Continuous Requirements Engineering for Mobile Application Development

Elza Stepanova and Marite Kirikova

Riga Technical University, Latvia
elzastepanova@outlook.com, marite.kirikova@rtu.lv

Abstract. The mobile industry can be defined as one of the most rapidly growing and dynamic IT industries where user requirements are changing very frequently. This is the main reason why project managers are looking for flexible software process models that can adjust to those frequently changing system requirements and produce valuable software in short time and with low budget. The purpose of this paper is to propose a continuous requirements engineering method for mobile application development projects. The proposed method meets the challenges that the requirements for mobile applications change frequently, the development time is short, and the emphasis has to be put on user interfaces. While these challenges are faced not only in mobile application development contexts, the method emerged from and was tested in mobile application domain; and therefore currently cannot yet be advised for other domains.
.

Keywords: Requirements engineering · continuous engineering · agile approach · mobile application development projects.

1 Introduction

Usually establishing market leadership can be achieved through delivering at lower cost or differentiating based on high quality. Additionally it is recognized that nowadays the devices or products that first meet the ever changing demands of consumers will have the competitive advantage [1].

The mobile industry can be defined as one of the most rapidly growing and dynamic IT industries. In the mobile industry, the user requirements are changing frequently. This is the main reason why project managers are looking for flexible software process models that can adjust to the frequently changing system requirements and produce valuable software in short time and with low budget.

The purpose of this paper is to propose a continuous requirements engineering method for mobile application development projects. The mobile application industry is addressed here due to the first author's work experience in this field. It was observed in practice, that this rapidly growing mobile application industry encounters with badly structured engineering process, which can pose severe consequences on software development process and product success. The focus is on both design-time

and run-time issues. In this paper, we consider design-time as the time when the user interfaces are created and actual application development process is performed; and the run-time – as the time when the application is actually running or released for using.

The requirements engineering concerns the lifecycle of the application development. The lifecycle of mobile application development, in general, is the same as the software development lifecycle for web or desktop applications where there are many different lifecycle models defined, for instance, waterfall model, spiral model, prototyping model, agile models and other models. Each of these models is usually described by a particular sequence of activities [2,3].

The model selection for developing software depends on the software project characteristics. To choose or develop the best suited process model for mobile application development project, the following main characteristics of mobile applications should be taken into account [4]:

1. User requirements are changing with time; this change is frequent.
2. Development time is short.
3. Emphasis is more on user interfaces.

In many aspects, developing mobile applications is similar to software engineering for embedded applications. Common issues include integration with device hardware, as well as security, performance, reliability, and storage limitations issues.

The rest of the paper is structured as follows. Section 2 recalls the related works in continuous requirements engineering. Section 3 proposes the continuous requirements engineering method for mobile application development. Section 4 discusses the application of the proposed method in practice. Section 5 concludes the paper.

2 Related Work

The formulation and implementation of the continuous requirements engineering process, particularly, for self-adaptive systems, are discussed by a Nauman A. Qureshi and Anna Perini. They both together and separately have looked at challenges in the engineering of self-adaptive software and focused on requirements engineering issues with two-fold, long term objectives. The first objective is to support the system analyst to engineer adaptive requirements at requirements-time; the second objective is to make software able to reason on requirements at run-time in order to enable a goal-oriented adaptation [5]. The conducted research tries to look in-depth into requirements specification problem in terms of a planning problem, aiming at specifying what a developed software system should do to meet user goals, quality constraints, and preferences under a given set of domain assumptions. This is done by, firstly, analyzing variability level in the application domain and the operational context. Secondly, the user goals and preferences should be identified, to define alternative software behaviors that may take place in different context conditions.

Further, in Service-Based Applications (SBA), open systems, which rest on a global infrastructure (i.e., the Internet), there is a need for managing the heterogeneity of

service providers and the variability of contexts and devices SBAs can be accessed from, still keeping expected functionalities and qualities of the applications. The main feature, which enables SBAs to continuously change themselves, is self-additivity. This characteristic helps to manage the uncertainty, dynamism, heterogeneity, and variability of SBA operating environment in design-time [6,7,8].

In general, requirements engineering activities allow systematically building a specification for the system-to-be-developed that is able to satisfy the goals of its planned users. In the context of open and adaptive systems, e.g., SBAs, the user goals or, in other situations, priorities may change at run-time. Also the context conditions, in which the goals should be achieved, can change. Therefore, the need of perceiving requirements engineering activities at run-time [6,7,8] arises.

Requirements engineering can be linked to the development with an interface which operates between the system and its environment; it collects from the environment information or domain models for the system to be developed. In the context of requirements engineering, sharing information between these processes and artifacts is handled through different activities – manual activities (for instance, feasibility study, requirements elicitation methods, etc.) or semi-automatic activities (for instance, requirements validation, requirements management, etc.). The information sources often share cross-cutting concerns, which are captured through traceability links between them. Traceability can be performed from initial phase artifacts to later ones – forward traceability; or from later phase artifacts to initial ones – backward traceability [9]. The model federation can be used to connect these artifacts and information sources to allow continuous and uninterrupted requirements engineering process [10,11].

Recently Golra and Beugnard et al. [10,11] have proposed a model federation definition – "it is an approach that provides the means to integrate multiple models confirming to different paradigms". This allows to develop new cross-concern viewpoints/models or to synchronize the models used for designing a system.

The ideas of above discussed sources have helped to develop the method presented in the next section.

3 Proposed Continuous Requirements Method for Mobile Application Development Projects

Based on model federation guidelines [10,11], in the proposed method we separate concerns by three types of modeling spaces, which are *generic*, *technical*, and *design spaces*, as shown in Fig. 1. These three different modeling spaces of the proposed method serve together for the use for the continuous development projects that utilize the continuous requirements engineering approach. The proposed continuous requirements engineering method is supposed to be applied in agile systems development context.

In Fig. 1 the generic space is where new models or views are developed by taking into consideration the concepts from already existing models. These already existing models are called Epics (large user stories). As Epics reuse the concepts from various

models conforming to different paradigms, the conceptual space is surrounded by multiple technical spaces. Each of these technical spaces is a collection of models standing for a common paradigm which gives a common ground for their interpretation. Usually, each technical model is developed to serve some specific concerns; they might even influence or somehow affect some different projects or systems. The functional space is a specific kind of technical spaces that serves for functional and design representation setting down the acceptance criteria.

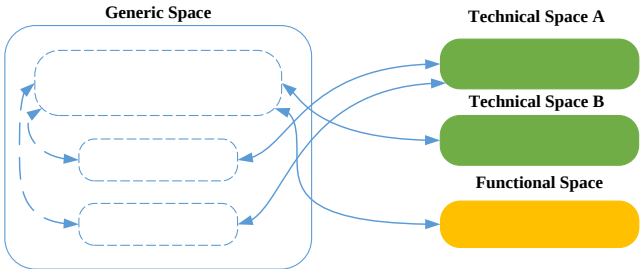

Fig. 1. Mapping between Modeling Spaces (each space is depicted by specific colour: generic – blue, technical – green, and functional – yellow)

Model federation discussed in Section 2, is realized here through virtual models to develop new concepts. The authors have adjusted the model federation framework [10,11], so it better fits for mobile application projects and agile software development approaches.

The generic part federating the concepts from already existing models cannot access the elements of a technical model unless it can interpret the formalisms used in its technical space. This can be done using a connection between the technology space and the generic space, called as connector. These connectors allow access for reading, editing, and synchronizing the information between the previously developed generic models and the technical models. When the model is developed it can be serialized into a new or existing technical space for further development. Apart from linking the concepts of different models, model federation allows to maintain those dynamic links. These models used in the requirements engineering process remain connected to the requirements model, so updating of the specific requirements is possible as soon as the information resources are modified.

Usually each stakeholder in the requirements engineering process has a different viewpoint on a developed system. Each of these observation points correspond to a subset of concerns related to the complete software development process. These various viewpoints often are created as different artifacts, e.g., requirements specification document, system models, goal models, etc.

From agile systems development perspective, mobile application requirements can also come from different stakeholder groups. Traditional Product Owner rather focuses on the implementation of functional requirements; in this case technical aspects of process operations might be neglected. Therefore, some demands on the requirements

management also emerge in agile methods to ensure that the expectations of all stakeholders are considered.

The application development process starts with defining the objectives of mobile application – why the development team is brought together, why the project is started – the basic goal the work is aimed towards. This step is followed by stakeholder identification; this step includes analysis of stakeholder "stakes": what stakeholders want from the system, the constraints they impose; how the project will create value for them and more. When the team is established, the next common step is to create a vision. The vision both expands on and meets the objective set in the first step. For instance, if the objective specifies a problem that needs solving, the vision gives an answer. This step is pursued by role distribution; these roles narrow down the stakeholder base to consider those who will actually interact with the system as envisaged by the vision. As the role holders are those who interact with the application, it is important to consider them when determining the functionality. They expand and elaborate certain roles, adding texture to requirements analysts. Because of this information user interface design specialists and software developers can better understand and focus towards those who will use the application. After this step the requirements engineering process can be started (Fig. 2).

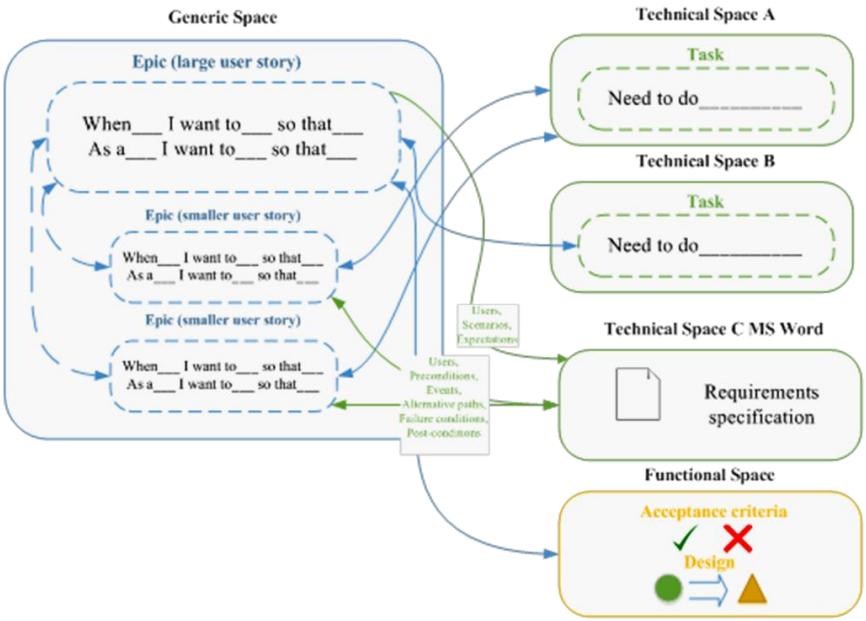

Fig. 2. Model federation for continuous requirements engineering in mobile application development

Fig. 2 describes the process how adapted model federation technique can be implemented in continuous requirements engineering process for agile mobile application development projects. When the objectives and users are well understood, it is

time to start specifying what the application will do. In models generic space the first Epics are created. An Epic captures a large body of work. It is essentially a large user story that can be broken down into a number of smaller stories. Basically Epics are realized through the link between generic space and technical space which contains requirements – generic requirements are uploaded in the generic space and can be used as the basis for epic development. During the epic and user story development process acceptance criteria also are set down, these are conditions of satisfaction – clear descriptions that define value proposition, user workflows, or characteristics of the solution. The next phase is to break down these epics in smaller user stories. This can be done in several ways. For instance, the epic can be broken down by user interface components; – each workflow required by the user can be a separate user story. Or the epic can be broken down by phases; – first doing a simple implementation and then adding new functionality with each additional user story. When this phase is done, the technical space should be updated with tasks. Tasks are smaller work items that build a story and are devoid of business benefits. Epic (user story) creation and task definition should be represented also in requirements specification; this is possible also through the same links which are between the generic space and the technical spaces.

This approach of model federation can be explained through the analogy of components based design, where models serve as components and technical connectors as connectors. The approach permits connecting more information resources to the requirements specification model also during later stages of software development. When artifacts are developed later or in future iterations, they can be easily linked with existing requirements. This support for linking new information resources to the requirements and maintaining them for synchronization allows carrying out the continuous requirements engineering process all along the software development lifecycle, regardless of phases and iterations when the new requirements appear.

To understand better the proposed method, it should be clarified how it changes the traditional requirements engineering approach. Fig. 3 illustrates the activities involved in the proposed method. In this process all basic requirements engineering activities are covered. The difference is in the way the activities are related and organized according to the model federation. The continuous requirements engineering method is explained through the following steps:

1. *Gather information sources.* This activity starts with information source identification that can be used to elicit requirements. These resources may vary from early information resources, for example, feasibility reports, standards, minutes or notes from meetings, to late information resources like test plans, involved data analytics, etc.
2. *Organize information sources*. When the information resources are collected they need to be linked with existing requirements. This process is based on model federation approach where the requirements are developed as models in generic space and other models are placed in their respective technology spaces.
3. *Elicit requirements*. Once the information resources are linked with the requirements model, the requirements can be elicited from these sources.

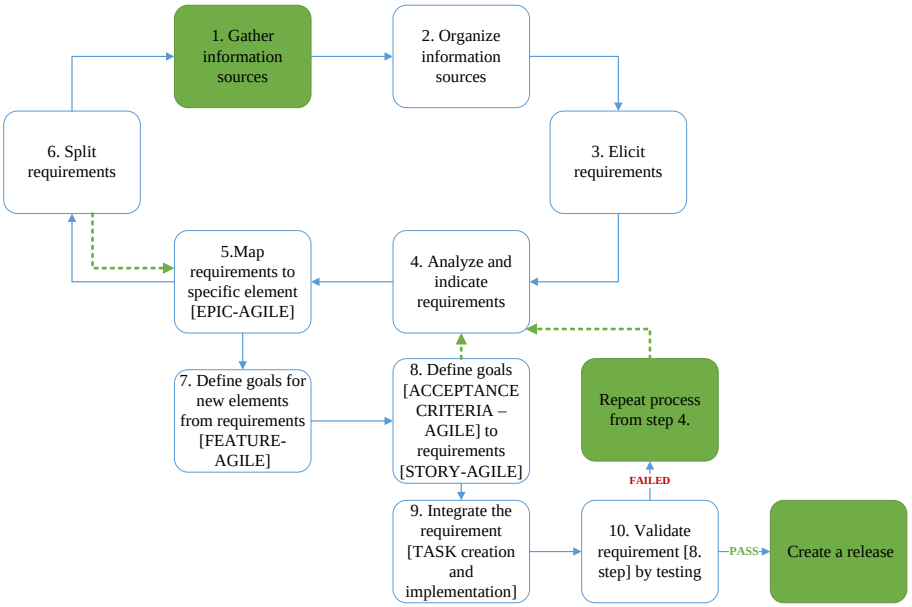

Fig. 3. Requirements Engineering Process with Model Federation

4. *Analyze and indicate requirements*. When the requirements are identified from multiple information sources, they are analyzed and in situation when the particular requirement is already specified the additional description can be added.
5. *Map requirements to specific elements*. In this activity, the specified requirements are mapped to the major project elements (smaller user stories to large user story). Several requirements can be mapped to a single project element.
6. *Split requirements*. Requirements mapped to a project element describe the expected functionality from that element. The decomposition of the major element can be done – each sub-element realizes a subset of the requirements mapped to its parent element.
7. *Define goals for new elements from requirements*. The goals for all new requirements associated with a project element should be defined.
8. *Define goals to requirements*. Create tasks and acceptance criteria for all requirements.
9. *Integrate the requirement*. The task integration process is carried out for each requirement.
10. *Validate requirements*. The acceptance criteria are verified by testing each resolved task.

The above-described method addresses the basic characteristics of mobile application development mentioned in the Introduction of this paper. It also well supports one more characteristic of mobile application development, namely, long maintenance periods of the applications. The method was tested on a practical case which is described in Section 4.

4 Application of the Proposed Method

To show how the proposed method can ensure continuous requirements engineering in practice, the mobile Android application development project "Meters" is discussed. The project idea belongs to the first author of the paper, and she was involved in the whole project lifecycle as a member of the team.

To apply the proposed method for project lifecycle two elements are to be prepared – the tool which will help to apply the method for particular project and the first draft of collected requirements. Visual Studio Team Services (TS) was chosen to be the tool for project lifecycle management experiment. This tool provides the work item tracking which is needed to show the proposed method in real project.

First step is to gather information sources (Step 1). For the experiment needs the requirements are used from "Meters" applications welcome screen which appears when the application is launched for the first time. This information should be added to methods generic space or, in other words, in TS the Epic "Application setup for the 1st step of work" is created. By creating this Epic, Step 2 "Organize information sources" has also been carried out.

As this Epic description contains many smaller requirements, the Step 3 "Elicit requirements" and Step 4 "Analyze and indicate requirements" are carried out. From TS perspective several user stories, which divide current requirement in smaller elements, are created under the established Epic. The template for user story consists of various fields. One of the main fields is description where all relevant information about the screen is provided. Filling field acceptance criteria fulfills the method's Step 8 "Define goals to requirements". The acceptance criteria define when and under what conditions the screen is shown to users and what the exceptions are. Afterwards, the acceptance criteria description is used to create the test cases based on which the requirements validation process is carried out – Step 10.

The user story can be created in two ways – one way is to create a new work item and chose the user story template for it; another way is to use SmartWord4TFS. This is a Word extension which facilitates Word integration with TFS, enabling users to bi-directionally synchronize Word documents with the work item information stored in TS.

When the process is carried out till this point, the Step 8 "Define goals to requirements" can be fulfilled and tasks are created. Afterwards the development team implements each task and the tasks which are resolved can be tested or implemented requirement can be validated – Step 10.

The method ensures the requirements engineering continuity through 3 repetitive cycles which are highlighted in Fig. 4. The Cycle 1 includes Step 5 and Step 6, the reverse link here is from requirements splitting process to requirements mapping to specific elements. As initially the requirements are usually received in groups or at high-level of abstraction, this cycle provides the possibility to split these requirements in separate parts, so it is easier to continue to work with these parts.

The Cycle 2 consists of Step 4, Step 5, Step 7, and Step 8. The reverse links here are from defining goals to requirements, to analyzing and indication of requirements. These links appear when the new requirement (which comes from Step 6, from split-

ting process) goes through Step 5, then Step 7 and reaches Step 8. The links can be used in situations when it is hard or even impossible to define the objectives of requirements in Step 8 and the process should be repeated to ensure that the requirements have right dependencies and there are no missing links with previously defined requirements.

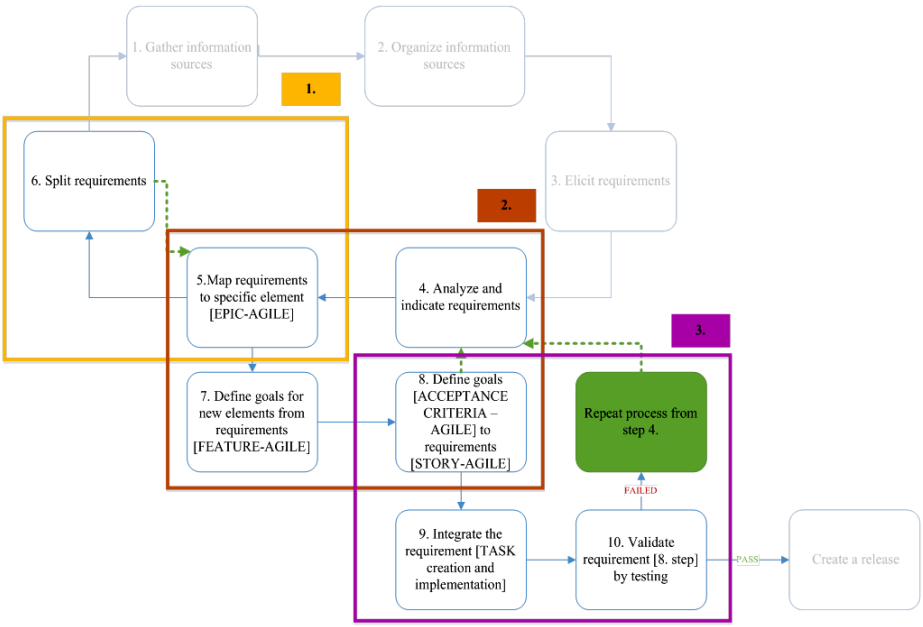

Fig. 4. Continuous requirements engineering method's repetitive cycles

Cycle 3 contains Step 8, Step 9, Step 10, and the result "failed" from requirements validation in Step 10. The scenario "not passed or failed" means that the requirement is not met and should be revised, the work goes back to the step where the requirements are analyzed and indications are carried out – Step 4. It may happen that, while the test cases are examined, a scenario that is not covered by the requirements can be identified. In this situation the process continues also from the Step 4.

5 Conclusion

In mobile application development projects usually agile approaches are used, however, these methods not always give a possibility to handle requirements transparently and ensure their traceability. Taking into consideration such features of mobile application development as frequent changes of user requirements, short time for application development and strong emphasis on user interfaces; the continuous requirements engineering method for mobile application development is proposed in this paper. The method separates 3 concerns: generic, technical, and design concern. It ensures

transparent traceability and handling of each requirement with respect to each of the concerns. The method has been successfully applied in practice.

The limitations of this research are that (1) the method has been tested in the development situation for one only mobile operating system and (2) the applicability of the method for other software application domains has not been investigated.

Further research will concern overcoming of mentioned limitations, investigation of possibilities of incorporating more run-time requirements engineering aspects in the method, and definition of the most appropriate tool support for the method.

Acknowledgement. We acknowledge CRE'17 anonymous reviewers and Prof. Uldis Sukovskis for valuable comments on this research.

References

1. IBM Software: The Competitive Advantage of Continuous Engineering. New York, United States of America: IBM Corporation Software Group, 16 p. (2014)
2. Vithani, T., Kumar, A.: Modeling the Mobile Application Development Lifecycle. In: Proceeding of the International MultiConference of Engineers and Computer Scientists 2014 (IMECS 2014), pp. 596–600 (2014)
3. Wasserman, A.I.: Software Engineering Issues for Mobile Application Development. In: Proceedings of the FSE/SDP workshop on Future of software engineering research (FoSER'10), pp. 397–400 (2010)
4. Ashishdeep, A., Bhatia, J., Varma (Thakkar), K.: Software Process Models for Mobile Application Development: A Review, IJCSC International Journal of computer Science & Communication, vol. 7, no. 1, pp. 150–153 (2015)
5. Qureshi, N.A., Perini, A.: Engineering Adaptive Requirements. In: Workshop on Software engineering for adaptive and self-managing systems (SEAMS'09), pp. 126–131 (2009)
6. Qureshi, A.N., Perini, A.: Requirements Engineering for Adaptive Service Based Applications. In: 18th IEEE International Requirements Engineering Conference, pp. 108–111 (2010)
7. Qureshi, N., Perini, A., Ernst, N.: Towards a Continuous Requirements Engineering Framework for Self-Adaptive Systems. In: Proceedings of the 2010 First International Workshop on Requirements@Run.Time (RE@RunTime), pp. 9–16 (2010)
8. Qureshi, N., Perini, A.: Continuous Adaptive Requirements Engineering: An Architecture for Self-Adaptive Service-Based Applications. In: Proceedings of the 2010 First International Workshop on Requirements@Run.Time (RE@RunTime), pp. 17–24 (2010)
9. Pinheiro, F.A.C.: Requirements Traceability. In: Perspectives on Software Requirements, pp. 91–113. Boston, MA: Springer (2004)
10. Golra, F.R., Beugnard, A., Dagnat, F., Guerin, S., Guychard, C.: Continuous Requirements Engineering Using Model Federation. In: Proceedings of the 24th International Requirements Engineering Conference, pp. 347–352 (2016)
11. Golra, F.R., Beugnard, A., Dagnat, F., Guerin, S., Guychard, C.: Addressing Modularity for Heterogenous Multi-Modal Systems Using Model Federation. In: Companion Proceedings of the 15th International Conference on Modularity, pp. 206–211 (2016)