SPLICE: A Lightweight Software Product Line Development Process for Small and Medium Size Projects

Tassio Vale*, Bruno Cabral†, Loreno Alvim†, Larissa Soares†, Alcemir Santos†, Ivan Machado†, Iuri Souza†, Ivonei Freitas‡ and Eduardo Almeida†
*Center of Exact Sciences and Technology, Federal University of Recôncavo da Bahia (UFRB), Cruz das Almas - Brazil
Email: tassio.vale@ufrb.edu.br
†Computer Science Department, Federal University of Bahia (UFBA), Salvador - Brazil
Email: {iuri.souza,loreno.alvim}@ufba.br, {bruno.cabral,larissars,alcemirsantos,ivanmachado,esa}@dcc.ufba.br
‡State University of West Paraná (UNIOESTE), Cascavel - Brazil
Email: ivonei.silva@unioeste.br

***Abstract*—Combining Software Product Line Engineering (SPLE) and Agile Software Development (ASD) is an approach for companies working with similar systems in scenarios of volatile requirements aiming to address fast changes and a systematic variability management. However, a development process covering the whole SPLE lifecycle and using agile practices in small and medium size development projects has not been established yet. There is a need to disseminate such combination through well-defined roles, activities, tasks and artifacts. This paper presents SPLICE, a lightweight development process combining SPLE and agile practices, following reactive and extractive approaches to build similar systems. SPLICE addresses the needs of small development teams aiming to adopt SPL practices with low upfront costs and fast return on investment. In order to evaluate our proposal, we report our experience in a case study by developing RescueMeSPL, a product line for mobile applications that assists users in emergency situations. The case study results point SPLICE achieves the three evaluated aspects by providing short and proper iterations, possibilities for activities adaptations and continous feedback.**

I. Introduction

Software Product Line Engineering (SPLE) exploits commonalities and variabilities of similar software systems in a systematic way to decrease development costs, increase productivity and improve product quality. It is comprised of a process involving domain and application engineering [1]. In general, SPLE requires a significant up-front commitment in developing a flexible product platform [2]. On the other hand, in a dynamic market environment involving time-to-market constraints, companies need agility to handle changes [2]. Agile Software Development (ASD) focuses on working code while reducing up-front design and process overhead of traditional approaches [3], through the adoption of fundamental values and principles [4].

Combining SPLE and ASD is an approach for companies working with similar systems in scenarios of volatile requirements aiming to address fast changes and a systematic variability management. For small and medium companies, achieving the balance between the speed offered by ASD and systematic software reuse is crucial [5], considering the limited availability of resources for such an adoption. In addition, SPLE and ASD present different practices, therefore, this combination must be handled carefully.

According to Diaz *et al.* [6] the objectives of combining SPLE and agile methods are manifold: *(i)* cutting down the long-term investment during the domain engineering phase; *(ii)* dealing with volatile business situations; and *(iii)* dealing with situations in which there is not enough knowledge on domain engineering.

To the best of our knowledge, a development process covering the whole SPLE lifecycle and using agile practices in small and medium projects has not been established yet. In fact, this research field is still challenging and requires additional research [6]. For instance, addressing the traceability of product variations in different abstraction levels (*e.g.* feature models, requirements, design, code and tests) considering agile practices to reduce the impact of changes in volatile domains by improving the team responsiveness is not a trivial task [7].

Therefore, this paper presents a lightweight development process combining SPLE and Scrum agile practices. The process follows the reactive and extractive approaches to build the similar systems, and it addresses the needs of small development teams aiming to adopt SPL practices with low upfront costs and fast return on investment. In addition, we report our experience on developing a product line for mobile applications that assists users in emergencies, aiming to evaluate the proposed process.

We organized the remaining sections of this paper as follows. Section II discusses related work. Section III describes the activities, tasks, roles and artifacts of the proposed process. Section IV presents a case study to evaluate the process through the RescueMe SPL project. Then, Section V discusses threats to the validity of this work. Finally, Section VI summarizes the study findings and presents future work.

II. Related Work

Several work addresses the SPL development process matter, some of them follow an unsystematic approach [8], [9] and others tried a more formal approach [10], [2]. In either case,

they left some aspects uncovered, such as, both reactive and extractive SPL approaches.

Paige *et al.* [8] applied Feature-Driven Development (FDD) directly to building a microwave oven software product line. They encountered two difficulties in applying FDD to build SPLs: integrating SPL architecture design into FDD; and incorporating component development in FDD. As a result, they proposed an extension to FDD by adding two new phases: one for consideration of architecture and one for SPL component design.

Kircher and Hofman [9] approach merges SPL and ASD into a holistic approach by using the concept of "feature orientation". Their approach considers an adaptation of the Scrum to feature orientation. They report how to adapt backlog-driven development, business-driven platform evolution to self-organizing feature-oriented Scrum teams.

Hanssen *et al.* [10] describe a process involving multiple disciplines, such as product planning, knowledge management, organizational aspects, and innovation. They present a case study that has successfully integrated SPL and agile practices. The proposed process follows a proactive approach. In this sense, specific disciplines for reactive and extractive approaches were not investigated in this work.

Mohan *et al.* [2] discuss a reactive SPLE approach that embodies agile principles to dynamic markets. They evaluated the approach in an organization which developed the initial set of systems and then refactored them to define commonality and variability, instead of using considerable up-front planning to develop core-assets for the product line. Although comprehensive practices emerged from this study, the authors do not present any systematic process to guide stakeholders in the SPL adoption.

Our proposal intends to provide a detailed description of a process to develop software product lines in both reactive and extractive scenarios. As consequence, stakeholders can take advantages of it to apply precisely a lightweight SPL process in their environment.

III. The Process

A software process consists of activities with related artifacts, human and computer aid resources, organizational structures and constraints, intended to produce and maintain the requested software deliverables [11]. Software processes document development practices that are empirically known to have an impact on software development time, cost, or quality [12]. In addition, they foster stability, control and organization for activities that, without the proper control, tend to be chaotic. In contrast, processes should offer flexibility to be adapted according to environment changes.

We propose the *Software Product Line Integrated Construction Environment* (SPLICE) process[1], that combines SPL and agile practices into a set of systematic activities to be adopted by companies with resource restrictions and volatile domains. We adopted the Scrum agile practices, since they provide a systematic way to improve feedbacks, iterativeness, and adaptability in volatile domains. We rely on the Software Process Engineering Metamodel (SPEM) [13] to describe the process roles, tasks and work products (artifacts or assets).

[1]SPLICE process specification is available at http://tassiovale.com/splice

Roles

This process defines a set of ten roles: *Business Expert*, *Scope Owner*, *Product Expert*, *Scrum Master*, *Legacy Systems Developer*, *Inspector*, *SPL Developer*, *SPL Tester* and *Scrum Team*. Table I describes the responsibilities of each role.

Business Expert: aligns business goals, prioritizes major features, and selects business goals as well as market strategies;
Scope Owner: represents the user, client, sponsor, business analyst, or any mix of these, having a deep knowledge about the domain; he also prioritizes the major features and explain details about them;
Product Expert:: has a deep knowledge about the business and technical value involving the features in one or more existing products;
SPL Expert: is the SPLICE process expert, leading stakeholders to perform the activities as described, and verifying the accuracy of the resulting work products;
Scrum Master: supports participants in the agile principles and practices;
Legacy Systems Developer: represents the developers, testers, or debuggers of the legacy code;
Inspector: performs inspection in the sub-feature list, feature model, and product map assets;
SPL Developer: responsible for the feature implementation tasks in the product line;
SPL Tester: develops and performs tests in the software product line; and
Scrum Team: represents a group of stakeholders with skills on ASD that perform sprint tasks.

TABLE I. SPLICE ROLES RESPONSABILITIES.

Activities, Tasks and Work Products

The SPLICE process activities, tasks and resulting work products are distributed in two phases: *Portfolio Planning* and *Release Development*.

A. Portfolio Planning

The *Portfolio Planning* phase aims to provide a high-level description of the SPL business domain. In this description, four aspects are identified: business goals, marketing strategies, products and major features. The participants achieve it by performing five tasks: *Select business goals and marketing strategies*, *Identify products*, *Identify major features*, *Build product map and feature model* and *Prioritize major features*. Figure 1 shows a workflow of this phase. In addition to an overview about the business domain and its strategies, the resulting assets allow stakeholders to perform a comprehensive planning for development releases and their related sprints.

1) Select business goals and marketing strategies: analyzes the market regarding its business goals, user profile, legal and cultural constraints, business opportunities, competitors, and other factors defined by the *Business Expert* and *Scope*

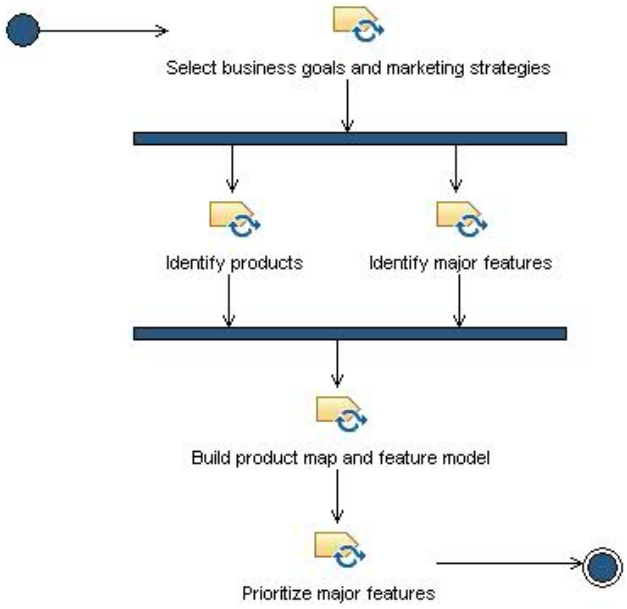

Fig. 1. SPLICE Portfolio Planning.

Owner. Marketing strategies define how the products will be released to the customers considering the sale units, integration methods, installation, product maintenance, and user skill level. On the other hand, business goals define information about customer needs, technological environment as well as the legal and cultural constraints (e.g. country laws, organizational workflows). As output, the *Market Analysis Document* contains a simple list of marketing strategies and business goals. The goals should be prioritized and every decision making during the development process must consider them.

2) Identify products: identifies the products available in the company portfolio that share business activities in common. This task involves the *Scope Owner* and the *Product Expert*. It results in a list of products to support the construction of product map and feature model.

3) Identify major features: identifies coarse-grained features and provides a description of them. Kang et al. [14] define feature as "*a prominent user visible aspect, quality, or characteristic of a software system or systems*". Model storming, investigation of main functionalities in the existing products portfolio and activity diagram modeling can help the *Scope Owner* and the *Product Expert* to identify major features. A list of candidate major features is the result of this task. Before building the product map, there might be fine-grained features in the list. Since the product map should contain only major features at the first time, this task intends to group the fine-grained features into major features. The *Scope Owner*, *Product Expert* and *Legacy Systems Developer* participate in this task. As result, it balances the granularity of the major features, providing an organized domain model. *Identify products* and *Identify major features* can be performed in parallel, since they can support decisions of each other.

4) Build initial product map: consists of tasks to build a matrix comprising features and products, indicating, for each product, which features it implements. *Scope Owner* and *Product Expert* can provide support for this task, if needed. As result, there is a common understanding on which features are present in the products and how these products are related among them in the domain.

5) Prioritize major features: this activity prioritizes the major features using classification criteria relevant for reuse. It facilitates choosing major features for the releases and sprints. An input for the prioritization is the Domain Potential Assessment [15], performed by using a set of criteria: experience, risks, volatility, maturity, resource/organizational constraints, available legacy artifacts potential, market potential, reuse potential, coupling/cohesion, and commonality/variability. The *SPL Expert* applies a questionnaire [15] with these criteria for the stakeholders (*Scope Owner* and *Product Expert*).

The questionnaire results are analyzed by the *Business Expert*, *Scope Owner* and *Product Expert* to define the final ranking of the major features. During this process, the stakeholders can select only a set of assessment criteria from the complete list, according to the importance for their scenario. This ranking generates the *Scope Backlog*, a list of major features ordered by importance considering both technical and non-technical aspects.

B. Release Development

After specifying assets with high-level abstraction (*Market Analysis Document*, *Product Map*, *Scope Backlog*) in the *Portfolio Planning* phase, the *Scrum Team* plan and execute activities related to the major features in order to deliver functional products at the end of each release. The *Release Development* phase (Figure 2) consists of the *Release planning* task as well as an iterative set of activities and tasks, *Sprint development*.

Fig. 2. SPLICE Release Development.

1) Release planning: comprises of estimating the effort to detail, implement, test, and perform derivation of the major features in the *Scope Backlog*. In this task, the *Scrum Team* uses a relative unit for size which encompasses effort, complexity and risk, called feature points in this context. The participants can use the Fibonnaci scale or Planning Poker to determine such feature points.

2) Sprint development: The backlog items (major features) for each release will be accomplished in a set of sprints. Figure 3 shows the workflow of activities and related tasks for each sprint. The release development iterations comprise three tasks: *Sprint planning*, *Sprint review* and *Sprint Retrospective*; and five activities: *Sub-features definition*, *Commonality and variability analysis*, *Sub-features implementation*, *Sub-features testing* and *Products derivation*.

Sprint planning

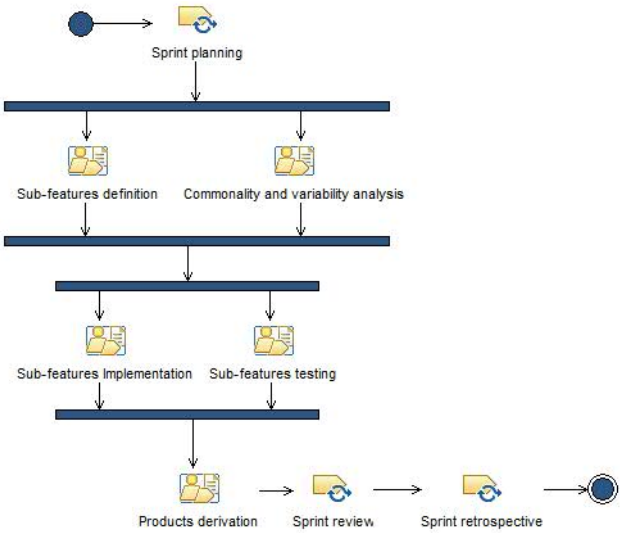

Fig. 3. SPLICE Sprint Development.

In this task, the *Scrum Team* members select major features from the *Scope Backlog* to include them in the *Sprint Backlog*. They verify the amount of major features that can be developed in one sprint, by comparing the feature capacity and total effort of the selected major features, both indicators expressed in feature points. In conjunction with the major features selection, participants determine the goal of the sprint, the tasks to develop each major feature and the assignments for the *Scrum Team* members.

Sub-features definition

Sub-features express details of major features. This activity provides descriptions and properties of the sub-features for each major feature in the domain. This activity comprises four tasks: *Identify Sub-Features*, *Mine Legacy Assets*, *Sub-Features Inspection* and *Sub-Features Validation*.

Identify sub-features. In this task, the *Domain Analysis Expert* captures reusable sub-features from the *Scope Owner* or *Product Expert* through workshops, where they provide information in a timely manner. The stakeholders interact to define a sub-features list following a specific template.

Mine Legacy Assets. This task involves the steps to learn from existing products, considering that experts do not know all details of the major features to provide it in a timely manner. The *Legacy Systems Developer* can help to understand the legacy systems. It consists of studying the system-as-is, their flows, work procedures, business rules, reports about defects, user manuals, screenshots of the products, and prototypes. These steps can be done in conjunction with the *Identify sub-features* task. Mining is important to save time of the *Scope Owner* because many features can be found in documents such as requirements, user manuals, design, algorithms, or test cases. This task results in possible improvements of the features list.

Inspect sub-features. It aims to consolidate and verify the sub-features list from the previous tasks. The *Inspector* inspects the features list in terms of non-conformities that consider aspects such as feature granularity, understanding, and duplication. In the traditional inspection activities, documents are sent to the reviewers before the inspection meeting, and the identified non-conformities are consolidated in the same meeting. We propose a lightweight inspection task to optimize the inspection effort and anticipate the specification issues. In this task, the finished specification is immediately submitted for inspection and correction in a meeting. This meeting involves two reviewers, one moderator and the specification authors. As result, the stakeholders should modify the features list according to the non-conformities identified from the features specification.

Validate sub-features. This task aims to reach agreements among stakeholders on the sub-features specification. The *SPL Expert* collects opinions from the *Product Expert* and *Scope Owner* to adapt the features specification whether necessary, and integrate them with the other available features. This task provides a consolidated list of major features and respective sub-features for the next activity.

Commonality and Variability Analysis

This activity specifies the commonalities and variabilities through feature model and product map work products. It defines which features are directly linked (father-son) and how they are classified (mandatory, optional, and alternative). It comprises three tasks: *Update product map*, *Update feature model* and *Inspect models*.

Update product map. This task updates the product map created during the *Portfolio Planning*. It takes the list of major features and sub-features to determine which ones are present in the domain products. The feature present in a product is "checked'" in the intersection between feature and the respective product.

Build feature model. It organizes the feature model based on the features list, commonalities and variabilities provided by the *Scope Owner*, *Product Expert* or *Legacy Assets Developer*. As result, this task structures and categorizes features hierarchically in the *Feature Model* [14]. The commonalities and variabilities (with variation points) can be also found in the legacy systems. In this case, the *Legacy Systems Developer* should support the *Scrum Team*.

Inspect models. Based on Souza et al. [16], this task verifies the *Product Map* and *Feature Model* inconsistencies. Stakeholders provide the criteria for inspection, such as: presence of dead or zombie features, non-conformities in the feature model, and other logical errors defined by the *Scrum Team*. The *SPL Expert*, *Scope Owner*, and *Legacy Systems Developer* should participate in this task leaded by the Inspector.

Sub-features implementation

After the *Sprint planning*, the *SPL Developers* have a backlog of tasks to perform. During the Sprint, they must focus on the specified tasks. At the first time, *SPL Developers* should choose the most suitable variability implementation technique (or a set of techniques) to build the SPL. Then, they will implement the sub-features and test cases simultaneously, following the Test-Driven Development (TDD) practice [17]. The goal of this activity is to deliver a high-quality *SPL source code*.

Sub-features testing

In this activity, *SPL Testers* work in pairs to design and implement the test cases, as a means to improve quality of the produced artifacts. A series of regular meetings take place to promote discussions among *SPL Testers*, *SPL Developers*, *SPL Experts* and *Scope Owner*. Three main steps surrounding the test case development: *initial meetings*, *problem solving meetings*, and *acceptance meetings*.

Initial meeting. As the requirements team either design a new functionality or propose any change and/or improvement in existing artifacts, a new request raises. Hence, both testing and requirements teams work together, aiming to understand and clarify the requests, and to identify macro testing goals, as a means to split them into small tasks. Hence, as tasks are split, testers have a set of functionalities to test.

Problem solving meetings. These meetings were regularly held to explore the expertise of other testers in the team. Whenever any impediment was raise by a tester, a problem solving meeting serve as the proper environment to discuss likely strategies to help handling the issue.

Acceptance meeting. As *SPL Testers* complete their assigned tasks on a regular basis, the team met in a so-called acceptance meeting to discuss the accomplishments. In case of all, or mostly, test cases, were accepted, i.e., reached the desired code and test coverage level, testers were assigned to new testing tasks, with new functionalities. Conversely, if the team agreed that a desired coverage level was not achieved yet, the test manager suggested another test case development iteration on the same set of functionalities.

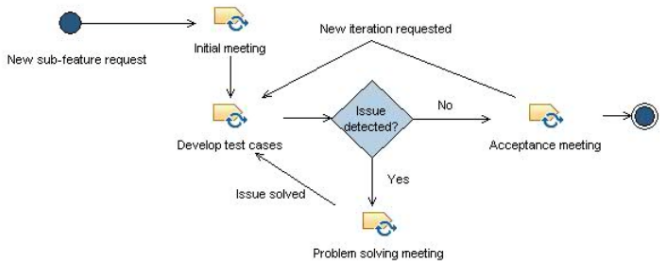

Fig. 4. Testing process activities.

Products derivation

This activity selects and customizes shared assets during application engineering of a product line. Based on variability provided by product line is possible to select a particular product and adapts for a particular purpose [18]. The main goal of derivation is to generate products to customers and bring return on initial investment by exploiting the commonality and variability established in domain engineering [19].

Sprint Review and Retrospective

In the *Sprint Review* meeting, the *Scrum Team* shows what they accomplished during the sprint. Typically this takes the form of a demo of the new sub-features. This meeting is kept very informal, allowing no more than two hours of discussion among stakeholders. During the review, the SPL is assessed against the sprint goal determined during the *Sprint Planning*.

The *Sprint Retrospective* meeting is the last task of a sprint, performed immediately after the sprint review. The entire *Scrum Team* should participate in this meeting. It can take, in average, one hour. In order to conduct an effective retrospective, the *Scrum Team* members answer three questions: "*What will the team keep doing?*", "*What will the team stop doing?*" and "*What should be improved?*".

Support tool

Aiming to support the defined process and due to the absence of a tool to be adapted for our scenario, we are developing SPLICE[2], a web-based SPL life-cycle management tool, providing traceability and variability management and supporting most of the SPL process activities such as scoping, testing, version control, evolution, management and agile practices.

According to Cabral *et al.* [20], the tool enables the *Scrum Team* to automatize assets creation and maintenance, providing traceability and variability management as well as offering detailed reports that allows a smooth navigation between the assets through traceability links. It also provides a basic infrastructure for development, and a centralized point for user management. Next, Table II presents the main SPLICE functionalities.

Agile planning: support for a set of agile practices, such as effort estimation, where team members use effort and degree of difficulty to estimate their own work.
Traceability: the trace links, expressed by features, provides traceability for all assets in the metamodel, and is able to report direct and indirect relations between them.
Reports: support for creating reports automatically including the set of the chosen assets related to the selected product. This format is suitable, for instance, to validate requirements by stakeholders. The tool is also able to compress all reports for a given product (*e.g.,* features report and user stories report) in a single file.
Issue Tracking: full-featured issue tracking support based on trac core. We extended it to implement SPL specific features and to provide traceability between other assets.
Custom SPL widgets: custom widgets to represent specific SPL models, such as Feature Map, Feature restriction, Product Map, and Agile Poker planning.
Change history and timeline: support for visualization of the project evolution, where the changes are happening, and who did it.
Unified control panel: it aggregates the configuration of all external tools in a unified interface, *i.e.* an unique credential to access all SPLICE features and external tools, such as version control systems.

TABLE II. SPLICE FUNCIONALITIES

IV. CASE STUDY

The process was evaluated qualitatively, through the case study technique. Case studies investigate a contemporary phenomenon within its real-life context, describing and analyzing a bounded system (or case) [21]. Details of the case study research protocol and the respective results are presented next.

[2]A demonstration video is available at http://bit.ly/1scwGpH

A. Research Questions

We identified three key aspects [7] for lightweight software development processes: continuous feedback, iterativeness, and adaptability. The traditional approaches do not provide systematic mechanisms to achieve these aspects through short iterations, and foster the continuous feedbacks and adaptations in an SPL development process. Thus, three research questions were defined to evaluate them:

How do the stakeholders characterize the iterativeness of the process?

Rationale: iterativeness means the potential to optimize the domain building through short iterations in sequence, and each iteration is a self-contained development cycle composed of SPL development tasks such as sub-features identification and sub-features implementation. This aspect fosters earlier changes that can arise from business demands, which improves the process responsiveness in volatile domains and meets the satisfaction of stakeholders.

How do the stakeholders characterize the adaptability of the process?

Rationale: adaptability means the potential to make adjustments in artifacts, team, technology, and process, when necessary. The process should provide mechanisms to allow changes in all elements: activities, tasks, roles and work products. SPLICE has to be adaptable, considering the differences among software development environments in companies.

How do the stakeholders characterize the feedback of the process?

Rationale: feedback involves the communication and collaboration among team members about the process and its results. A continuous feedback can help stakeholders to identify problems earlier and adapt the process to get better performance.

B. Data Collection and Analysis

In order to enable the triangulation of information [21], the sources of evidence adopted in this study were field observation, document analysis, and focus group. Concerning the confidentiality of information, the extracted evidence from participants of the projects are available only to the researcher and participants of the case study.

We performed a participative observation, where the researcher can be actively involved in the work being observed, and shadowing observation, where the researcher follows the participant around and records their activities [22].

Documentation analysis is a technique focused on the documentation generated by the software engineers. We analyzed all the documents generated by the process execution such as sprint plannings, feature model, product map, inspection documents, test cases and source code.

In addition, we performed a focus group with the team developers relying on the following questions:

How do the stakeholders characterize the iterativeness of the process?

How do the stakeholders characterize the adaptability of the process?

How do the stakeholders characterize the feedback of the process?

What challenges and learned lessons emerge in the RescueMe-SPL project?

Comparison, coding, categorization and grouping are the basis of data analysis to answer the defined case study research questions. As result, propositions, assumptions, and insights identified were validated by the researchers and participants of the study.

C. The Case

SPLICE was applied in the development of RescueMeSPL, a set of similar systems of mobile applications. The initial scope of this SPL project was to develop products for the iOS[3] platform applied to smartphone devices, using the Objective-C language. This SPL is based on the Savi application[4], both developed in the RiSE Labs[5]. The development team was comprised of one post-doctoral researcher, five Ph.D. students, two M.Sc. students and two B.Sc. students.

The RescueMe products aim to help its users in emergency and dangerous scenarios. The main screen of all products present a red button, pressed by the users to send messages for all RescueMe contacts in the list. Optionally, contacts can be reached through social networks, and they can track the RescueMe user by checking his updated location in a map.

Fig. 5. RescueMe screenshots.

D. Results

The application of SPLICE in the RescueMeSPL project is a result of three months of effective work on the process activities spread in the *Portfolio Planning* and one release with six sprints. Table III shows the assignments of roles for each participant in the project.

[3]iOS platform - https://www.apple.com/ios/
[4]Savi on Apple Store - https://itunes.apple.com/us/app/savi/id590385285
[5]RiSE Labs - http://rise.com.br/

Parcitipant	Assigned roles
Post-doctoral researcher	*Business Expert*
Ph.D student #1	*Scope Owner*, *Product Expert*, *SPL Expert* and *Scrum Master*
Ph.D student #2	*SPL Expert*
Ph.D student #3	*Inspector*
Ph.D students #4 and #5	*SPL Tester*
Master students #1 and #2	*SPL Developer*
B.Sc. student #1	*SPL Developer*
B.Sc. student #2	*Legacy Systems Developer*

TABLE III. RESCUEMESPL ROLES ASSIGNMENT

For the first phase of the process, *Portfolio Planning*, the *Scrum Team* decided to make an adaptation. The *Sub-features definition* and *Commonality and variability analysis* were performed in conjunction with the other activities of the first phase. Since RescueMeSPL had not a complex domain (considering the number of features), all of these activities could be performed simultaneously in a short period of time. The benefits of this adaptation was to provide concrete specifications of the domain features to make more precise sprint plannings.

As first task, *Select business goals and marketing strategies* resulted in a *Market Analysis Document* containing four different aspects: market strategies, business goals, SPL goals and developer/customer goals. The considerations for each aspect in the document are showed in Figure 6.

The major features were identified by analyzing two sources: the single system previously developed in the RescueMe project and the similar products in the Apple Store[6]. The similar products are: Help Me! [7], RescueMe Now[8], Rescue Button[9], and Red Panic[10]. A set of nine major features emerged: `Access_Control`, `Contact`, `Destination`, `Emergency_Numbers`, `Tracking`, `Location`, `Language`, `User_Info` and `About`.

Before determining the portfolio products, the *Scrum Team* decided to identify the sub-features and analyze the commonality and variability of them. Then, they identified five products using the criteria of incorporating only basic features for simple products and providing more complex features for the other ones. The products are: *RescueMe Lite*, *RescueMe Standard*, *RescueMe Social*, *RescueMe Pro*, *RescueMe Ultimate*. As result, the participants built the product map and feature model work products. Figure 7 and Table IV presents part of the feature model and part of the product map, respectively.

The SPLICE process recommends to perform the sub-features specification immediately after identification. However, the participants decided to identify all domain features and perform the specification afterward, since having an overview of the domain will help to elicit each feature more precisely according to them.

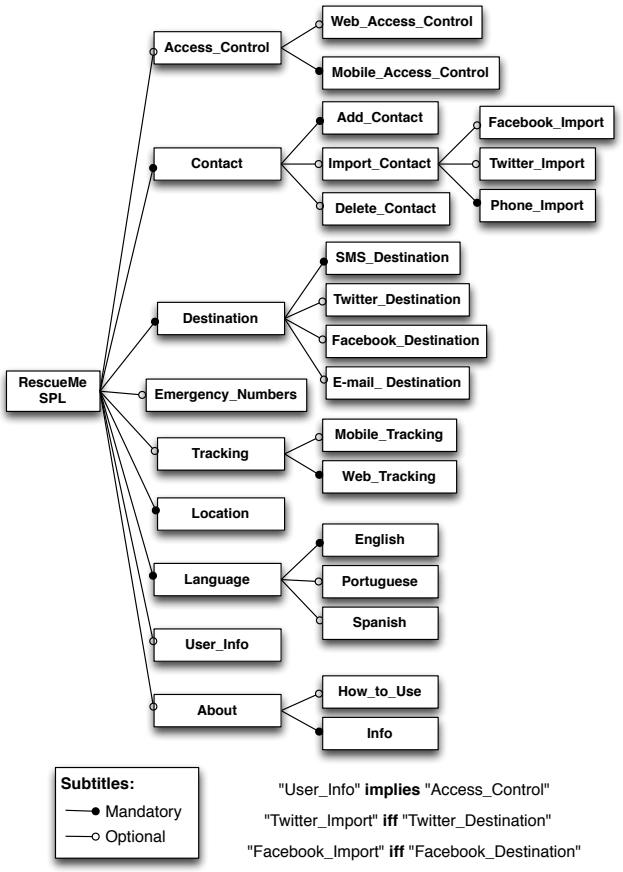

Fig. 7. Part of the RescueMe Feature Model.

Table V shows an example of specifications for the `Contact` major feature.

In order to verify the quality of features specification, feature model and product map, the inspection task reviewed iteratively an amount of twenty-seven features in three meetings of one hour each. The inspection activity identified and removed twenty-eight non-conformities such as incompleteness, incorrectness, ambiguity and unnecessary information [16].

The prioritization was performed in this project after having specifications and models related to the domain features. This activity was adapted by the *Scrum Team* to use criteria suitable for the domain. Since the participants had not expertise on the RescueMe domain, the prioritization criteria was mainly based on developers judgment in getting the simpler features, that ones easier to develop according to their experience and with few dependencies to other features in the feature model. As result, for instance, sub-features related to `Contact` (e.g. `Add_Contact`, `Import_Contact` and `Delete_Contact`) had the higher priority in the domain.

The *Release planning* consisted of selecting major features and sub-features capable of generating products able to be delivered for the users. The eighteen features of the first release were: `Contact`, `Destination`, `Location`, `About`

[6]Apple Store - http://store.apple.com/

[7]Help Me! - https://itunes.apple.com/au/app/help-me/id510561786

[8]RescueMe Now - https://itunes.apple.com/us/app/rescueme-now/id330785171

[9]Rescue Button - http://linskyapps.webs.com/

[10]Red Panic - http://www.redpanicbutton.com/

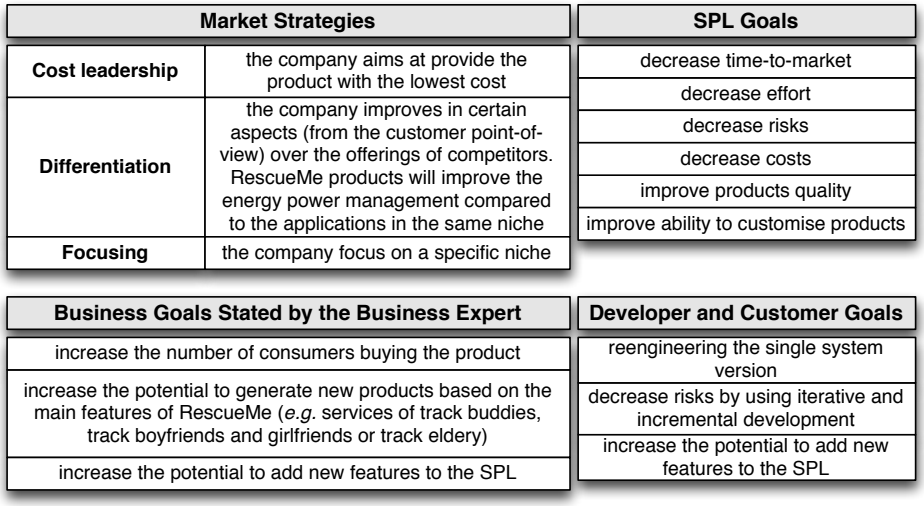

Fig. 6. RescueMe Market Analysis Document.

TABLE IV. A FRAGMENT OF THE RESCUEMESPL PRODUCT MAP.

#	RescueMe Features	Lite	Standard	Social	Pro	Ultimate
1	Destination	✔	✔	✔	✔	✔
2	SMS_Destination	✔	✔	✔	✔	✔
3	Twitter_Destination			✔	✔	✔
4	Facebook_Destination			✔	✔	✔
5	Email_Destination		✔	✔	✔	✔
6	Access_Control			✔	✔	✔
7	Web_Access_Control			✔	✔	✔
8	Mobile_Access_Control			✔	✔	✔
9	Contact	✔	✔	✔	✔	✔
10	Add_Contact	✔	✔	✔	✔	✔
11	Import_Contact		✔	✔	✔	✔
12	Phone_Import		✔	✔	✔	✔
13	Twitter_Import			✔	✔	✔
14	Facebook_Import			✔	✔	✔
15	Delete_Contact	✔	✔	✔	✔	✔
16	Emergency_Numbers		✔	✔	✔	✔
17	Tracking				✔	✔
18	Mobile_Tracking					✔
19	Web_Tracking				✔	✔
20	Location	✔	✔	✔	✔	✔
21	Language	✔	✔	✔	✔	✔
22	English_Language	✔	✔	✔	✔	✔
23	Portuguese_Language					✔
24	Spanish_Language					✔
25	About	✔	✔	✔	✔	✔
26	Info	✔	✔	✔	✔	✔
27	How_to_Use		✔	✔	✔	✔
28	User_Info		✔	✔	✔	✔

✔: Selected feature.

TABLE V. SPECIFICATION OF THE CONTACT FEATURE.

[F001] CONTACT			
Priority:	High	**Variability:**	Mandatory
Requires:	-	**Excludes:**	-
Binding Time:	Compile	**Father Feature:**	-
Child Features:	F002, F003	**Type:**	Abstract
Description:	The RescueMe contact contain useful information about people who receives the rescue message.		

and the respective sub-features, as well as `Language` and `English_Language`.

The sprints were planned for periods of two weeks, with objective of implementing and testing the selected features. The *SPL Developers* selected features with effort compatible with their development capacity. Each features had a set of related sprint tasks such as *analyzing legacy system*, *implementing sub-feature* and *testing sub-feature*. In the first sprint, for instance, five features were selected: `Contact`, `Import_Contact`, `Add_Contact`, `Delete_Contact`, and `Phone_Import`.

The *Sub-features implementation* was carried out using Objective-C language with XCode IDE[11] version 4.6.2. The chosen variability implementation technique was conditional compilation, since XCode provides support for pre-processor directives through macro definitions (each macro has the same name of the feature). Only optional and alternative features had been inplemented with pre-processor directives. An example of conditional compilation in RescueMeSPL is showed in the Listing 1.

With regard to *Sub-features testing*, the initial meetings produced testing tasks to each *SPL Tester*. For instance, when the sub-feature of `Import_Contact` was raised, the output of this meeting revealed smaller tasks (*e.g.* import contact from

[11]XCode - http://developer.apple.com/xcode/

phone address book, import contact from twitter account, and import contact from Facebook account). As the process goes on, the problem solving meetings tackled the found technical difficulties. In the end of the testing activities, an acceptance meeting deployed the test cases developed.

```
1                              ⋮
2  @interface ImportContactViewController ()
3  #ifdef FACEBOOK_IMPORT
4  @property (retain, nonatomic)
       FBFriendPickerViewController *
       friendPickerController;
5  #endif
6                              ⋮
```

Listing 1. Excerpt code from the *ImportContactViewController.m* (feature `Facebook_Import`).

The *Products derivation* is supported by the pre-processor directives implemented in the source-code for the RescueMeSPL sub-features. In XCode, the products are represented by targets. The *SPL Expert* configures the targets by selecting the macro definitions related to the features included in the respective products.

After finishing the sprint development, the reviews and retrospectives were performed as face-to-face meetings with *Scrum Team* members. During the review, the products were presented to the participants in order to compare development results and features specifications (the improvements were added to the immediately next sprint). Then, the retrospective identified strong points, drawbacks and improvements for the next sprints as well as the responsible for these improvements. All retrospective information were registered in a black board, and pictures were taken for consultation during the next sprint.

E. Findings

Aiming to synthesize evidence gathered from RescueMeSPL and considering each case study research question, we discuss on iterativeness, adaptability and feedback as follows:

Iterativeness. Adopting agile practices in SPLICE allowed the participants to perform activities in short iterations to anticipate development problems. As consequence, the *Scrum Team* was capable of providing faster response to changes, which was a proper strategy to deal with volatile domains as well as using new technologies as in RescueMeSPL.

Evidence pointed delivering artifacts and the set of products in the end of each short sprint improved the *Scrum Team* responsiveness. In the beginning of the project, the participants performed development tasks through larger iterations, and decisions considering new technologies (e.g. integration with social network platforms) were made late, since bigger changes should be considered in the *Sprint review* and *Sprint retrospective* tasks. The same ocurred for changes in the domain features. SPLICE helped the team to reach the best sprint size, achieving shorter iterations and solving such issues decisions as soon as possible.

Adaptability. Evidence during the RescueMeSPL development showed the ability of the process to be adapted whether necessary. Adapting when and how the activities were performed allowed the process to fit the specific project environment. We believe it allows SPLICE to be easily applied in different contexts.

For instance, the process was adapted the execution order of two *Release Development* activities, since the *Portfolio Planning*, *Sub-features definition* and *Commonality and variability analysis* were performed in conjunction, without iterativity. The participants argued RescueMeSPL implemented a simpler business domain, therefore, there were not a considerable number of features to divide these activities iteratively. In addition, the inspection was adapted, since the participants took advantage of face-to-face meetings to perform the inspection steps in conjunction, instead of dividing these steps in different meetings and remotely (considering the different participants).

Feedback. Frequent feedbacks can help to identify and communicate development problems earlier, as well as fostering the collaboration in the *Scrum Team* to mitigate such issues. Three agile practices helped to improve the feedback among the RescueMeSPL participants: *Sprint review*, *Sprint retrospective* and *Regular meeting*. The informal way these tasks were performed helped the participants to have more commitment compared to formal meetings, according to them.

F. Lessons Learned

The first lesson emerged in the beginning of the project. This is related to the decision of modeling the business domain as a whole (a global scope approach) instead of analyzing each major feature iteratively (an incremental scope approach). In more complex domains, our research group adopted an incremental construction of the domain scope. It allows to provide faster results in terms of deliverable software. However, the construction of the scope as upfront investment is a suitable alternative for simpler domains, since it provides a holistic overview of the SPL, and allows stakeholders to determine precisely the integration among major features that can be implemented as modules of the systems.

Using different IDE versions was a problem to be solved during the sprints. The developers had a heterogeneous resources, and there was not standard for the development environment. This difference affected the way *SPL Developers* coded the SPL, and consequently, problems emerged such as the integration problems with incompatible API versions. However, the process iterativeness allowed to quickly fix those problems. In order to avoid it, the development environment should be standardized, and *SPL Developers* must provide feedback every time a different API is imported to the project.

During the focus group meeting, the *SPL Developers* claimed the presence of an experienced *Scrum Master* was fundamental for the *Scrum Team* organization. The *Scrum Master* guided the participants on agile methods to perform the agile practices.

Furthermore, *SPL Developers* had to use a specific technology that had already been previously understood by one *Scrum Team* member. As the turn-around of *SPL Developers* was high, considering there was no place to share it, the knowledge about the technology was shared in an ad hoc way. However, this knowledge could be managed in SPLICE through tasks that captured learning and leveraged expertise in the team.

V. Threats to Validity

We discuss four threats to validity in this study [21]: construct validity, internal validity, external validity, and reliability.

Construct validity reflects to what extent the operational measures that are studied really represent what the researcher has in mind and what is investigated is conform to the research questions. For example, if the constructs discussed in the interview questions are not interpreted in the same way by the researcher and the interviewed persons, there is a threat to the construct validity. Our case study design considers multiples sources of evidence in the data collection (documentation, focus group meeting and observation) as a way of encouraging convergent lines of inquiry.

Internal validity is of concern when causal relations are examined. Our strategy consisted of using data analysis results to make possible inferences by using strategies such as coding-causal loop diagram on the evidence, textual explanation-building and discussion-validation.

External validity is concerned with to what extent it is possible to generalize the findings, and to what extent the findings are of interest to other people outside the investigated case. Since the generalization of the findings is not possible by applying one case study, we intend to apply SPLICE in different scenarios to address external validity.

Reliability is concerned with to what extent the data and the analysis are dependent on the specific researchers. Hypothetically, if another researcher later on conducted the same study, the result should be the same. This threat was addressed by developing a detailed case study protocol, in order to clarify all activities of this case study.

VI. Conclusion

In this paper, we presented a lightweight development process combining SPLE and agile practices (SPLICE). After describe the roles, tasks and activities to develop software product lines, we presented a case study pointing that SPLICE achieves short and proper iterations, possibilities for activities and tasks adaptations as well as continuous feedback. Additionally, the lessons learned from this experience bring evidence that SPLICE can guide throughout small SPL projects development.

Moreover, once SPLICE is flexible and accommodate adaptations we believe our process can also be used on medium projects to guide stakeholders during the development of rich-variant and domain volatile systems in a cost-effective manner. In this sense, we encourage further research to validate it.

As future work, we intend to define knowledge management practices for SPLICE, propose a method for proper configuration management and define activities, tasks, artifacts and roles for a lightweight SPL architectural design. In addition, we intend to perform evaluations to collect evidence from industrial environments aiming to make feasible the adoption of the process.

References

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

[2] K. Mohan, B. Ramesh, and V. Sugumaran, "Integrating software product line engineering and agile development," Software, IEEE, vol. 27, no. 3, pp. 48–55, 2010.

[3] A. Cockburn, Agile software development. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

[4] K. Beck and et. al., "Manifesto for agile software development," http://agilemanifesto.org/, Last Accessed April 2014 2001.

[5] A. Martini, L. Pareto, and J. Bosch, "Communication factors for speed and reuse in large-scale agile software development," in Proceedings of the 17th International Software Product Line Conference, ser. SPLC '13. ACM, 2013, pp. 42–51.

[6] J. Díaz, J. Pé;rez, P. P. Alarcón, and J. Garbajosa, "Agile product line engineering: A systematic literature review," Software Practice and Experience, vol. 41, no. 8, pp. 921–941, Jul. 2011.

[7] I. F. da Silva, P. A. da Mota Silveira Neto, P. O'Leary, E. S. de Almeida, and S. R. de Lemos Meira, "Agile software product lines: A systematic mapping study," Software Practice and Experience, vol. 41, no. 8, pp. 899–920, Jul. 2011.

[8] R. Paige, X. Wang, Z. Stephenson, and P. Brooke, "Towards an agile process for building software product lines," in Extreme Programming and Agile Processes in Software Engineering, ser. Lecture Notes in Computer Science, P. Abrahamsson, M. Marchesi, and G. Succi, Eds. Springer Berlin Heidelberg, 2006, vol. 4044, pp. 198–199.

[9] M. Kircher and P. Hofman, "Combining systematic reuse with agile development: Experience report," in Proceedings of the 16th International Software Product Line Conference - Volume 1, ser. SPLC '12. New York, NY, USA: ACM, 2012, pp. 215–219.

[10] G. K. Hanssen and T. E. Fígri, "Process fusion: An industrial case study on agile software product line engineering," Journal of Systems and Software, vol. 81, no. 6, pp. 843–854, Jun. 2008.

[11] J. Lonchamp, "A structured conceptual and terminological framework for software process engineering," in International Conference on the Continuous Software Process Improvement, 1993, pp. 41–53.

[12] R. Bendraou, J. Jezequel, M.-P. Gervais, and X. Blanc, "A comparison of six uml-based languages for software process modeling," IEEE Transactions on Software Engineering, vol. 36, no. 5, pp. 662–675, 2010.

[13] OMG, "Software & Systems Process Engineering Metamodel Specification (SPEM)," "OMG", Tech. Rep., Apr. 2008. [Online]. Available: http://www.omg.org/spec/SPEM/2.0

[14] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (foda) feasibility study," Carnegie-Mellon University Software Engineering Institute, Tech. Rep., November 1990.

[15] K. Schmid, "A comprehensive product line scoping approach and its validation," in Proceedings of the 24th International Conference on Software Engineering, ser. ICSE '02. ACM, 2002, pp. 593–603.

[16] I. S. Souza, G. S. da Silva Gomes, P. A. da Mota Silveira Neto, I. do Carmo Machado, E. S. de Almeida, and S. R. de Lemos Meira, "Evidence of software inspection on feature specification for software product lines," Journal of Systems and Software, vol. 86, no. 5, pp. 1172–1190, 2013.

[17] Beck, Test Driven Development: By Example. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[18] R. Rabiser, P. O'Leary, and I. Richardson, "Key activities for product derivation in software product lines," Journal of Systems and Software, vol. 84, no. 2, pp. 285–300, Feb. 2011.

[19] R. Rabiser, P. Grünbacher, and D. Dhungana, "Requirements for product derivation support: Results from a systematic literature review and an expert survey," Information and Software Technology, vol. 52, no. 3, pp. 324–346, Mar. 2010.

[20] B. Cabral, T. Vale, and E. S. d. Almeida, "Splice: Software product line integrated construction environment," in Tools Session of CBSoft 2014, 2014.

[21] R. K. Yin, Case Study Research: Design and Methods (Applied Social Research Methods), 4th ed. Sage Publications, 2008.

[22] F. Shull, J. Singer, and D. I. Sjøberg, Guide to Advanced Empirical Software Engineering. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.