A Feature-Driven Requirements Engineering Approach for Software Product Lines

Raphael Pereira de Oliveira¹, Emilio Insfran², Silvia Abrahão², Javier Gonzalez-Huerta², David Blanes², Sholom Cohen³, Eduardo Santana de Almeida¹,⁴

¹ Department of Computer Science, Federal University of Bahia (UFBA), Salvador, Brazil
² Department of Information Systems and Computation, Universitat Politècnica de València (UPV), Valencia, Spain
³ Software Engineering Institute (SEI), Carnegie Mellon University, Pittsburgh, USA
⁴ Fraunhofer Project Center (FPC) for Software and Systems Engineering, Brazil

Email: {raphaoliveira, esa}@dcc.ufba.br, {einsfran, sabrahao, jagonzalez, dblanes}@dsic.upv.es, sgc@sei.cmu.edu

Abstract—The importance of Requirements Engineering within software development has long been established and recognized by researchers and practitioners. Within Software Product Lines (SPL), this activity is even more critical because it needs to cope with common, variable, and product-specific requirements not only for a single product but for the whole set of products in the family. In this paper, we present a Feature-Driven Requirements Engineering approach (FeDRE) that provides support to the requirements specification of software product lines. The approach follows a strategy where features are realized into functional requirements considering the variability captured in a feature model. It also provides guidelines on how to associate chunks of features from a feature model and to consider them as the context for the Use Case specification. The feasibility of the approach is illustrated in a case study for developing an SPL of mobile applications for emergency notifications. Preliminary evaluations on the perceived ease of use and perceived usefulness of requirements analysts using the approach are also presented.

Keywords—Product Lines; Requirements Specification; Reuse;

I. INTRODUCTION

Defining requirements to determine what is to be developed is generally accepted as a vital but difficult part of software development. Establishing the driving architectural requirements not only simplifies the design and implementation phases but also reduces the number of errors detected in later stages of the development process, reducing risk, duration and budget of the project [19].

In particular, the specification of requirements in Software Product Lines (SPL) [10] development is even more critical. In this context, it needs to cope with common, variable, and product-specific requirements not only for a single product but for the whole set of products in the family. A fundamental aspect of engineering SPLs is to apply Requirements Engineering (RE) practices to deal with scoping and specifying the product line in both the Domain Engineering and Application Engineering processes.

In the Domain Engineering process, the RE activities are intended to define the extent of the product line to determine its products (scoping), and also to identify common, variable, and product-specific features throughout the product line. The specification of requirements needed to deploy features must also be specified in a systematic way by establishing explicit traceability between features and requirements. In the Application Engineering process, the RE activities are intended to specify the requirements for a particular product in the product family. Therefore, it is important to determine which requirements from the product line are relevant to the product to be developed (common and variant feature selection), and also to refine or to add new specific requirements, which are not in the product line (delta requirements).

Most of the approaches dealing with requirements engineering in SPL development tend to include variability information in traditional requirements models (e.g., use case diagrams) [20] or to extract feature models [16] from requirements specifications following a bottom-up strategy [6] [21]. Some limitations of these approaches arise from the possibility of a large number of requirements and features making the specification of requirements hard to understand, maintain and prone to inconsistencies. The contribution of our approach is that we circumscribe the requirements specifications to deal with complexity in a more effective way. Effectiveness is achieved by chunking the requirement activity based on areas of the feature model. This constrains the extent of the requirements specification at any one time to a more specific area of the product line. We use the feature model as a basis mainly due to the fact that in the SPL community, features are first-class citizens, which are easily identifiable, well-understood, and easy to communicate by SPL developers and domain experts. Hence, there is a strong need to define traceability links between these features and the requirements, and whenever possible, to keep the model and specification synchronized and consistent [4] [5] [15].

In this paper we introduce a Feature-Driven Requirements Engineering (FeDRE) approach to help developers in the requirements engineering activity for SPL development. The approach proposes a set of artifacts, activities, roles, and guidelines on the basis of the features to be developed. Due to space constraints, we focus on the requirements specification of the Domain Engineering activity. We further focus our description of FeDRE by starting once a feature model has been defined (in the scoping activity). However, we do not deal with Quality Attributes (QAs) in the feature model. The next FeDRE
activity consists of the systematic realization of features in terms of use cases. This activity specifies requirements but also establishes traceability between the features and the requirements. This, allows us to provide variability mechanisms at the requirements level (by using use cases and alternative scenarios) according to the chunk of the feature model that these requirements specify. The main contributions of FeDRE is a RE approach that 1) systematically realizes features into requirements considering the variability captured in the feature model and 2) breaks the top-down driven paradigm through use of the feature model to prioritize features according to architecturally significant areas of the product line.

The remainder of the paper is structured as follows. Section II discusses related work on SPL-specific requirements engineering approaches. Section III presents our feature-driven requirements engineering approach. Section IV illustrates the feasibility of the approach through a case study conducted to develop an SPL of mobile applications for emergency notifications. Finally, Section V presents our conclusions and future work.

II. RELATED WORK

Over the last few years, several models and techniques that deal with the specification of requirements in SPL development have been proposed. Traditional requirements models, such as use case models [13] [14], scenario-based models [2] [4] [8] [20], UML models [3] [15] [25], or goal models [26], have been adapted and effectively used as an abstraction mechanism for modeling requirements in SPL development. One of the main differences between traditional RE and RE for product lines is the inclusion of commonality and variability in requirements identification and modeling.

Some approaches combine feature models with more traditional requirements engineering techniques such as use cases [14] [13]. FeatuRSEB [14] proposes simultaneously building a use case model and a feature model, and afterwards performing the commonality and variability analysis first over the use case models and then over the feature model. PLUSS [13] improves the FeatuRSEB approach by adding more variability mechanisms: i) at the use case level; ii) at the alternative scenario level; iii) at the flow of events from an included alternative scenario; and, iv) with cross-cutting aspects that affect several use cases. Neither FeatuRSEB nor PLUSS propose roles or guidelines to help in the RE activity. In addition, input and output artifacts are only partially defined. In FeDRE, the feature model is the main artifact to model variability, and a use case model is built for chunks of this feature model in a systematic way. This improves our ability to deal with complexity by narrowing the context of the use case specification. Regarding the variability mechanisms in requirements, FeDRE borrows the first two types of variability from PLUSS (use case level, and alternative scenario).

The idea of combining a feature model with use cases was also used by the Variability Modeling Language for Requirements (VML4RE) approach in the context of the AMPLE project [4]. This approach presents two main contributions. First, the VML4RE language is a Domain Specific Language able to compose requirements with feature models; and second, a requirements process that uses: i) a feature model to perform the variability identification; ii) use cases and activity diagrams to describe the SPL domain requirements; and a iii) a VML4RE model to relate the feature model with the requirement models. These three models are taken as input with a feature configuration by an interpreter to obtain the application requirements. They also provide consistency checking between features and use case scenarios. An important difference with our approach is that FeDRE guides the requirements analyst in specifying requirements from the feature model, instead of having to focus on constructing the feature model to compose a requirements specification.

Other related work includes approaches that extend different requirements models such as use cases and scenarios with variability concepts without explicitly using feature modeling [8] [20]. The Pulse-CDA approach [8] is a subset of the Pulse methodology [7] that takes the information from the economic scope (a range of system characteristics and a scope definition) and then outputs a domain model (composed of a set of work products that capture different domain views) and a decision model. In Muthig et al. [22], the use-case technique is used as a work product to represent the variability in the use cases. Any element from the use case diagram or in the textual scenario can be a variant (e.g., an actor or a use case). The variant elements are enclosed with XML-style tags to explicitly mark variability. DREAM [20] is a different technique that extends traditional use cases to support variability, where the starting point is a set of legacy systems analyzed to extract the requirements. DREAM uses two stereotypes that are defined to represent variability in use case diagrams: «common» when the use case is always present for every product configuration and «optional» when the use case is present in some product configurations. In Pulse-CDA the decision model is traced to the variable elements in the use case and scenario description in order to instantiate the models. In FeDRE, this variability from use cases and scenarios is traced to the feature model through a traceability matrix. Neither Pulse-CDA nor DREAM propose roles in their RE process to extend the requirements models to support variability. However, both approaches define the input and output artifacts in their processes. Pulse-CDA does not provide guidelines, but DREAM proposed a set of guidelines to obtain the domain requirements specification from legacy systems.

Another traditional RE technique is goal modeling. In Soltani et al. [26], the stakeholder’s intentions represented as goals are mapped to the software features to express their variability in an annotated feature model. In Aspect-oriented User Requirements Notation (AoURN) [21] the authors propose four main domain engineering activities: i) build a stakeholder goal model; ii) build a feature model, where features are represented as goal-tasks with the «feature» stereotype; iii) build the feature impact model to establish the impact of features on the stakeholder’s goals; and iv)
create the feature scenario model, where non-leaf features are described in more detail with the Aspect-oriented Use Case Maps (AoUCM). In AoURN the traceability from features to requirements is done by using links from the stereotyped tasks in the feature model to the AoURN scenario model. These approaches do not define the roles in their processes and only provide partial guidelines for their use. Additionally, the input and output artifacts are only partially defined. Both proposals allow the RE expert to obtain a feature model from a previous goal model. In FeDRE, the starting point is a feature based on concepts that the domain expert directly works with, instead of unfamiliar goal models to guide the creation of the feature model.

Another alternative to specify requirements in SPL is to extend UML traditional notations with variability information. Shaker et al. [25] propose the Feature-Oriented Requirements Modeling Language (FORML) based on feature modeling and UML state-machines. FORML decomposes the requirements into the world model and the behavior model. In the world model, a feature model describes the features that compose the problem. One feature in the world model is decomposed into several feature modules in the behavior model. A feature module is represented with an UML-like finite state machine. This decomposition permits feature modularity, which is one of the main contributions of the work. FORML does not define roles and guidelines in the process in order to obtain the requirements specification. Comparing FORML and FeDRE, both approaches give support to modularity. FORML decomposes a feature model from the world model into several feature modules in the behavior model; similarly, FeDRE allows decomposing sets of features into functional requirements using use cases, scenarios, and traceability links.

In summary, we analyzed several RE approaches for SPL development, and found a disparate set of approaches and techniques. In many cases the scoping and requirements specification activities are considered as independent activities. According to John et al. [18], well-defined relationships and interfaces between scoping and requirements artifacts should be defined in order to reduce rework. To alleviate this problem, FeDRE considers the scoping artifacts as the starting point and defines guidelines to conduct the SPL requirements specification driven by the scoping artifacts. Another important factor is the strategy followed to specify the requirements. Several approaches extend RE models such as use cases [13] [14] or goal models [6] [21] adapted to the SPL domain, and extract feature models from these RE models. In our view, SPL developers and domain experts are more familiar with the concept of feature and variability modeling; therefore, as a way to deal with complexity, we restrict the requirements specification in accordance with chunks of the feature model. Moreover, guidelines to specify functional requirements related to features are provided, thus resulting in an explicit traceability framework built in a systematic way.

### III. A Feature-Driven Requirements Engineering Approach for SPL

The **Feature-Driven Requirements Engineering (FeDRE)** approach for SPLs has been defined considering the feature model as the main artifact for specifying SPL requirements. The aim of the approach is to perform the requirements specification based on the features identified in the SPL domain in a systematic way using guidelines that establish traceability links between features and requirements. By domain, we mean the context where the family of products or functional areas across the products exhibits common, variable or specific functionalities.

The main activities of the FeDRE approach are: Scoping, Requirements Specification for Domain Engineering, and Requirements Specification for Application Engineering. Figure 1 shows Domain Engineering activities where FeDRE is applied. The following roles are involved in these activities: Domain Analyst, Domain Expert, Market Expert and the Domain Requirements Analyst. The Domain Analyst, Domain Expert and Market Expert perform the scoping activity and the Domain Requirements Analyst performs the requirements specification for domain engineering activity.

![Fig 1. Overview of the FeDRE approach](image)

#### A. Scoping

Scoping determines not only what products to include in a product line but also whether or not an organization should launch the product line. According to Bosch [9], the scoping activity consists of three levels: product portfolio scoping, domain scoping, and asset scoping. Product portfolio scoping determines which products and product features should be included in a product line. Domain scoping defines the functional areas and subareas of the product line domain, while Asset scoping identifies assets with costs and benefits estimated for them. In FeDRE, scoping results in three main artifacts: the Feature Model,
the Feature Specification, and the Product Map, using the Existing Assets (if any) as input artifacts. These three artifacts will drive the SPL requirements specification for domain engineering. Each of these artifacts is detailed below.

1) Existing Assets
When performing an extractive or reactive SPL adoption [17], existing assets (e.g., user manual or existing systems) help the Domain Analyst and the Domain Expert identify the features and products in the SPL. Otherwise, a proactive approach can be followed to build the SPL from scratch.

2) Feature Model
Feature modeling is a technique used to model common and variable properties, and can be used to capture, organize and visualize features in the product line. The Domain Analyst and the Domain Expert identify features using existing assets as input or by eliciting information from experts. A Feature Model diagram [16] will identify features, SPL variations, and constraints among the features in the product line.

3) Feature Specification
The Domain Analyst is responsible for specifying the features using a feature specification template. This template captures the detailed information of the features and keeps the traceability with all the involved artifacts. According to the template, each feature must have a unique identifier Feature Id and a Name. The values for the Variability field can be Mandatory, Optional, or Alternative according to the specified feature. The Priority of the feature should be High, Medium or Low. If the feature requires or excludes another feature(s), the Feature Id(s) from the required or excluded feature(s) must be specified. If the feature has a Parent Feature, the Feature Id from the parent feature must be specified. The Binding Time can be compile time or run time, according to the time that the feature will be included in a concrete product [11]. The Feature Type can be concrete or abstract, and the Description is a brief explanation of the feature.

4) Product Map
Each identified feature is assigned to the corresponding products in the SPL. The set of relationships among features and products produces the Product Map artifact, which describes all the required features for building a specific product in the SPL. It is usually represented as a matrix where columns represent the products and rows represent the features.

All these artifacts are the input for the Requirements Specification for Domain Engineering activity, which is described below.

B. Requirements Specification for Domain Engineering

This activity specifies the SPL requirements for domain engineering. These requirements allow realization of the features and desired products identified in the Scoping activity. The steps required to perform this activity are described in the Guidelines for Specifying SPL Requirements, Sub-Section C below.

The FeDRE approach was defined using and extending the PLUSS approach [13], which represents requirements specifications as use case scenarios. The use case scenarios “force requirements analysts to always think about interfaces since separate fields exist for describing actor and system actions”. Our approach supports the relationship between features and use cases; thus, the feature variability is also expressed within the use cases. FeDRE differs from PLUSS in our approach to two types of variability: i) use case variability, considering the whole use case as a variant; and ii) scenario variability, where the variants are alternative scenarios of a use case. In our approach, these two types of variability are sufficient to capture the variations within SPL requirements, and do not require the steps variability nor cross-cutting parameters, as presented in [13].

When a requirement is identified or refined, it should be determined if it is a shared requirement for different products in the product line, or if it is a specific requirement of a single product. Shared requirements must also be classified into common and variable requirements. Common requirements are used throughout the product line and variable requirements must be configured or parameterized in the specification of different variants of the product line. In addition, some requirements may require or exclude other requirements, or may restrict possible configurations of other requirements. Feature models may help in handling the different types of dependencies among requirements, which can be complex and must be properly addressed.

The Requirements Specification for Domain Engineering activity is usually performed in an iterative and incremental way. Therefore, sets of selected features from the Feature Model can be defined as units of increments for the specification (there may be different criteria for choosing features in a unit of increment, e.g., priority of implementation, cost, quality attributes). This activity uses the Feature Model, Feature Specification and Product Map as input artifacts and produces the Glossary, Functional Requirements and Traceability Matrix as output artifacts. Each of these artifacts is detailed below.

1) Glossary
An important characteristic of software product line engineering is the presence of multiple stakeholders, domain experts, and developers. Thus, it is necessary to have a common vocabulary for describing the relevant concepts of the domain. The Glossary describes and explains the main terms in the domain, in order to share a common vocabulary among the stakeholders and to avoid misconceptions. It is represented as a two-column table containing the term to be defined and its description (see Table 2 in Section IV).

2) Functional Requirements
This artifact contains all the functional requirements identified, common or variable, for the family of products that constitute the SPL. Use cases are used to specify the SPL functional requirements (each functional requirement is represented as a use case), and the required variations can be related to the use case as a whole or to alternative scenarios inside a use case. FeDRE adapts the template used in [13] for specifying functional requirements as use cases in order
to support the two types of variability. The specification of functional requirements follows the functional requirements template shown in Table 3 from Section IV. Each functional requirement has a unique Use case id, a Name, a Description, Associated Feature(s), Pre and Post-Conditions, and the Main Success Scenario. Additionally, a functional requirement can be related to an Actor and may have Include and/or Extend relationships to other use case(s). Extends relationships should describe a condition for the extension.

For the Main Success Scenario and the Alternative Scenarios, there are Steps (represented by numbers), Actor Actions (representing an action from the actor) and Blackbox System Responses (representing a response from the system). In addition, for the Alternative scenarios, there is a Name, a Condition and optionally relations to affected features through the Associated Feature field.

3) Traceability Matrix

The Traceability Matrix is a matrix containing the links among features and the functional requirements. The rows in the matrix show the features and the columns show the functional requirements, as shown in Table 4 in Section IV. This matrix is also useful for helping in the evolution of the requirements since each change in the feature model will be traced up to the requirements through the traceability matrix (and vice versa).

C. Guidelines for Specifying SPL Functional Requirements

The purpose of the guidelines is to guide the Requirements Analyst in the specification of SPL functional requirements for domain engineering. The guidelines are based on a meta-model (see Figure 2) that represents the concepts involved when specifying use cases with alternative scenarios and the relationships among them. The meta-model is used to keep the traceability among all the elements and to facilitate understanding. The meta-model comprises the following elements:

- **RequirementsSpecification**: Is the container of all the elements in the specification;
- **Feature**: It represents a feature from a variability model. Although it is not defined in this model, it is related to zero or many requirements;
- **Requirement**: It is an abstract metaclass used to represent functional requirements;
- **UseCase**: Represents a functional requirement. A UseCase is associated with a Feature, other UseCases through the include, extend or inheritance relationships, or with Actors. It contains a Main Scenario and zero or many Alternative Scenarios;
- **UseCasePackage**: Is the container of a UseCaseDiagram;
- **UseCaseDiagram**: Is a view for Actors, UseCases and Relationships;
- **Actor**: Represents an actor and can be related to other Actors or associated with UseCases;
- **Relationship**: Represents the different types of relationships among UseCases, which are Include, Extend and Inheritance;
- **Scenario**: Is an abstract metaclass used to represent the two types of scenarios for the UseCase, which are MainScenario and AlternativeScenario;
- **MainScenario**: Represents the “normal flow” of steps for a UseCase;
- **AlternativeScenario**: Represents an alternative set of steps for a UseCase. It can be associated with a Feature to represent the variability in the scenario;
- **Step**: Represents a step in the MainScenario or AlternativeScenario.

In order to structure the guidelines to specify functional requirements, we address the following questions: i) **Which** features or set of features will be grouped to be specified by use cases? (In our future work, we intend to group features according to QAs) ii) **What** are the specific use cases for the feature or set of features? iii) **Where** should the use cases be specified? (when having a set of features in a hierarchy, do we specify the use cases for each individual feature or only for the parent features?) iv) **How** is the use case specified in terms of steps?

The guidelines consider four types of feature variability that may be present in the feature model, as shown in Table 1. The steps are the following:

<table>
<thead>
<tr>
<th>TABLE 1. FEATURES VARIABILITY</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Mandatory Feature" /></td>
</tr>
<tr>
<td><img src="image" alt="Optional Feature" /></td>
</tr>
<tr>
<td><img src="image" alt="Alternative Feature (OR)" /></td>
</tr>
<tr>
<td><img src="image" alt="Alternative Feature (XOR)" /></td>
</tr>
</tbody>
</table>

1. Identifying Use Cases.

1.1. Identify **which** Features will be grouped together to be specified by use cases. The group of features must belong to the same hierarchy in the feature model;

1.1.1. Root mandatory Features or intermediate mandatory Features must have UseCases;

1.1.2. Mandatory leaf Features may have UseCases or can be specified as alternative scenarios from UseCases in the parent Feature;
1.1.3. Intermediate alternative Features (XOR) must have UseCases;
1.1.4. UseCases identified for leaf alternative Features (XOR) should be specified as alternative scenarios from UseCases in the parent Feature;
1.1.5. Root optional Features or intermediate optional Features must have UseCases;
1.1.6. UseCases identified for leaf optional Features should be specified as alternative scenarios from UseCases in the parent Feature;
1.1.7. Intermediate alternative Features (OR) must have UseCases;
1.1.8. UseCases identified for leaf alternative Features (OR) should be specified as alternative scenarios from UseCases in the parent Feature;
1.2. Identify what the specific UseCases for the feature or group of features are;
1.2.1. UseCases identified for children Features (that have the same parent) with similar behavior must be specified just once in the parent Feature (where);
1.2.2. For each identified UseCase update the Traceability Matrix;
1.3. Each Feature, which has one or more identified UseCase, should have a UseCasePackage;
1.4. Each UseCasePackage should have a UseCaseDiagram;
1.5. Each UseCaseDiagram should have the identified UseCases for the Feature, Relationships and Actors (with the Inheritance and Association Relationships).

2. Specifying Use Cases.
2.1. All the UseCase specifications should follow the specifying use cases with alternative scenarios template (how), as shown in Table 3 in Section IV;
2.2. All mandatory fields from the template must be filled;
2.3. Each Extend must have a condition;
2.4. Create the UseCase MainScenario and UseCase AlternativeScenario. Each step, represented here by a number, comprises the Actor Action and a Black Box Response from the system;
2.5. Each AlternativeScenario should have a name and a condition;
2.6. An AlternativeScenario can be additionally associated with a Feature.

IV. CASE STUDY
An exploratory case study to assess the usefulness of FeDRE was performed by following the guidelines presented in [24]. The stages of the case study development are: design, preparation, collection of data, and analysis of data, each of which is explained below.

A. Design of the case study
The case study was designed by considering the five components that are proposed in [24]: purpose of the study, underlying conceptual framework, research questions to be addressed, sampling strategy, and methods employed. The purpose of the case study is to show the usefulness of applying FeDRE to specify SPL requirements for Domain Engineering. The conceptual framework that links the phenomena to be studied is the feature-driven approach
where the SPL requirements are specified as use cases based on the features. The research questions that are intended to be addressed are: 1) How the SPL development team perceives the FeDRE approach in practice? — Is it easy to use and useful? 2) What limitations does the FeDRE approach present?

The sampling strategy of the case study is based on an embedded single-case design. To answer these questions, FeDRE was applied to develop an SPL for the mobile software called SAVi (http://goo.gl/1Q49O), which is an application that notifies and allows a mobile contact list to track a user in an emergency situation, sending to the contact list a code by SMS and email. We chose SAVi in order to apply FeDRE in a real SPL project using an extractive / reactive SPL adoption. The SPL development team, which applied FeDRE approach, was composed of three people responsible for the scoping activity and six people responsible for the requirements specification activity. One of the authors, with more than 15 years of experience in requirements engineering, was responsible for validating the created artifacts. The technique that was used to obtain feedback about the usefulness of FeDRE was a survey answered by Ph.D. students from Universitat Politècnica de València and Federal University of Bahia after applying the FeDRE approach in practice.

B. Preparation of the case study

With regards to the Scoping activity, all the artifacts (i.e., Feature Model, Feature Specification and Product Map) were created by one domain analyst and one domain expert, who were also assisted by a scoping expert with more than 6 years of experience in SPL scoping activities. The marketing analysis was done based on other products, with similar purpose to that of SAVi, available at the AppStore. Functionalities of these products were included into the SAVi feature model, in which 27 features were identified. Since the FeDRE approach is flexible to support the incremental requirements specification, a set of features was selected to be the driver of the first iteration for the requirements specification. The selection of these features was done based on which features are present in most of the products in the Product Map and are easier to be implemented. Figure 3 shows an excerpt of the Feature Model and the selected features for the first iteration.

Each one of the 27 features from the feature model was specified according to the feature specification template. Also, during the Scoping activity, a list of products for the mobile application for emergency notifications domain was defined allowing the creation of the Product Map artifact. Regarding the Requirements Specification for Domain Engineering activity, two requirements analysts from the team created the Glossary artifact based on the artifacts that were created in the Scoping activity. A total of 16 relevant terms were identified for the domain. An excerpt of this artifact is shown in Table 2.

With the artifacts created by the Scoping activity (Feature Model, Feature Specification and Product Map) and the Glossary artifact created by the Requirements Specification for Domain Engineering activity it was possible to create the Functional Requirements and the Traceability Matrix artifacts by applying the guidelines for specifying SPL requirements.

C. Collection of the data

The data for this case study was collected during the Requirements Specification for Domain Engineering activity. For specifying the SPL Functional Requirements, four additional requirements analysts were recruited. They were asked to apply the guidelines for specifying SPL functional requirements (shown in Sub-Section III-C) in order to answer the following questions: i) Which features can be grouped to be specified by Use Cases (UC)?; ii) What are the specific use cases for the feature or set of features?; iii) Where the use case should be specified?; and iv) How each use case is specified in terms of steps?

a) Which features can be grouped to be specified by UC?

This step analyzes all the features included in the unit of increment for the current iteration, and for these features (see Figure 3) the analyst has to decide which ones will be specified by use cases. According to the guidelines, the requirements analysts decided that the features Contact (according to the step 1.1.1 from the guideline), Add Contact (according to the step 1.1.2 from the guideline) and Import Contact (according to the step 1.1.7 from the guideline) would be specified as use cases. Since there are three ways of implementing an import contact (two optional: Facebook_Import and Twitter_Import; and one

---


---

Table 2: Excerpt from the Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contact</td>
<td>It represents a person to be contacted in an emergency situation. It includes relevant information like e-mail, phone number, Facebook ID, Twitter ID.</td>
</tr>
<tr>
<td>Contact List</td>
<td>Collection of contacts sorted in an alphabetical order.</td>
</tr>
<tr>
<td>Twitter</td>
<td>Micro blogging service. It is a site where the user can share small messages and retrieve contacts to SAVi.</td>
</tr>
<tr>
<td>User</td>
<td>It represents the person who uses the application.</td>
</tr>
</tbody>
</table>

*Mandatory fields.*
Another advantage of using alternative scenarios is that, depending on the selected feature, an alternative specification for one of the use cases related to the feature Import_Contact. b) What are the specific UC for the feature or set of features? After deciding which features will be specified as use cases, the requirements analysts identified what will be the use cases for each feature. Moreover, the Traceability Matrix is incrementally filled in with traceability information between the use case and the feature. For the Contact feature, the following use cases were identified: Show Contact, Delete Contact and Update Contact. For the Add Contact feature, the following use cases were identified: Define Message Language and Add Contact. The Import Contact feature included the following use cases: Retrieve Contacts and Import Contacts. The data representation variability is the reuse. Since the alternative scenarios are specified just once within a use case, several products can be instantiated reusing the same use case. Thus, different behaviors can arise from the same use case, for different products, depending on the selected features. The traceability matrix (features X UC) is shown in Table 4.

<table>
<thead>
<tr>
<th><em>Use_case_id:</em></th>
<th>UC013</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Name:</em></td>
<td>Retrieve Contacts</td>
</tr>
<tr>
<td><em>Description:</em></td>
<td>It retrieves a contact list from an external system</td>
</tr>
<tr>
<td><em>Associated feature:</em></td>
<td>Import Contact</td>
</tr>
<tr>
<td><em>Actor(s)</em> [0..*]:</td>
<td>-</td>
</tr>
<tr>
<td><em>Pre-condition:</em></td>
<td>The system should be allowed to communicate with the external system</td>
</tr>
<tr>
<td><em>Post-condition:</em></td>
<td>The contacts from an external system are retrieved to the system</td>
</tr>
<tr>
<td>Includes To:</td>
<td>-</td>
</tr>
<tr>
<td>Extends From:</td>
<td>-</td>
</tr>
</tbody>
</table>

### Table 3: Retrieve Contacts Use Case Specification

**Main Success Scenario**

<table>
<thead>
<tr>
<th>Step</th>
<th>Actor Action</th>
<th>Blackbox System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The logged user requests his/her retrieve contacts</td>
<td>The System shows the retrieved contacts</td>
</tr>
</tbody>
</table>

**Alternative Scenario name:** Retrieve via Facebook

**Condition:** The logged user must be linked within an Facebook account

<table>
<thead>
<tr>
<th>Associated feature [0..1]:</th>
<th>Facebook.Import</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step</td>
<td>Actor Action</td>
</tr>
<tr>
<td>1.1</td>
<td>[None]</td>
</tr>
</tbody>
</table>

**Alternative Scenario name:** Retrieve via Twitter

**Condition:** The logged user must be linked within a Twitter account

<table>
<thead>
<tr>
<th>Associated feature [0..1]:</th>
<th>Twitter.Import</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step</td>
<td>Actor Action</td>
</tr>
<tr>
<td>1.1</td>
<td>[None]</td>
</tr>
</tbody>
</table>

**Mandatory fields.**

---

**D. Data Analysis**

For the data analysis, 6 requirements analysts, who are Ph.D. students with experience in requirements specification, from Universitat Politècnica de València and Federal University of Bahia, also applied the guidelines for specifying the functional requirements for the SAVi SPL individually. These analysts were different from the authors of the guidelines. After applying the guidelines, a meeting was organized to analyze the obtained results (functional requirements specification and traceability matrix) and the discrepancies were resolved with consensus. The purpose of the meeting was to identify possible problems with the guidelines. The data gathered helped us to fine-tune the definition of the guidelines. For instance, feature variability was made explicit in the guidelines facilitating the identification of which features need a use case specification and a new step was included in the guidelines (the update traceability matrix, step 1.2.2).

After applying the guidelines, the 6 requirements analysts were asked to fill in a survey in order to answer the
two stated research questions (Sub-Section IV-A). This survey is available online (http://goo.gl/RRNzu0). To measure the perceived ease of use and perceived usefulness of the requirements analysts after applying the FeDRE approach (first research question), we relied on an existing measurement instrument proposed for evaluating requirements modeling methods based on user perceptions [1]. Specifically, we adapted two perception-based variables from that instrument, which were based on two constructs of the Technology Acceptance Model (TAM) [12]:

- **Perceived Ease of Use (PEOU):** the degree to which a person believes that using FeDRE would be effort-free. This variable represents a perceptual judgment of the effort required to learn and use the FeDRE approach;
- **Perceived Usefulness (PU):** the degree to which a person believes that FeDRE will achieve its intended objectives. This variable represents a perceptual judgment of the FeDRE approach effectiveness.

Table 5 shows the items defined to measure these perception-based variables. These items were combined in a survey, consisting of 9 statements. The items were formulated by using a 5-point Likert scale, using the opposing-statement question format. Various items within the same construct group were randomized, to prevent systemic response bias. PEOU and PU are measured by using four and five items in the survey, respectively.

**TABLE 5. QUESTIONNAIRE FOR EVALUATING FeDRE**

<table>
<thead>
<tr>
<th>Item</th>
<th>Item Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>PEOU1</td>
<td>The FeDRE approach is simple and easy to follow</td>
</tr>
<tr>
<td>PEOU2</td>
<td>Overall, the requirements specifications obtained by the FeDRE approach are easy to use</td>
</tr>
<tr>
<td>PEOU3</td>
<td>It is easy for me to follow the guidelines proposed by FeDRE approach</td>
</tr>
<tr>
<td>PEOU4</td>
<td>The guidelines for specifying SPL functional requirements are easy to learn</td>
</tr>
<tr>
<td>PU1</td>
<td>I believe that FeDRE approach would reduce the time required to specify SPL requirements</td>
</tr>
<tr>
<td>PU2</td>
<td>Overall, I found the FeDRE approach to be useful</td>
</tr>
<tr>
<td>PU3</td>
<td>I believe that the SPL requirements specifications obtained with the FeDRE approach are organized, clear, concise and non-ambiguous</td>
</tr>
<tr>
<td>PU4</td>
<td>I believe that the FeDRE approach has enough expressiveness to represent functional SPL requirements</td>
</tr>
<tr>
<td>PU5</td>
<td>I believe that the FeDRE approach would improve my performance in specifying SPL functional requirements</td>
</tr>
</tbody>
</table>

The use of multiple items to measure a same construct requires the examination of the reliability of the survey. This was done by using Cronbach’s alpha. The results of the reliability analysis are as follows: (PEOU = 0.764; PU = 0.760). Both constructs have an alpha value equal to, or greater than 0.7, which is a common reliability threshold [23]. As a result, we can conclude that the items in the survey are reliable. The survey also included three open questions in order to gather the opinions of the requirements analysts about the limitations of the approach and suggestions on how to make the FeDRE approach easier to use and more useful. These responses were used to answer the second research question. Both research questions are discussed as follows.

1) **How the SPL development team perceives the FeDRE approach in practice? – Is it easy to use and useful?**

The first research question was addressed by defining the following hypotheses:

- **H1:** FeDRE is perceived as difficult to use, \( H1_1 = H1_2 = 0 \);
- **H2:** FeDRE is perceived as not useful, \( H2_1 = H2_2 = 0 \).

The hypotheses relate to a direct relationship between the use of the FeDRE approach and the users’ perceptions. Table 6 shows descriptive statistics for the perceived-based variables. The results of the survey showed that the subjects perceived FeDRE as easy to use and useful. This can be observed by the mean value for these variables (i.e., all of them are greater than the neutral score (i.e., the score 3)).

To test the hypotheses, we verified whether the scores that the subjects assign to the constructs of the TAM are significantly better than the neutral score on the Likert scale for an item. The scores of a subject are averaged over the items that are relevant for a construct. We thus obtained two scores for each subject. The Kolmogorov–Smirnov test for normality was applied to the PEOU and PU data. As the distribution was normal, we used the One-tailed sample t-test to check for a difference in mean PEOU and PU for the FeDRE approach and the value 3. The results shown in Table 6 allow us to reject the null hypotheses, meaning that we corroborated empirically that the analysts perceived the FeDRE approach as being easy to use and useful.

**TABLE 6. DESCRIPTIVE STATISTICS AND 1-TAILED ONE SAMPLE T-TEST RANK FOR PERCEPTION-BASED VARIABLES**

<table>
<thead>
<tr>
<th></th>
<th>( \bar{x} )</th>
<th>( \sigma )</th>
<th>Mean Diff</th>
<th>t</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PEOU</td>
<td>4.14</td>
<td>0.69</td>
<td>1.14</td>
<td>4.382</td>
<td>0.0025</td>
</tr>
<tr>
<td>PU</td>
<td>3.71</td>
<td>0.75</td>
<td>0.71</td>
<td>2.500</td>
<td>0.024</td>
</tr>
</tbody>
</table>

2) **What limitations does the FeDRE approach present?**

Some limitations were identified after applying the FeDRE approach. Those limitations are related to approach scalability, productivity and effectiveness. Since the use of the approach in large projects is not yet clear, to overcome this scalability limitation we intend to apply FeDRE in new SPL domains with more features than the one used in this paper. To improve the approach productivity, a tool should be developed to support the specification and maintenance of the SPL requirements. Finally, to overcome the lack of effectiveness on the created artifacts, meetings among the requirements analysts should be performed in order to mitigate SPL requirements specification discrepancies.

V. CONCLUSIONS AND FURTHER WORK

This paper introduced the FeDRE approach to support the requirements specification of SPLs. In this approach, chunks of features from a feature model are realized into functional requirements, which are then specified by use cases. The required requirements variations can be related to the use case as a whole or to alternative scenarios inside of a use case. A set of guidelines was provided to help SPL
developers to perform these activities and as a means to systematize the process and ensure a correct traceability between the different requirements artifacts. We believe that this approach provides a solution for dealing with the complexity of specifying SPLs with a large number of requirements and features.

The feasibility of FeDRE was evaluated using a mobile application for emergency notifications case study. The results show that the analysts perceived the approach as easy to use and useful for specifying the functional requirements for this particular product line. However, the approach needs further empirical evaluation with larger and more complex SPLs. Such evaluation is part of our future work. We plan to apply FeDRE in the development of other SPLs at the domain and application engineering processes. In addition, we intend to extend the approach to cope with non-functional requirements, quality attributes in the feature model and explore the use of model-driven techniques to (partially) automate the guidelines to check the completeness and consistency of artifacts.

ACKNOWLEDGEMENTS

This research work is funded by the Hispano-Brazilian Interuniversity Cooperation Program under ref. No.HBP-2011-0015 and the MULTIPLE project (TIN2009-13838) from the Spanish Ministry of Education and Science. This material is based upon work funded and supported by the DoD under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. This material has been approved for public release and unlimited distribution (DM-0000547). This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES²), funded by CAPES, CNPq and FACEPE, grants 573964/2008-4 and APQ-1037-1.03/08 and CNPq grants 305968/2010-6, 559997/2010-8, 474766/2010-1 and FAPESB. The authors also appreciate the value-adding work of all their colleagues Lorenzo Alvim, Larissa Rocha, Ivonei Freitas, Tassio Vale and Iuri Santos who make great contributions to the Scoping activity of FeDRE approach.

REFERENCES


2 INES - http://www.ines.org.br