousa, Anderson Uchôa, Eduardo Fernandes, Carla I. M. Bezerra, José Maria Monteiro, and Rossana M. C. Andrade. 2019. REM4DSPL: A Requirements Engineering Method for Dynamic Software Product Lines. In *XVIII Brazilian Symposium on Software Quality (SBQS'19), October 28-November 1, 2019, Fortaleza, Brazil.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3364641.3364656

SBQS'19, October 28-November 1, 2019, Fortaleza, Brazil

ACM ISBN 978-1-4503-7282-4/19/10…$15.00
https://doi.org/10.1145/3364641.3364656

1 INTRODUCTION

Software Product Line (SPL) is a family of software products that target a common market segment or mission [12]. SPL products share a common set of features, i.e., commonalities, and a set of varying features, i.e., variabilities [2] [18]. The SPL design ultimately targets a massive software reuse [31]. There is an underlying set of software requirements that characterize an SPL domain [12]. Requirements engineers have to carefully elicit, model, and manage SPL requirements because they summarize the needs of many types of users at once [19]. Thus, the SPL requirements engineering is far from being trivial. Various techniques have been proposed to support the key SPL requirements engineering tasks [17] [45]. Additionally, systematic methods have been proposed to support SPL requirements engineering so far [8] [28].

Context-sensitive computing systems have recently emerged as systems able to self-adapt according to context changes in run-time [40]. Context regards the environment in which the system runs [30] [40]. Aimed at the massive reuse of these systems, we have the Dynamic Software Product Line (DSPL) [21]. DSPL extends the SPL design with context-sensitiveness so that DSPL products self-adapt and configure in run-time [5]. New requirements engineering challenges emerged since then [2] [7] [12] [35]. On the one hand, traditional SPL requirements vary among products whenever variabilities are concerned, but these variabilities are managed at the domain engineering by specialists [8] [20]. Any requested changes are incrementally discussed and implanted after discussions and implementation. On the other hand, DSPL requirements have to be carefully conceived with context variations in mind [5]. Once context variations occur in run-time, specialists have to predict and carefully design as many variations as possible [19].

We advocate that DSPL requirements engineering may only be possible through massive discussions, supervised brainstorming, and systematic documentation. Unfortunately, the literature that aimed to support these activities has fallen short in many ways. Ultimately, previous studies do not systematically support the particularities of requirements engineering at the three most important activities: eliciting requirements, modeling requirements, and managing variabilities. Aimed to fill this literature gap, this paper introduces REM4DSPL: a Requirements Engineering Method for managing variabilities in DSPL. We relied on existing techniques employed for context-sensitive systems and SPL in general.

Whenever possible, we tried to incorporate the use of well-known requirements-related techniques (use case documentation, etc.), so that DSPL engineers feel familiar with the method to some extent.

The current knowledge about the activities required to elicit, model, and manage variability requirements in DSPL is quite scattered in the literature [13]. Such scattering helps little in guiding domain engineers to perform certain activities that are a key to reason about variability requirements in DSPL. Aimed at addressing this gap, we previously performed a literature review [13] that summarized those key activities for performing requirements engineering in DSPL. In this work, we rely on the outcomes of this literature review, not just in terms of activities, but also in terms of supporting tools and techniques for each activity. This paper moves forward by systematically guiding domain engineers through the Requirements Engineering Method for DSPL (REM4DSPL). Our method is constituted of three major phases. 1) Requirements Elicitation: via supervised discussions, both domain engineers and stakeholders discuss the DSPL requirements and their possible variations in run-time. The discussion outcomes are documented with the support of the Context Aware Software Product Line Use Case (CAPLUC) template [16]. 2) Requirements Modeling: with the literature support, engineers specify requirements and features. 3) Variability Management: engineers manage variability requirements with tool support [36] [37] and an adapted version of the FMCheck inspection technique for feature models [14].

We performed a preliminary evaluation of REM4DSPL with two DSPL experts aimed at qualitatively capturing their perception of the method's applicability in real settings. Our preliminary results point out a promising way to support DSPL engineers plus some refinements to be applied and re-evaluated in future work. In summary, we expect REM4DSPL can help DSPL requirements engineering by combining empirically-derived activities and well-known support techniques extracted from other software domains. Section 2 provides background information, aimed at supporting the proper understanding of our work, and discusses related work. Section 3 describes the process we employed for building the REM4DSPL. Section 4 introduces our method. Section 5 presents a preliminary evaluation of the proposed method. Section 6 discusses some threats to the study validity [43]. Finally, Section 7 concludes the paper and suggests future work.

2 BACKGROUND AND RELATED WORK

Section 2.1 discusses the major challenges of requirements engineering for Dynamic Software Product Line (DSPL). Section 2.2 discusses previous studies aimed at supporting DSPL requirements engineering to some extent.

2.1 DSPL Requirements Engineering

There is an increasing demand for software systems capable to adapt their features according to user needs and resources constraints [32]. Many application domains demand capabilities for flexible adaptation and post-deployment reconfiguration. Dynamic Software Product Line (DSPL) [21] addresses this demand by extending the traditional Software Product Line (SPL) with product self-adaptation and configuration in run-time [5]. The products of a DSPL are context-sensitive computing systems; thus, they can self-adapt according to context changes. Similarly to SPL, DSPL products share a common set of features that satisfies the specific needs of a particular market segment or mission [12].

Requirements engineering is the process of defining the software requirements that meet the needs of system users and stakeholders [25]. As for any other software system, requirements engineering is fundamental for DSPL development [5] [9] [21]. In the particular case of DSPL, requirements engineering occurs at the domain engineering through two activities: Domain Analysis and Context Analysis. *Domain Analysis* aims to define and specify the needs and details of the domain that a DSPL must support. *Context Analysis* aims to capture the contexts that a DSPL must support, thereby identifying the context information required to self-adapt and re-configure products in run-time [9].

New requirements engineering challenges for DSPL engineering have emerged when compared to SPL [2] [7] [12] [35]. On the one hand, traditional SPL requirements vary among products whenever variabilities are concerned, but these variabilities are managed at the domain engineering by specialists [8] [20]. Any requested changes are incrementally discussed and implanted after discussions and implementation. On the other hand, DSPL requirements have to be carefully conceived with context variations in mind [5]. Once context variations occur in run-time, specialists have to predict and carefully design as many variations as possible [19]. Unfortunately, the current support to DSPL requirements engineering is quite scattered in the literature [13]. Especially, there is a lack of systematically methods for performing three activities that are key to the DSPL requirements engineering: eliciting requirements, modeling requirements, and managing variabilities. We aim to address this literature gap through our current work.

2.2 Current Support and Limitations

We have found a few studies aimed at supporting the SPL requirements engineering [1] [28]. The first study [1] proposed a scenario-based method to elicit requirements for ubiquitous computing systems. Ubiquitous systems are typically integrated within the environment and people's daily routine; thus, these systems constantly capture context information to operate properly [1] [22] [24]. The authors decided to shorten the scope and focus on eliciting functional requirements only, i.e., the system's perceivable functionalities [10] [29]. The proposed method has three major activities. 1) Scenario Description aimed at describing the possible scenarios in a plain text. 2) Scenario Analysis aimed at analyzing each scenario according to settings (context of the environment), agents or actors, and sequences of actions, events, and goals. 3) Scenario Modeling with the support of the Business Process Model and Notation (BPMN) [42]. Besides the limited scope (functional requirements only), this work provides insufficient support to DSPL requirements engineering due to the lack of guidance to identify and model context feature and dynamic processes.

The second study [28] proposed a whole method for SPL requirements engineering called RiPLE-RE. Differently, from the previous study, the authors proposed a more comprehensive step-by-step constituted of three major activities. 1) Model Scope aimed at modeling the SPL scope in terms of requirements, features, and consistency. 2) Define Requirements aimed at eliciting, describing and

verifying the consistency of requirements. 3) Define Use Cases aimed at eliciting, describing and verifying the consistency of use cases. Unfortunately, this method is quite limited when it comes to reasoning about self-adaptation, especially in DSPL.

We recently have performed a literature review [13] aimed to summarize the current support to DSPL requirements engineering. We then characterized some activities that are key to guide domain engineers along with DSPL requirements elicitation, modeling, and variability management. Unfortunately, we observed the lack of systematic methods that are sufficiently comprehensive to support those activities in practical settings. Thus, we used the aforementioned studies [1] [28] as a basis for characterizing and addressing limitations towards the creation of REM4DSPL.

3 STUDY SETTINGS

This section describes our study design aimed at building and evaluating the REM4DSPL method. Section 3.1 illustrates the steps we followed for building the method. Sections 3.2 and 3.3 summarize the techniques and templates incorporated into the method for supporting specific DSPL requirements engineering activities.

3.1 Steps to Build the Method

Figure 1 presents the five steps designed to build and evaluate the Requirements Engineering Method for DSPL (REM4DSPL). The continuous arrows indicate the relationships between activities. Dashed arrows indicate the relationships between an artifact and activity. We explain below each step.

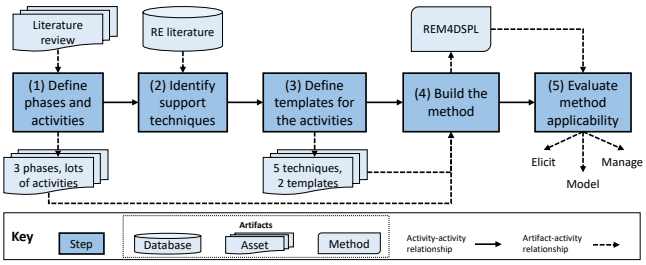

Figure 1: Steps performed to build and evaluate REM4DSPL

Step 1: Define phases and activities. In our previous work [13], we have conducted a systematic literature review in order to investigate how requirements engineering and variability management has been performed in DSPL domain engineering. We have found a total of 37 research papers published from 2008 to 2015. We have extracted from each paper: the activities either recommended or performed by the researchers, besides the artifacts and tools used to support each activity. As a result, we empirically derived a catalog of DSPL requirements engineering activities that served as a basic to the definition of activities and phases covered by REM4DSPL. We have grouped these activities in three phases: *Requirements Elicitation*, *Requirement Modeling*, and *Variability Management*. We describe each phase and activity in Step 4.

Step 2: Identify support techniques. Although we already performed a systematic literature review [13], we did not find at that time requirements engineering techniques specifically shaped for DSPL. Once we were aware of the proposal of new techniques after the literature review's publication date, we performed an *ad hoc* selection of papers to find techniques for incorporation into REM4DSPL. Our goal was identifying requirements engineering techniques either proposed or used by previous studies in three different but related domains: context-sensitive systems (CSS), SPL, and DSPL. We have found the 13 papers listed in Table 1. We listed techniques aimed at supporting requirements elicitation (third column) and modeling (fourth column). The table data helped us to decide which techniques to incorporate into REM4DSPL.

Table 1: Requirements engineering techniques by paper

Paper	Domain	Support Technique	
		Requirements Elicitation	Requirements Modeling
[1]	CSS	Scenario-based	BPMN
[3]		5W1H, Group Storytelling	BPMN, BVCCoN
[22]		metamodel-based	n/a
[24]		n/a	Context Model
[30]		Scenario-based	Context Model Based on Ontology
[39]		5W1H	n/a
[40]		Goal Model	Goal Model
[14]	SPL	n/a	Feature Model, FMCheck
[17]		Existing Assets	Use Cases
[28]		Interview, Brainstorming	Feature Model
[13]	DSPL	n/a	Feature Model, Fama, MOSkitt4SPL, Atlas Model, BPMN, Familiar, FeatureIDE, eMoflon, Odyssey, DOPLER, VariaMos
[36]		ReMINDER	Context Feature Model, Quality Feature Model, DyMMer-NFP
[37]		DyMMer-NFP	Context Feature Model, Quality Feature Model, NFR Catalog

n/a: not applicable

We have found two studies [1] [30] that use *scenario-based* techniques for eliciting requirements and context information for context-sensitive systems. Once these studies relied on empirical validation, we decided to incorporate the scenario-based technique into our method. We also found that many studies (e.g, [36] [37]) rely on *feature models* for modeling requirements. Thus, we also incorporated this technique into our method. We have found only a few DSPL-specific techniques that we could immediately incorporate into the method, such as: *the ReMINDER approach* [36] aimed to support the modeling of non-functional requirements (NFR); and *the DyMMer-NFP tool* [37] aimed at providing a tooling support to modeling NFR and context adaptation scenarios in DSPL feature models. Section 3.2 details some key techniques.

Step 3: Define templates for the activities. Methods for supporting requirements engineering of different domains typically rely on templates that guide requirements documentation [25]. Aimed at defining templates that meet the needs of DSPL requirements engineering, we have looked for previous studies in three domains: context-sensitive computing systems, ubiquitous systems, and DSPL itself. We have selected the Context Aware Software Product Line Use Case (CAPLUC) template [16] in order to systematically describe the scenarios elicited by requirements engineers. CAPLUC is designed for context-sensitive systems, which makes it easy to describe contexts in which a DSPL operates. We also selected the Traceability Matrix template [11] to support the traceability of software requirements and features for different contexts. The Traceability Matrix assists the domain engineer in maintaining traceability of both software requirements and the context features. We present details about some key templates in Section 3.3.

Step 4: Build the method. Aimed to build our method, we first discussed its high-level phases. Based on our previous literature review [13] and our experience with DSPL engineering, we agreed

that the three phases retrieved from the literature review [13] (*Requirements Elicitation*, *Requirement Modeling*, and *Variability Management*) would suffice to guide the DSPL requirements engineering. For each phase, we reasoned about the work products derived from the activities by phase. Finally, we established both activity sequences and inter-dependencies. We relied on BPMN for illustrating the overall REM4DSPL sequence of activities (details in Section 4). REM4DSPL was shaped for constructing a DSPL in a proactive fashion: domain engineers address the DSPL stakeholders' needs from scratch. We explain below each phase.

- **Phase 1: Requirements Elicitation.** This phase consists of systematically identifying the initial DSPL requirements via supervised discussions (with the participation of a mediator) and the documentation of possible adaptation scenarios based on context changes. A scenarios document based in the CAPLUC template is the phase's output. This document enables the definition of functionalities, variabilities, and the context adaptation of the product line.
- **Phase 2: Requirements Modeling.** This phase consists of modeling the DSPL requirements elicited in Phase 1 based on the DSPL particularities, such as the context classification and the adaptation scenarios. Requirements modeling includes: the requirement classification priority (essential, important or desirable); the feature classification by category, variability, and optionality; and the adaptation modeling by the user, computational, and physical environment contexts. A requirement specification document is the phase's output.
- **Phase 3: Variability Management.** This phase consists of tracking requirements and features specified in Phase 2. After that, the features that represent the DSPL domain are systematically represented in the form of a feature model [23]. Feature models provide a general view of mandatory, optional, and alternative features that may compose a DSPL product. Finally, the feature model is verified by means of defects affecting the features and their relationships.

Step 5: Evaluate method applicability. As a preliminary evaluation of REM4DSPL, we have performed a preliminary observational study with DSPL experts. We aimed to obtain the first evidence on the method's applicability from an expert viewpoint. We introduce the evaluation design and results in Section 5.

3.2 Supporting Techniques

Technique 1: Capturing variability in business process models with Provop. Business Process Modeling (BPM) is the activity of representing the underlying operational process of an organization [4]. This activity has been largely employed for the organization of different natures, including software development organizations. BPM provides a general view of organization actors, activities, and their relationships [4]. In software development, BPM is typically employed to represent the candidate system users and how they interact with a software system [4]. Thus, BPM can be used to represent use cases. REM4DSPL adopts the Provop approach [20] for modeling use cases. Provop is particularly interesting for the DSPL domain because it allows capturing and managing variabilities in a single business process model. Figure 2 illustrates the Provop notation, which represents the general activity flow (with activities, connectors, and adjust points) and discriminates alternative flows. With Provop, varying processes can be derived from an existing process through trivial operations, such as addition and deletion (see the bottom corner of the figure).

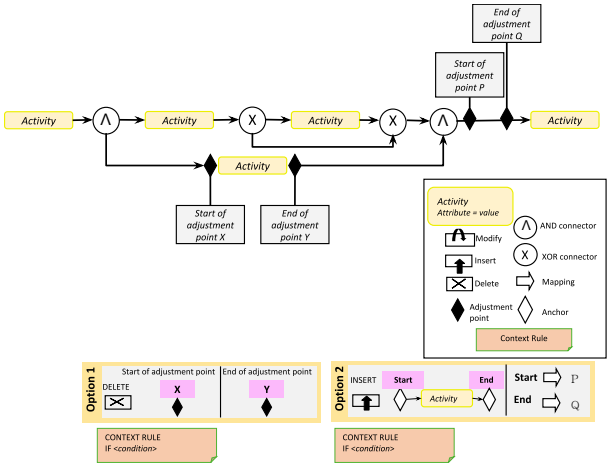

Figure 2: Example that illustrates the Provop notation

Technique 2: Classifying features with Odyssey-FEX. Before performing the feature modeling tasks, we recommend the feature classification with the support of *Odyssey-FEX* [15]. Odyssey-FEX is a meta-model that represents features in three dimensions. 1) Category regards the type of feature to be modeled. Features have originally four categories: entity, domain, technological, and implementation feature [26]. Aimed at providing DSPL-oriented support, we added the context feature category. Once we are concerned about modeling rather than implementation, we did not consider implementation features in our work. 2) Variability regards the variability degree of a feature (invariant or variant). 3) Optionality regards the optionality degree of a feature (mandatory or optional). Table 2 summarize the feature classification standard that we adopted through Odyssey-FEX by dimension.

Table 2: Categories of feature classification

Classification by Category
Entity Feature: Entity Features represent the actors of the model. They are real-world entities that act on the domain. Entity features relate to features of operating environment features and domain features
Domain Features: These features are intrinsically linked to the essence of the domain. They are responsible for representing functionalities and concepts of the model and can be mapped to use cases
Conceptual Features: They are features that represent a concept of the domain
Functional Features: They are features that represent the functionalities of the domain
Technological Features: It is an abstract category that groups features that complement the conceptual layer. These features should not be related to conceptual features
Operating Environment Features: They are features that represent an environment that a domain application can operate or use
Context Features: Features used to detect and manage context conditions, therefore represent characteristics that change the behavior of the model according to the input of a given context
Classification by Variability
Invariant: Fixed or static elements, which are not configurable to the domain
Variant: It is an element fatally linked to a variation point and acts as an alternative to setting to that variation point
Variation Point: It is a feature that reflects the definition of the necessary parameters in the domain in an abstract way. It is configurable through variants
Classification by Optionality
Mandatory: Feature that is present in all derivations of domain configuration changes
Optional: Feature that is not present in all derivations of domain configuration changes

Technique 3: Classifying contexts. Context modeling is important for summarizing and estimating all contexts that a DSPL should address. A key to the success of context modeling is classifying context by means of the context information that helps in describing the context. This work reuses the context categories defined by a previous work [22]. *Use Contexts* typically includes user preferences, agenda, and profile. *Computing Contexts* regard information of the computing environment in which the system will operate, which includes battery autonomy, and available storage, for instance. *Physical Environment Contexts* are usually related to the current time, weather, location, and so forth.

Technique 4: Feature modeling with DyMMer-NFP. We selected the DyMMer-NFP tool [37] to support the DSPL feature modeling with NFR. The tool allows to create, visualize, and analyze an SPL and DSPL feature model. Many customization operations are allowed, such as: adding and deleting features; adding and deleting constraints between functional requirements (FR) and NFR; and modeling multiples context adaptation scenarios. We have used this tool for building the DSPL feature model, specify the context features and the context adaptation scenarios.

Technique 5: Inspecting feature models with FMCheck. To define a feature model that captures all components of a DSPL is fundamental. However, without the proper validation, feature models may incorrectly represent the DSPL domain and, consequently, lead to a poor DSPL product configuration. Aimed at validating DSPL feature models, we have employed FMCheck [14]. FMCheck is a checklist for assuring the model consistency (i.e., at least one DSPL product configuration is valid) and correctness (i.e., there are no dead features) [6]. FMCheck has three parts. *Individual Verification of Features* checks if feature descriptions are clear, correct, and objective; it also validates if the feature actually belongs to the modeled domain. *Verification of Relationships between Features* checks if the feature inter-relations are well represented according to the modeled domain. *Verification of Composition Rules* checks if the feature composition rules are clear, correct, complete, consistent, and relevant to the domain. FMCheck has been shown more effective than *ad hoc* inspections [34].

3.3 Supporting Templates

Template 1: Documenting scenarios with the CAPLUC template. The scenario document is the output artifact of the requirements elicitation phase. It contains scenarios described in tabular form and based on the CAPLUC template [16]. The construction of this document occurs through the detection of correspondences between elements of scenarios narrated by the stakeholders with the elements provided by the CAPLUC template. We have adopted this template because it has a format oriented to context-sensitive systems, which is quite appropriate for DSPL. Figure 3 illustrates a use case modeled from the CAPLUC template. This use case is from the Mobiline SPL [27] aimed to develop guides for context-sensitive mobile tours. The use case comprises a variation point (Capture) with three variants (Sensor, Memory, and External Service). The steps concerning this variation point have the same index because they are alternative to each other; "*" indicates a context constraint (Internet Connection). The illustrated use case says that Sensor, Memory and External Service can capture context, but choosing the latter requires internet connection.

Element	**Description**		
Name	Context Capture		
Reuse Category	Mandatory		
Context Constrain	--		
Summary	Capture context information		
Actors	Context Manager		
Precondition	User must be logged		
Postcondition	Context information was acquired		
Step		**User**	**System**
	1	--	System require to Context Manager some context information
[Capture] Alt [Sensor]	**2**	--	Sensor returns the context information
[Capture] Alt [Memory]	**2**	--	The Memory returns the context information
[Capture] Alt [External Service]	**2***	--	External Service returns the context information
Alternative Flows			
(Constrain)	(Steps)		
Summary of Alternative Variations			
[**Capture**]: "What is the mechanism to get context information?"	**[Sensor]** (Context Constrain: null)		Step 2
	[Memory] (Context Constrain: null)		Step 2
	[External Service] (Context Constrain: Internet Connection)		Step 2
Summary of Optional Variations			

Figure 3: Example of use case written with CAPLUC [16]

Template 2: Modeling contexts with a Provop-adapted template. Context modeling is fundamental to define the DSPL adaptation scenarios. This template guides the DSPL feature modeling and the use case processes, which represent the DSPL adaptation alternatives. Table 3 exemplifies the context model template used by this work, which is quite similar to the one employed by Provop. We applied three refinements here: replace *variable name* with *context name*; value intervals are represented by *context qualification*; and add the *context quantification* attribute. Context qualification determines variation intervals of the context (e.g., high, medium, or low battery). Context quantification determines thresholds [38] by interval (e.g., low $\leq$ 15%). As stated in Section 3.2, we adopted a previous work's context classification [22].

Table 3: Context model sample

Context	**Context Type**	**Qualification**	**Quantification**
Battery	Computational	High	$>$ 75%
		Medium	$>$ 15% AND $\geq$ 75%
		Low	$\leq$ 15%
Bluetooth connection	Computational	Available	True
		Unavailable	False

4 REM4DSPL AT A GLANCE

Figure 7 introduces the Requirements Engineering Method for DSPL (REM4DSPL). We describe each method phase as follows. Section 4.1 discusses the first phase aimed to elicit DSPL requirements. Section 4.2 discusses the second phase in which domain engineers perform the requirements modeling. Finally, Section 4.3 discusses the third and last phase aimed to manage variability.

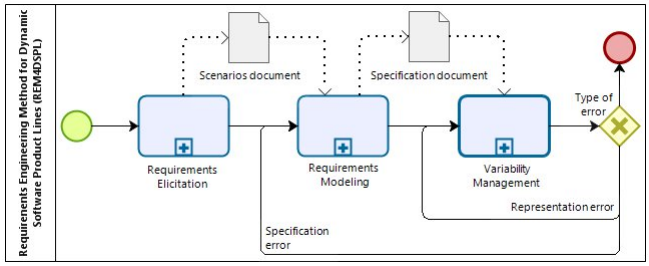

Figure 4: The REM4DSPL Overview

4.1 Phase 1: Requirements Elicitation

Figure 5 overviews the requirements elicitation phase, including the roles, activities, and their relationships. This phase has three roles involved: domain engineer, stakeholders, and redactor. Phase 1 aims to elicit both functional and non-functional system requirements, but also to identify contexts, features, and context features (i.e., dynamic properties that are specific for DSPL).

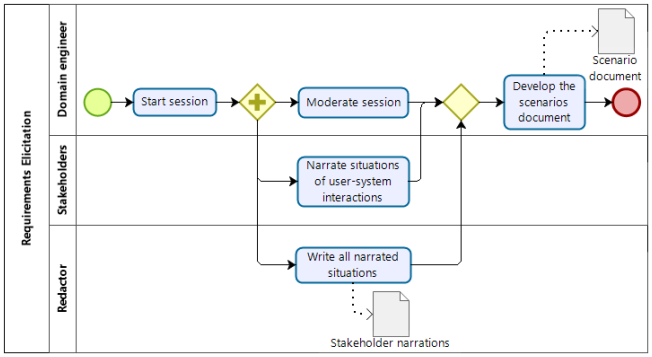

Figure 5: Phase 1 – Requirements Elicitation

As discussed in Section 3.1, we rely on the scenario-based technique because it contributes to the understanding of the domain, analysis of the stakeholders, and requirements elicitation [45]. Discussing scenarios also enables the identification of features, since a feature can be represented as an abstraction of a requirement. By discovering the features present in the scenarios, possible points of variation and their variants are also discovered. Let us take a feature called *connection* as a example; this feature can be variation point whose variants are *wi-fi* and *Bluetooth* connection. Contexts can be identified or mapped through either context constraints or preconditions. In fact, a context is defined by capture knowledge; this knowledge can either restrict or trigger system behaviors along with the system use [44]. Complementary to scenarios, we recommend group narrative techniques [3] to detect as many scenarios as contexts as possible. These techniques include workshop and storytelling. Phase 1 has five activities described below.

- **Start session** occurs when the domain engineer starts the identification of as many DSPL requirements as possible. Once the domain engineer is typically a specialist in the DSPL domain, he must be aware and prepared to discuss different use contexts of the future DSPL products. Thus, the use of scenarios can be properly discussed and documented with broad stakeholder participation.
- **Moderate session** is a fundamental task to the success of the requirements elicitation phase. The domain engineer is responsible for moderating the discussion sessions aimed at ensuring that each participant cooperates with others proportionally. We recommend the allocation of a moderator capable to articulate ideas and keep people engaged.
- **Narrate situations of user-system interactions** occurs when the stakeholders narrate candidate interaction between the DSPL products and their respective users. This activity is important to capture relevant information on the products' expected behaviors, and to elicit major DSPL functionalities.
- **Write all narrated situations** occurs when the redactors take note of all narrated situations in a document.
- **Develop the scenarios document** consists of analyzing all notes made by the redactor in order to summarize them in a well-formatted scenario document.

4.2 Phase 2: Requirements Modeling

Figure 6 overviews the requirements modeling phase. Phase 2 aims to specify the requirements elicited in Phase 1. In this phase, the requirements and features are organized to (1) assure a better understanding of the domain and (2) support the feature modeling. Similar to the traditional software development, functional and non-functional requirements will be documented. Additionally, this phase includes the documentation of contexts and their proprieties, as well as the modeling of business processes of the uses cases.

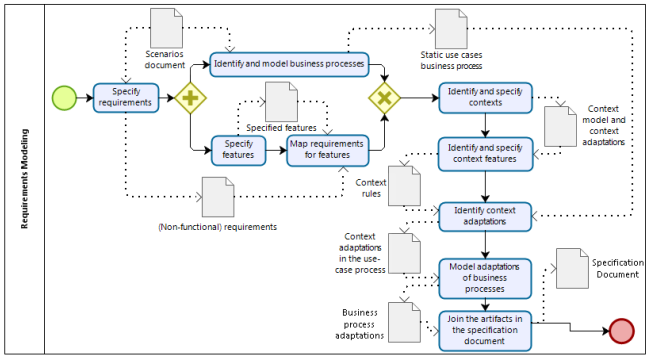

Figure 6: Phase 2 – Requirements Modeling

Phase 2 strongly depends on the scenario document produced in Phase 1. By analyzing the scenario document, it becomes possible to identify and model the business processes and to specify DSPL features. After the features are specified, they must be associated with one or more requirements. Next, contexts must be identified and specified in a context model. From this context model, the context features are specified to enable the identification of variation points that guide the DSPL self-adaptation. Finally, adjustments in the business process of use cases must be modeled. The output of this phase is the document of requirement specification and features. We describe below each of the eight activities of Phase 2.

- **Specify requirements** consists of specifying the DSPL requirements and features. Requirements are documented, mapped to uses cases, and their priority is defined. The requirements specification is performed similarly to traditional software development. Thus, each requirement is defined with an identifier and associated with one or more use cases. NFRs are also specified, and their priorities are specified accordingly. Priority can be categorized, in decreasing order, as: essential, important, and desirable.
- **Identify and model business process** aims to identify use case elements, including triggers that create business logic automatically. After that, the identified elements are modeled into use case processes. These processes guide domain engineers in understanding the system dynamic behavior according to the user interactions. Use case modeling occurs by interrelating use case elements. We convert actions into tasks; triggers become events that may change the process flow. Preconditions determine the system behavior in a particular situation; thus, preconditions provide hints of process contexts. The context constraint element of CAPLUC [16] captures the necessary use case adaptations.
- **Specify features** requires classifying each elicited feature, besides associating features to requirements and defining their priority. The feature is defined as a fundamental, and typically perceivable by stakeholders, characteristics of a product line [26]. We propose specifying features with the Odyssey-FEX meta-model [15]. Each feature is classified by category, variability, and optionality (details in Section 3.2).
- **Map requirements to features** means associating DSPL features to requirements, plus defining the feature priority. The feature-requirement association aims at supporting the variability management: establishing these associations helps in tracking features. This activity is supported by a heuristic that maps variability into domain artifacts of lower level abstraction levels [8].
- **Identify and specify context** means estimating the possible contexts that encompass a DSPL and modeling these contexts in a comprehensive manner. The output is a context model that should carefully address the domain engineer estimations and the contexts elicited with the support of stakeholders. Context models are responsible for qualifying, quantifying, and classifying each context. Context models are very helpful in specifying and documenting context features. We adapted a template originally provided by Provop to support the context modeling (details in Section 3.2).
- **Identify and specify context features** has to major goals. The first goal is identifying the so-called context features, i.e., those features that should be managed in run-time according to the context. The second goal is specifying each context feature. Context features have to be clearly associated with one or more contexts. Thus, it is possible to describe whether and when a feature will be (de)activated for a given context. From the context feature specification, we have the context rules that drive the DSPL self-adaptation.
- **Identify context adaptation** consists of identifying, in the elicited DSPL adaptation scenarios: (1) the DSPL requirements that have variations and (2) which are the variants by requirement. Dynamic adaptation scenarios typically have trigger (i.e., events) and contexts that change the main flow of a use case business process. Thus, it is fundamental to carefully characterize as many variations as possible.
- **Model context adaptations of business process** consists of building a model that represents all context adaptations. This model is responsible for indicating which features belong to each adaptation scenario. This model has a scenario identifier, the specified contexts, and the qualification values by context. After building the model, it is necessary to specify each context feature. The relationships between features and context are defined. Thus, it becomes clear which features should be (de)activated given a context; this information derives helps to derive the context rules.

4.3 Phase 3: Variability Management

Figure 7 overviews the variability management phase. This phase has three major goals. The first goal is building a feature model that summarizes all the DSPL features that should be implemented. The second goal is performing the tracking of variability requirements and features. The third goal is inspecting the feature model. The entire phase has four activities in total, which are distributed according to the three goals as discussed below.

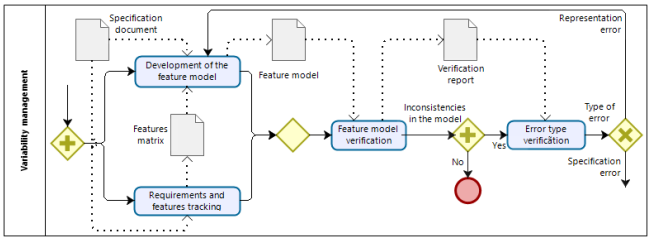

Figure 7: Phase 3 – Variability Management

- **Development of the feature model** means organization in a model that represents all possible product-line configurations [23] [26]. Feature models usually have four layers [26]: *Capability Layer* represents general services, operations, and non-functional aspects of products; *Operating Environment Layer* regards the internal functions of products required to provide services; *Domain Technology Layer* represent the domain-specific techniques employed to implement services and operations; and *Implementation Technique Layer* covers generic implementation techniques for services, operations, and domain functions. This activity is supported by DyMMer-NFP [37], ultimately performed by the domain engineer.
- **Requirements and features tracking.** Traceability is related to the capability to clearly understand the interrelations of entities [41]. Project management has been taking advantage of traceability for understanding, explaining, and evolving processes of different natures. Our work explores traceability management as a means to understand the relationship between DSPL requirements and features. For this purpose, we adapted the *traceability matrix* introduced by a previous work [31]. The original traceability matrix deals

with traditional SPL. Instead of marking as *mandatory* the features that are included into all derived products, we mark these features as either *mandatory* or *context* (i.e., the feature inclusion depends on the context information).

- **Feature model verification** aims at identifying errors in the feature model built during the previous activity. We recommend the feature model inspection guided by FM-Check [14]. This is a checklist-based approach that captures different quality aspects of a feature model, such as correctness and consistency (see Section 3.2). We adapted FMCheck according to the feature classification used in Phase 2.
- **Error type verification.** We recommend errors in the feature model to be classified as follows: omission, incorrect fact, inconsistency, ambiguity, and strange information. If the feature model has no apparent errors, then the phase is completed. Otherwise, in case of representation errors, domain engineers go back to the first activity of Phase 3; in case of specification errors, go back to Phase 2.

5 PRELIMINARY EVALUATION

Section 5.1 introduces the preliminary evaluation goal and participants. Section 5.2 presents the evaluation procedures. Section 5.3 discusses the obtained results.

5.1 Goal and Participants

We followed the Goal Question Metric template [43] to define our evaluation goal as follows: *analyze* some basic activities defined in REM4DSPL; *for the purpose of* understanding the potential method applicability; *with respect to* fundamental DSPL requirements engineering activities (elicit requirements, document contexts, and document use cases) as supported by the method; *from the viewpoint of* two experts in DSPL domain engineering; *in the context of* a small-sized context-sensitive system.

As a preliminary evaluation, we recruited participants (P1 and P2) with a varied background. We asked them about self-assessed skills in a five-point Likert scale: 1 (low), 2 (medium), 3 (good), 4 (high), and 5 (excellent). P1 reported: high skills in SPL, DSPL and feature modeling; good skills in software modeling and dynamic aspects modeling; and medium skills in BPMN. P2 reported: good skills in SPL, DSPL, feature model, and software modeling; medium skills in dynamic aspects modeling; and low skills in BPMN. P1 had between one and five years of experience with SPL and DSPL engineering against less than one year for P2. Both participants: applied a (D)SPL approach to developer product lines along with their research; they had between one and five years of experience with software modeling each; and they had less than one year of experience with dynamic aspects modeling. In summary, we could state that P1 is slightly more experienced than P2.

5.2 Procedures

We designed a controlled evaluation to be performed in a single session and in a research laboratory environment. We did not restrict the total evaluation time, and participants took 3 hours and 18 minutes on average to complete all assigned tasks. We guided the execution of tasks so that the participants could perform them at the same time. We made available the evaluation artifacts (in Portuguese) in the paper's companion website [33]. The controlled evaluation followed three steps as follows:

- **Preparation** consisted of preparing the participants for the evaluation. We first applied a *Consent Form* so that each participant could allow us to collect data of study purposes only. The participants filled out a *Characterization Form* aimed to collect self-assessed skills with (D)SPL engineering and feature modeling. In the sequence, we trained the participants about REM4DSPL (30 minutes were spent) and the evaluation procedures (30 minutes spent). We spent ten minutes to answer questions without compromising the evaluation.
- **Execution** consisted of performing the REM4DSPL evaluation itself. We first distributed the *Activity List* that discriminated the four evaluation activities: 1) Identify use cases from a given use scenario description defined by a researcher; 2) Identify and model requirements, features, context adaptations, and use cases process; 3) Manage variabilities in order to build a feature model for a given domain; and 4) Track requirements and features for different scenarios. After that, we distributed the following support artifacts: 1) Use Case Description; 2) Scenarios Document; 3) Specification Document; and 4) Traceability Matrix Template.
- **Conclusion** consisted of completing the preliminary evaluation. We applied a *Follow-up Form* in which participants could provide their feedback on the performed evaluation.

We present below the use case description.

> *Mary and John are a couple that travels to work a lot. Both do some great meals whenever they are at home. Mary drinks milk every morning, and John always cooks eggs. Aimed to keep products always available and fresh, besides saving money by opening the fridge less frequently, Mary convinced John to buy a smart fridge. Since then, the newly acquired fridge notifies the couple in many ways (either wi-fi, Bluetooth, or even the fridge's LCD screen), about the status of previously registered products. The LCD screen displays the products. The fridge monitors products, via electronic tags, and informs whenever products are close to the expiration date, based on the product bar codes. The fridge also notifies if some a couple's favorite product is missing, or maximum and minimum product amounts are reached. Mary defined that the fridge can have a minimum and maximum of seven and 50 units by product, respectively. Certain amounts of products stored make the fridge to change its cooling and pressurizing settings. all changes are notified to Mary and John. Changes may also reflect the number of times the fridge's door was opened in a given time interval.*

5.3 Preliminary Results

We have investigated the participants' perceptions of the method's applicability based on five questions. We introduces each question and the observed results as follows.

Q_1: *What was the effort expended to identify use cases using the template provided by the method?* – In general, the effort spent to identify use cases was low. The participants have identified four different use cases. Three out of four identified use cases (75%) were expected by considering our reference list of use cases. P2 mentioned that identifying use cases was helpful to elicit DSPL requirements and candidate features. However, P2 also mentioned

his lack of experience with defining some elements of the CAPLUC template [16], e.g., *reuse category*. Such lack of experience partially explains why participants spent a considerable time to identify use cases. Additionally, we consider this feedback as *an opportunity for adding up descriptive information to the template*; thus, domain engineers can clearly understand what kind of information is expected by the template field.

Q_2: *What was the effort expended to perform the identification and specification of features?* – Regarding the activity of identifying features, the participants P1 and P2 reported that a medium and low effort was required, respectively. However, P1 reported a certain difficulty in associating features to requirements. Regarding the creation of rules for the activation and deactivation of features, both participants reported a medium effort. In addition, *P2 has suggested representing in the rules the quantification instead of the qualification.* This suggestion would require a low adaption effort in the template.

Q_3: *What was the effort expended to specify the contexts using the template provided by the method?* – In general, the participants have reported a medium effort to specify the contexts. During the observation activity, P1 said it was difficult to associate the template information with the context modeling. Thus, *we have found an interesting opportunity to design mechanisms (e.g., a notation or a documentation template) that could guide domain engineers during this task.* Conversely, P2 reported that the use of the template was helpful during the modeling of context and their adaptation rules.

Q_4: *What was the effort expended to modeling use cases using the Provop approach proposed by the method?* – In general, both participants reported a medium effort to model use cases using the Provop approach. The participant P1 mentioned that the method might be used to explore and communicate the likely actions of users when interacting with the system. Besides, the participant P1 mentioned that the used approach to modeling the context adaptation might support in identifying alternative behaviors. We observed that the participant P2 was able to model the static process without difficulty. However, P2 has struggled a little while modeling the context adaptation of use cases. *In future work, we could elaborate mechanisms for alleviating such struggle.*

Q_5: *What was the effort expended to build the traceability matrix of the features and the feature model?* – Regarding the traceability matrix, both participants reported a low effort to construct the traceability matrix. For instance, the participant P2 reported that it was possible to visualize possible differences in feature model configuration. The participant P1 also reported that the templates provided by the method facilitated the construction of the traceability matrix. Regarding the construction of the feature model, the participants have used the DyMMer-NFP tool [37] to support the construction of feature models. Both traceability matrix and context adaptation scenarios served as sufficient guidance to building the feature model. None of the participants found hard to build the feature model, which is a quite positive result to the overall selection of supporting techniques and templates for this task.

In summary, both participants (P1 and P2) agreed that REM4DSPL assisted in the elicitation, modeling, and management of variability requirements in DSPL. Participants have emphasized the usefulness of the recommended and the tooling support, especially for feature modeling (which is a key activity). In addition, the facility to perform the elicitation and management of variability has been reported as an advantage of the method. Nevertheless, the time spent by the two DSPL experts to fill up the templates, summed up with the complexity of using Provop for modeling the use case processes, has been mentioned as a drawback. We expect to address these limitations in future work and re-evaluate our method.

6 THREATS TO VALIDITY

We discuss below some threats to the study validity [43].

Construct Validity. We carefully designed the protocol employed for building the REM4DSPL method. For instance, we defined the method's phases and activities based on empirically-derived data. Especially, we relied on the outcomes of a literature review [13] in order to mitigate problems with a low literature coverage. Thus, we expect our method to encompass the key activities of DSPL domain engineering. We also relied on the literature for identifying techniques and templates to support certain activities (see Section 3). Whenever adaptations to the DSPL domain were necessary, the paper authors have discussed strategies to refine the existing techniques and templates. By doing that, we expected to reuse as much literature contributions as possible. With respect to the preliminary method evaluation, we designed and refined the artifacts (especially the forms) prior to the evaluation execution. We validated the forms in pairs after a few discussion rounds, so that we could properly collect data for analysis.

Internal Validity. While building and documenting REM4DSPL, we counted on the participation of at least two of the paper authors. The BPMN diagrams and the descriptions of phases and activities were carefully defined and discussed. Thus, we expected to avoid missing and conflicting information. Regarding the preliminary evaluation, we have followed strict procedures to run the observational study with DSPL experts. We collected the participants' background prior to the evaluation execution. We trained the participants with respect to the method and the evaluation procedures. We answered participants' questions carefully to avoid biasing the evaluation results and misunderstandings.

Conclusion Validity. We knew that it would be quite hard to evaluate the whole method through an observational study that lasted only a few hours. Thus, we carefully performed the qualitative data analysis in order to draw conclusions about the particular activities covered by the evaluation design. We carefully tabulated and validated the participants' data performing the analysis. Thus, we expected to avoid missing and incorrect data.

External Validity. We are aware of the generality of our study results with respect to the DSPL domain engineering community. In fact, our preliminary evaluation counted on only two DSPL experts. We tried to minimize threats to validity by recruiting experts with varied background: while one participant has considerable expertise with (D)SPL modeling, the other participant holds a more modest background. Despite the limitations, we were able to capture some advantages, drawbacks, and improvement opportunities for REM4DSPL. Still, we will plan to perform additional evaluations of the method in the near future.

7 FINAL REMARKS

This paper introduces the Requirements Engineering Method for Dynamic Software Product Line (REM4DSPL). Our method provides

guidance to domain engineers while elicitation, modeling, and managing variabilities in DSPL. REM4DSPL was empirically-derived and relies on techniques and templates either reused or adapted from the literature. Aimed at performing a preliminary evaluation of the method, we counted on the feedback of two DSPL experts. From an expert perspective, our method was quite useful to reveal critical aspects of DSPL (features, interrelations, and adaptation scenarios). The participants provided us with insights for future refinements that can drive a future method re-evaluation. As future work, we plan to improve our method based on the experts' feedback. Especially, we aim to propose mechanisms that alleviate some DSPL domain engineering activities that are still hard to perform.

ACKNOWLEDGMENTS

This work is funded by CNPq (465614/2014-0), INES 2.0, FACEPE (APQ-0399-1.03/17), and CAPES (88887.136410/2017-00).

REFERENCES

[1] Mohammed Alawairdhi and Eisa Aleisa. 2011. A scenario-based approach for requirements elicitation for software systems complying with the utilization of ubiquitous computing technologies. In *35th COMPSACW*. 341–344.
[2] Vander Alves, Nan Niu, Carina Alves, and George Valença. 2010. Requirements engineering for software product lines: A systematic literature review. *Inform. Softw. Tech. (IST)* 52, 8 (2010), 806–820.
[3] Carlos Batista and Carla Silva. 2015. Um Processo Criativo de Elicitação de Contextos para Sistemas Sensíveis ao Contexto. (2015). In Portuguese.
[4] Jörg Becker, Michael Rosemann, and Christoph Von Uthmann. 2000. Guidelines of business process modeling. In *Business process management*. Springer, 30–49.
[5] Nelly Bencomo, Svein Hallsteinsen, and Eduardo Santana De Almeida. 2012. A view of the dynamic software product line landscape. *Computer* 45, 10 (2012), 36–41.
[6] Carla I. M. Bezerra, Jefferson Barbosa, João Holanda Freires, Rossana M. C. Andrade, and José Maria S. Monteiro. 2016. DyMMer: A Measurement-based Tool to Support Quality Evaluation of DSPL Feature Models. In *20th SPLC*.
[7] Andreas Birk, G Heller, I John, S Joos, K Muller, K Schmid, and T von der Massen. 2003. Report of the GI Work Group" Requirements Engineering for Product Lines. (2003).
[8] Ana Paula Terra Bacelo Blois, Regiane Felipe de Oliveira, Natanael Maia, Cláudia Werner, and Karin Becker. 2006. Variability modeling in a component-based domain engineering process. In *9th 2006*. 395–398.
[9] Rafael Capilla, Jan Bosch, Pablo Trinidad, Antonio Ruiz-Cortés, and Mike Hinchey. 2014. An overview of Dynamic Software Product Line architectures and techniques: Observations from research and industry. *J. Syst. Softw. (JSS)* 91 (2014), 3–23.
[10] Lawrence Chung and Julio Cesar Sampaio do Prado Leite. 2009. On non-functional requirements in software engineering. In *Conceptual modeling: Foundations and applications*. Springer, 363–379.
[11] Jane Cleland-Huang, Raffaella Settimi, Chuan Duan, and Xuchang Zou. 2005. Utilizing supporting evidence to improve dynamic requirements traceability. In *13th RE*. IEEE, 135–144.
[12] Paul Clements and Linda Northrop. 2002. Software product lines: practices and patterns. (2002).
[13] Léuson M. P. da Silva, Carla I. M. Bezerra, Rossana M. C. Andrade, and José Maria S. Monteiro. 2016. Requirements Engineering and Variability Management in DSPLs Domain Engineering: A Systematic Literature Review. In *18th ICEIS*. Rome, Italy, 544–551.
[14] Rafael Maiani de Mello, Eldânae Nogueira, Marcelo Schots, Cláudia Maria Lima Werner, and Guilherme Horta Travassos. 2014. Verification of Software Product Line Artefacts: A Checklist to Support Feature Model Inspections. *J. UCS* 20, 5 (2014), 720–745.
[15] Regiane Felipe de Oliveira, Ana Paula Blois, Aline Vasconcelos, and Cláudia Werner. 2005. Metamodelo de Características da Notação Odyssey-FEX: Descrição de Classes. (2005).
[16] Ismayle de Sousa Santos, Rossana M de Castro Andrade, and Pedro de Alcantara dos Santos Neto. 2013. A Use Case Textual Description for Context Aware SPL Based on a Controlled Experiment.. In *25th CAiSE*. 1–8.
[17] Alessandro Fantechi, Stefania Gnesi, Isabel John, Giuseppe Lami, and Jörg Dörr. 2003. Elicitation of use cases for product lines. In *15th PFE*. 152–167.
[18] Eduardo Fernandes, Gustavo Vale, Leonardo Sousa, Eduardo Figueiredo, Alessandro Garcia, and Jaejoon Lee. 2017. No code anomaly is an island. In *16th ICSR*. Springer, 48–64.
[19] Samuel Fricker and Reinhard Stoiber. Relating Product Line Context to Requirements Engineering Processes Using Design Rationale.. In *Software Engineering (Workshops)*. 240–251.
[20] Alena Hallerbach, Thomas Bauer, and Manfred Reichert. 2010. Capturing variability in business process models: the Provop approach. *Journal of Software Maintenance and Evolution: Research and Practice* 22, 6-7 (2010), 519–546.
[21] Svein Hallsteinsen, Mike Hinchey, Sooyong Park, and Klaus Schmid. 2008. Dynamic software product lines. *Computer* 41, 4 (2008), 93–95.
[22] Dan Hong, Dickson KW Chiu, and Vincent Y Shen. 2005. Requirements elicitation for the design of context-aware applications in a ubiquitous environment. In *7th ICEC*. Taiwan, China, 590–596.
[23] Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson. 1990. *Feature-oriented domain analysis (FODA) feasibility study*. Technical Report. Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst.
[24] Kavi Kumar Khedo. 2006. Context-aware systems for mobile and ubiquitous networks. In *International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies (ICNICONSMCL'06)*. IEEE, 123–123.
[25] Gerald Kotonya and Ian Sommerville. 1998. *Requirements engineering: processes and techniques*. Wiley Publishing.
[26] Kwanwoo Lee, Kyo C Kang, and Jaejoon Lee. 2002. Concepts and guidelines of feature modeling for product line software engineering. In *7th ICSR*. 62–77.
[27] Fabiana G Marinho, Rossana MC Andrade, Cláudia Werner, Windson Viana, Marcio EF Maia, Lincoln S Rocha, Eldânae Teixeira, João B Ferreira Filho, Valéria LL Dantas, Fabrício Lima, and others. 2013. MobiLine: A Nested Software Product Line for the domain of mobile and context-aware applications. *Science of Computer Programming* 78, 12 (2013), 2381–2398.
[28] Danuza Neiva, Fernando Cesar de Almeida, Eduardo Santana de Almeida, and Silvio Lemos Meira. 2010. A requirements engineering process for software product lines. In *11th IRI*. 266–269.
[29] Nan Niu and Steve Easterbrook. 2008. Extracting and modeling product line functional requirements. In *16th RE*. 155–164.
[30] Vanessa Tavares Nunes, Flavia Maria Santoro, and Marcos RS Borges. 2007. Capturing context about group design processes. In *11th CSCWD*. 18–23.
[31] Klaus Pohl, Günter Böckle, and Frank J van Der Linden. 2005. *Software product line engineering: foundations, principles and techniques*. Springer Science & Business Media.
[32] P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein. 2010. Requirements-Aware Systems: A Research Agenda for RE for Self-adaptive Systems. In *18th RE*. 95–103.
[33] Amanda Sousa, Anderson Uchôa, Eduardo Fernandes, Carla I. M. Bezerra, José Maria Monteiro, and Rossana M. C. Andrade. 2019. Research website. (2019). Available at: https://anderson-uchoa.github.io/SBQS2019/.
[34] Iuri Souza, Rafael de Mello, Eduardo de Almeida, Cláudia Werner, and Guilherme Travassos. 2016. Experimental evaluation of FMCheck: A replication study. In *15th SBQS*. 121–135.
[35] Anil Kumar Thurimella and Bernd Bruegge. 2007. Evolution in product line requirements engineering: A rationale management approach. In *15th RE*. 254–257.
[36] Anderson G Uchôa, Carla IM Bezerra, Ivan C Machado, José Maria Monteiro, and Rossana MC Andrade. 2017. ReMINDER: an approach to modeling non-functional properties in dynamic software product lines. In *16th ICSR*. 65–73.
[37] Anderson G Uchôa, Luan P Lima, Carla IM Bezerra, José Maria Monteiro, and Rossana MC Andrade. 2017. DyMMer-NFP: Modeling Non-functional Properties and Multiple Context Adaptation Scenarios in Software Product Lines. In *16th ICSR*. 175–183.
[38] Gustavo Vale, Eduardo Fernandes, and Eduardo Figueiredo. 2018. On the proposal and evaluation of a benchmark-based threshold derivation method. *Software Quality Journal (SQJ)* (2018), 1–32.
[39] Vaninha Vieira, Marco AS Mangan, Cláudia Werner, and Marta Mattoso. 2004. Ariane: An awareness mechanism for shared databases. In *10th CRIWG*. 92–104.
[40] Jéssyka Vilela, Jaelson Castro, and João Pimentel. 2016. A systematic process for obtaining the behavior of context-sensitive systems. *Journal of Software Engineering Research and Development* 4, 1 (2016), 2.
[41] Robert Watkins and Mark Neal. 1994. Why and how of requirements tracing. *Ieee Software* 11, 4 (1994), 104–106.
[42] Stephen A White. 2004. Introduction to BPMN. *Ibm Cooperation* 2, 0 (2004), 0.
[43] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Springer Science & Business Media.
[44] Chang Xu, SC Cheung, Xiaoxing Ma, Chun Cao, and Jian Lu. 2012. Adam: Identifying defects in context-aware adaptation. *J. Syst. Softw. (JSS)* 85, 12 (2012), 2812–2828.
[45] Didar Zowghi and Chad Coulin. 2005. Requirements elicitation: A survey of techniques, approaches, and tools. In *Engineering and managing software requirements*. Springer, 19–46.