Towards a Software Product Line Architecture to Build M-Learning Applications for the Teaching of Programming

Anderson S. Marcolino, Ellen F. Barbosa
Institute of Mathematics and Computer Science – University of São Paulo (ICMC–USP)
São Carlos (SP), Brazil
email: andersonmarcolino@gmail.com, francine@icmc.usp.br

***Abstract*—Software Product Line (SPL) is concerned with the sharing of common features within a family of products. It offers benefits, proven in several industry success cases. Regardless of its success, such a reuse-based development methodology has not been well explored in educational fields yet, as mobile platforms. In a different but related perspective, several initiatives have been undertaken as an attempt to improve the teaching of programming; however, no reuse approaches have been considered. In this paper we discuss the most significant approaches and methodologies for the conception of an SPL architecture according to the specificities of mobile devices and the teaching of programming. As main contributions, we highlight the identification of a set of approaches that support the conduction of the initial SPL processes, the design of a conceptual architecture model, and its qualitative evaluation with stakeholders.**

***Keywords*-Mobile learning; software architecture; software product line engineering; teaching of programming.**

I. Introduction

The reduction of costs of information and communication technologies (ICTs) have allowed the widespread use of these ICTs in social and economic sectors.

In the educational field, the increasing adoption of ICTs has contributed to the appearance of new learning modalities, such as electronic learning (e-learning), digital television based learning (t-learning) and mobile learning (m-learning) [19]. Consequently, the development of software and applications for educational purpose has been promoted as well [9].

Regarding the development of educational software, the inclusion of pedagogical and didactic issues for attending specific disciplines, as the teaching of programming, has required the adoption of methods and approaches for a better process at lower costs and time, and reduced efforts. Besides that, learning how to program is not an easy task and many problems can emerge in this domain, making it difficult to develop single software solutions capable of mitigating these problems [22].

Additionally, the number of features to be included in different ICT-based environments has expanded and led to a rise in threats to software quality aspects, since such features have not been well encapsulated for a better reuse [9].

Different reuse-based approaches have been adopted for improving and to mitigating problems in the development of educational solutions, as the Software Product Line (SPL) approach. SPL concerns the sharing features within a family of products, addressing business, architecture, processes and organizational aspects [3, 14]. SPL also provides mechanisms for the evolution of products, since new features may arise from stakeholders' needs. The addition of new features and their impact on architectural artifacts require either plenty of attention from the development team, or the automation of the updating task. The creation of new products requires technical support, which hinders and constrains the SPL adoption for development of educational applications [3].

Other issue regards the difficulty in selecting methodologies in the SPL initial phases to allow the conception of software products in the SPL life cycle. Since a wrong decision may lead to extra costs in SPL advanced phases, an appropriate selection is required and should be conducted according to the literature. However, there is a lack in the conducted works, which do not well describe the decisions to build SPL of educational purpose [15].

This paper deals with the establishment of project and design decisions for an SPL architecture designed for the development of m-learning applications for the teaching of programming. Our research encompasses the SPL domain engineering process and three of its subprocesses. The question raised regards the methodologies adopted for the development of educational SPLs and the viability of a proposal of an SPL architecture for mobile applications for the teaching of programming.

As main contributions, we aim at investigating the issues involved in a technological perspective of SPL, considering educational concepts and the inclusion of features from other modalities, as e-learning and t-learning, for m-learning. Firstly, we discuss each selected approach for allowing the conduction of the first phases of the SPL methodology. Secondly, we present the SPL architecture model, the results from its qualitative evaluation and the limitations to be overcome in future works.

URI: http://hdl.handle.net/10125/41922
ISBN: 978-0-9981331-0-2

Both contributions bring benefits: (i) for educational community, since the products can be developed with more quality; and (ii) for developers and software engineers, since they can benefit with respect to the support and to the easiness for the choice of approaches for the adoption of an SPL methodology.

The paper is organized as follows: Section II presents the concepts of SPL, m-learning, teaching of programming and the opened challenges that encompass all of them. Section III addresses three sub-processes conducted towards the SPL architecture and the methodology adopted for the selection of each approach. Section IV proposes an architecture for the SPL m-learning applications and reports the evaluation results. Section V provides our conclusions and future work.

II. Background

A. Software Product Line Approaches

SPL has proven to be a methodology for the development of software products at lower costs, shorter time, and with higher quality [18, 21]. In SPL, different artifacts used in different products must be sufficiently adaptable to fit the different systems produced. There are two classifications for the SPL features, namely commonalities and variabilities, both stored in the core asset – the repository of all reusable artifacts [3].

The selection of the features and their representations in the SPL architecture must be precise; therefore, several variability managements and SPL approaches have been developed for the management of variabilities and facilitation of the design and creation of an SPL [1].

Among such approaches there are KorbA (*Komponentebasierte Anwendungsentwicklung*) and FORM (Feature-Oriented Reuse Method for SPL) [1], PLUS method (Product Line UML-Based Software Engineering)[14], FODA (Feature-Oriented Domain Analysis)[10] and SMarty (Stereotype-based Management of Variability) [14, 17].

All such approaches can be integrated in SPL frameworks or methods, as SPLE and PLUS. The SPLE (Software Product Line Engineering) framework [18] covers all Software Engineering Institute (SEI) process to concept and manage an SPL, i.e., development of core assets, product development and management [21]. Figure 1 shows the SPLE framework.

It is composed of two life-cycles, namely *domain engineering* and *application engineering*. Those life-cycles are composed of nine sub-processes, of which eight form four pairs, namely requirements engineering, design, realisation and testing. The engineering sub-process domain results in common assets used in their application engineering counterparts for the creation of products. The ninth sub-process (*product management*) is part of the engineering life-cycle domain and deals with the scope of the product line [13, 18].

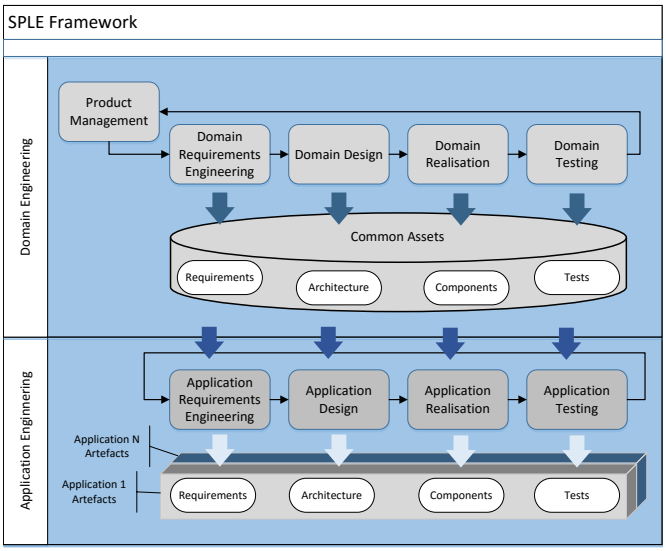

Figure 1. Life-cycles of the SPLE Framework [18].

B. Mobile Learning

The ICT-based learning modalities offer many benefits for the educational process, such as support to teachers, reduction in their workload and their empowerment for a more appropriate monitoring and feedback from learners. Educational software developed for these modalities should provide learning activities and their automatic correction and assessment. The idea is to identify the learners' difficulties, helping them to overcome their limitations [9, 12].

Particularly for m-learning, some significant changes can be noticed. M-learning has literally led the teaching and learning process to streets, in lines, trips or other places. Another important characteristic refers to personalization; since the mobile device is personal, applications with some level of configuration are preferred over the ones that do not exhibit configurable features. Several studies have been conducted for the adoption of mobile devices as a way of mitigating problems in the educational area [24]. Furthermore, the reduced price of such ICTs contributes to their acquisition [20].

Software and applications from different learning modalities, including m-learning, share several common features. Therefore, researches can identify those to be adapted for use among the different modalities. For instance, a code editor can be developed from an e-learning system for an m-learning environment. However, the mobile device specificities, as its reduced screen and the input mechanism, must be respected.

Despite the positive results showed by the learning modalities in several fields, there still are some domains

that do not well explore these advantages [8]. This is the case of the teaching of programming domain, which is briefly discussed next.

C. Teaching of Programming Domain

The teaching of programming has been significantly adopted in undergraduate courses nowadays. Furthermore, many initiatives have also investigated its adoption in primary and secondary education [5, 8].

Although several mobile applications have been developed for the teaching of programming[1], they still do not provide mechanisms for giving support to teachers as ICT-based educational software. Most applications are created for informal learning [2, 16], i.e., they do not explore features from formal learning environments that directly support the achievement of educational goals and curricula. Also, they do not provide a more configurable level of their features to attend the teaching of programming and their users, and mitigate the problems in this domain (e.g., right use of programming concepts, comprehension of programs, refactoring and factoring programs, learners' motivation, among others) [22].

Considering the teaching of programming and the lack of effective m-learning solutions in this domain, another issue can be noticed: the few solutions that adopt reuse-based software engineering approach. Specifically, we highlight the reduced number of SPLs for both educational software of general purpose and educational software for the teaching of programming [15].

In this perspective, several opened challenges to be addressed by our SPL proposal can be identified: (a) the reduced number of m-learning SPL [15]; (b) the several commonalities among the learning modalities (e-learning and t-learning) that may be adapted or reused in m-learning applications, and the several commonalities among the teaching of programming software and applications [16] that may also be adapted or reused; (c) the few adaptable solutions for the mitigation of problems in the teaching of programming and those that do not support formal learning; (d) the need for creating quality m-learning applications that support teachers' activities and facilitate feedback and decision making according to the learners' performance; and (e) the need of investigating the adoption of different mobile development technologies for the implementation of an SPL that supports different platforms and disseminates our practices and lessons learned.

III. SPL Project and Design Decisions

Based on the relevance of the three SPL main activities defined by SEI [21] and with the need for adoption a set of processes that guide the development of our SPL, our first project decision was the selection of the SPLE framework (Figure 1). We chose this framework mainly because it shows a significant and complete material, and has a well-defined process (nine sub-process) into two life-cycles. Additionally, the SPLE framework has its roots in different well-known initiatives and supports different development methods, as RUP (Rational Unified Process) [18].

Furthermore, as SPLE framework has several tasks in its sub-processes, it is easy adapting the framework for our domain, allowing the inclusion of other approaches for supporting the SPL activities, as well.

Finally, based on the SPLE framework sub-process, our study focuses on the three first sub-processes from the *domain engineering*, namely *product management*, *domain requirements engineering* and *domain design*. As these sub-processes are the most relevant for allowing the development of the SPL and its products, and literature does not provide many discussions about the approaches adopted in these first phases, we: (i) discuss and justify several project and design decisions; (ii) conduct the three first phases; and (iii) propose and evaluate an m-learning conceptual architecture model. We have selected the three first sub-processes because of their importance as requirements for the conduction of next six sub-processes of the SPLE framework.

A. Methodology

The methodology applied to support the selection of the approaches and methods that complement each sub-process of the SPLE framework follows these steps:

1) Conduction of two systematic mappings (SM) for: (i) the identification of educational SPLs and approaches adopted to support our selection of approaches; and (ii) the identification of m-learning architectures, for a possible adoption of an existing architecture, or even for proposing improvements or comparisons with our architecture model;
2) Selection of approaches and their artifacts for each SPLE sub-process based on the evaluation results of these approaches from the literature primary studies (first SM);
3) Identification of m-learning architectures based on the literature primary studies (second SM);
4) Conduction of the three first phases of the SPLE framework; and
5) Evaluations of each artifact generated for possible improvements in the selected approaches and for guaranteeing that each selection is the best for integrating the proposed SPL.

Each sub-process that integrates the approaches and methods (Section II) is discussed next.

[1] goo.gl/019DIJ

B. Product Management

This SPLE sub-process determines the common and variable features in an SPL for the definition of the scope of the line. If a new feature is required, the sub-process measures its impact and the implications for updating all the other sub-processes involved. To support the management of the features in this sub-process, we selected FeatureIDE tool[2] [23], which is an Eclipse[3] Plugin that generates an XML (eXtensible Markup Language) capable to be used together with mechanisms (e.g., a parser) or other tools for management of the products, and support to source code traceability.

The FeatureIDE tool was also used to conduct the validation of our feature model. The tool found no dead feature and identified 874 possible valid products.

C. Domain Requirements Engineering

This sub-process is responsible for the five basic requirements engineering tasks, namely elicitation, documentation, negotiation, validation/verification and management. It also encompasses the conception of the requirements artifacts, i.e., goals, features, scenarios, use cases, and data, functional and behavioral models [13].

The goal artifact describes the intention (objective) the system or application under consideration should achieve. It is defined to be aligned with the stakeholders' needs. The aim of our research is *how to improve the teaching of programming through m-learning applications*. To achieve it, the conception of the applications should consider a reuse-based approach for a better development process. As addressed in Section I, we chose to work with an SPL mainly due to: (i) the results from a Systematic Mapping (SM) [15], which did not retrieve SPLs for m-learning and for the teaching of programming domains; (ii) the benefits provided by the adoption of the SPL methodology; and (iii) the capability of working with variabilities and commonalities from several software and applications for teaching of programming [16].

Features, in the context of requirement artifacts, are end-user visible characteristics of a system. In an SPL, they define the set of common and variable characteristics to be selected and used for the development of products. Each feature, when representing variabilities, needs to satisfy constraints to be consistently used. Such features can be represented in the SPL context, through many methods, among them we chose the feature model [10] to be used as an input artifact for the next sub-process.

Scenarios and use cases are requirements artifacts that facilitate communication among the parts representing the requirements. They represent the actors' interactions (users and other entities) with the system, reducing the chances of ambiguity. Since our aim is to develop an SPL that attends a wide range of applications for the teaching of programming, the models were not modeled at the initial phase. Such decision was taken according to the following project restrictions: (i) reduced team of developers, (ii) short time for the development process; and (iii) delivery of quick solutions. The scenarios and use cases diagrams will be included as artifacts in the core assets during and after the *domain realisation* sub-process.

The next task regards the definition of the set of requirements according to the five basic requirements engineering tasks [18]:

Elicitation: in this phase, the analysis of the stakeholders' needs was conducted. A catalog was proposed encompassing features for dealing with the teaching of programming domain. We have elicited thirty-three primary studies from a previously conducted SM [16], extracting their features;

Documentation: in this phase, the features previously identified as product requirements were written in a more precise way. Since our aim is the development of m-learning applications for the teaching of programming, we did not only consider the applications features, but also those related to the specificities of the mobile platform and general educational concepts;

Negotiation: in this phase the stakeholders' opinion is considered to obtain an adequate level of consensus on the requirements. Since this phase was not adequate for the catalog conception, a case study with a teacher and a team of three developers was conducted as a way to get feedback.

Validation and verification: in this phase, the requirements were analyzed for the creation of a set of clear, complete, correct and understandable requirements. Several interactions were conducted for refining the number of requirements from 498 in the initial analysis to 97 in the beta version of the catalog[4]. However, it is important to notice that these requirements can change in the course of our research.

Management: this phase deals with the maintenance of the requirements in the development and the product line life-cycle, implying in possible changes in the defined requirements and the inclusion of new ones. The phase should be retaken at each new inclusion or modification of the requirements scope and is directly related to the *product management* sub-process.

D. Domain Design

This sub-process generates the reference architecture of the m-learning applications for the teaching of pro-

[2]goo.gl/JfqxEu

[3]http://www.eclipse.org/

[4]http://caed.icmc.usp.br/mlearning/?page=index

gramming. For our SPL, the architecture is divided in two parts. The first part represents an infrastructure for supporting educators in the development of m-learning applications using the core assets. The second part presents the m-learning architecture model, which was proposed with basis on the requirements catalog and on the feature model. All features presented on the second part come from the core assets and should generate a valid mobile application from the supporting infrastructure. For modeling both parts of the proposed architecture, two systematic mappings were conducted.

Systematic Mapping of Educational SPLs

The first SM was conducted in 2015, retrieving 10 SPLs for educational purposes and none SPL for the teaching of programming [15]. The results were considered for the selection of approaches for supporting the sub-processes from SPLE framework and for the development of the architecture of the infrastructure for creation of the SPLs products (first part). The complete protocol is available at https://goo.gl/Q70pL1.

The most significant SPL conception method adopted in the SM results was the extractive, following by the proactive; none of them adopted the reactive method. Those three methods were propose by Krueger [11]. Briefly, in the proactive method the SPL is developed from a scratch. In the reactive, the SPL is incremented according to the demand. Finally, in the extractive method, artifacts are extracted from software and applications for evolving the SPL [11].

For the engineering domain, FODA approach [10] was adopted in most of the studies, followed by SMarty [14, 17]. The technologies and tools adopted in the application engineering were: REST (representational state transfer), Java, Android, Ajax, AOP (aspect oriented programming) and S.P.L.O.T. (SPL online tools)[5], a tool that, such as the FeatureIDE, can validate feature models.

The most important primary study retrieved was the proposed by FalvoJr et al. [6]. In this study, the authors specify an m-learning SPL for educational generic purpose with a supporting mechanism. Based on this SPL, we considered the adopted technologies and the project, a SOA (service-oriented architecture), for developing our supporting infrastructure, described next.

Systematic Mapping of M-Learning Architectures

The second SM was conducted in 2016, retrieving 26 mobile learning architectures. The complete protocol of this mapping is available at http://goo.gl/Gp9Fd0.

The SM searched for general architectures that encompass a big number of features from m-learning applications and learning environments. However, none of the returned architectures of the SM allowed its adoption as main architecture in our research.

The returned architectures do not encompass the features identified in our domain requirements engineering phase, as contents, learning activities, assessment, feedback and learning performance. Those features were found isolated in most of the cases, hindering their adoption. Thereby, the primary studies were used only for allowing comparisons and improvements in our architecture model, during the domain design phase.

Finally, the results of both SMs, the m-learning requirements catalog and the feature model modeled using FeatureIDE tool were considered as inputs for the conduction of the *domain design* sub-process. The SPL architecture developed in this phase is presented next.

IV. A M-Learning SPL Architecture Proposal

The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them. The architecture is defined as the fundamental organization of a system, embodied in its components, their relationships with each other and the environment, and the principles governing its design and evolution [4].

Different levels must be adopted for the representation of an architecture. Specifically for an SPL approach, notations used for documenting software architecture in single systems engineering generally do not provide sufficient means to represent variability, requiring specific documents, as a reference architecture.

The architecture developed in the *domain engineering* is called reference architecture and includes the variation points and variants documented in the feature model. It also determines the reusable components to be developed during *Domain Realisation* and tested in the *Domain Testing* sub-process [18]. The main artifacts of an SPL architecture are: (i) the architectural structure, (ii) the architectural texture, and (iii) the architecture views (logical view, development, process and code view) [18].

Basically, the *architectural structure* exhibits, in a macro view, the layers (packages) and their sub-packages of the applications, whereas the *architectural texture* determines the general rules each of the parts must obey. Finally, the architecture views represent the system in different ways to capture all the specifications for the architecture development.

- *Logical view*: describes the application related with its problem domain. In our project, it is represented by the feature model;
- *Development view*: represents the hierarchical decomposition of the system into pieces. Since we adopted UML and SMarty to represent this decom-

[5]http://www.splot-research.org/

position, the most used representations are package diagram, component diagram and class diagram;
- *Process view*: shows the decomposition of the running system into ordered activities and their relationships. The activity and sequence UML diagrams are used to represent them; and
- *Code view*: represents the decomposition of the executable code into files and their assignment for processing units. The main artifact that expresses this decomposition is the UML deployment diagram.

Figure 2 shows a macro representation of our SPL architecture model through a diagram of layers (packages), its sub-packages and a list of approaches, tools and technologies adopted with their respective life-cycle and architectural layers. It is highlighted that the technologies presented are related only with the first part of the architecture, responsible for facilitating the conception of the m-learning applications. Such a structure is proposed based on the primary study [6] returned in the first SM conducted [15].

At the top of Figure 2 is the SPLE framework and its layers. The interaction between *domain engineering* life-cycle and *application mechanism* (green arrow) supports the creation of applications. *Application mechanism* interacts with the *application engineering* life-cycle for the creation of m-learning applications for the teaching of programming (blue arrow), which reduces the amount of technical support.

The user selects the desired features through a web user interface (*Web UI*) and sends their requisition to the server (*REST services*), which identifies them, selecting the *application template* desired. All artifacts are put together and the server returns the desired m-learning application to the user.

In the domain engineering layer, *Domain Requirements Engineering* is the sub-process that creates the *M-learning Requirement Catalog*, which is the main artifact used with the FeatureIDE tool for the creation of the *Feature Model*. As addressed in Section III, the model was adopted in the *Domain Design* for the proposal of the architecture model, including the variability management facilities provided by the *SMarty* approach.

Our model was divided according to the nine categories defined in the m-learning requirement catalog, namely feedback and results, users, monitoring and learning performance, contents, programming mechanisms, assessments, support to teaching and learning, learning activities and nonfunctional requirements. All categories were considered during the design and definition of the layers (packages) and their sub-packages of the m-learning applications architecture proposal, as illustrated in Figure 2. Packages, sub-packages and relationships in dashed lines represent variable features, the ones represented with solid lines are common features.

The blue arrow specifies the m-learning applications architecture. The mobile client application provides a macro view of *Presentation Layer*, *Service Layer*, *Business Layer* (Educational and Programming), *Data Layer* (persistence) and *Cross-Cutting/Orthogonal Services Layer* and their interactions with external elements, *External Infrastructure* and *External Sources*.

In this point, we highlight the first adaptation of the SPLE framework to our needs. To fit our educational domain and to perform a fast delivery of m-learning applications we have adopted the proactive and the reactive development approaches [11] for the next phase (*domain realisation*) (Figure 2 – Red arrow).

The decision implies in the identification of possible new features required by the users (teachers or learners). Consequently, the sub-processes of the *application engineering* must send feedback to the *domain engineering* layer, triggering the reactive process of development of new features and updating of other artifacts. This is essential to guarantee that the SPL allows a concise generation of new product configurations.

A. Architectural Structure and Texture

The structure of the m-learning applications depicted in Figure 2 includes five layers, besides *external infrastructure* and *external sources*. The layers and their texture are characterized as follows:

Presentation Layer: it is the layer responsible for providing an user graphic interface and its components presented in *UI Components* sub-package. It is also responsible for translating the users' interactions in logic commands that will be redirected for the other application layers though the *presentation logic components/-controller* sub-package.

The challenges for the *domain realization* (the next SPLE sub-process) related to such components are the conception of different and adaptable presentations, since an application may be executed in different mobile devices of different screen sizes. Besides, it is need to reduce graphical elements from other modalities solutions, decreasing the user's memory load for recognizing the interface and their functionalities. The main resource to reduce the load of graphical elements is put together all functionalities in floating menus and buttons which can be hidden when other functionality is being used.

Additionally, input interactions, mainly the virtual keypads, must be more pleasant and natural for not demotivating the users. In learning activities, for instance, where it is need to write long texts as source-code, it is important to allow the communication with other input physical devices or, as an alternative, creating new presentations for the virtual keypads.

Services Layer: the *service interfaces* and data contracts (or message types) are defined and implemented

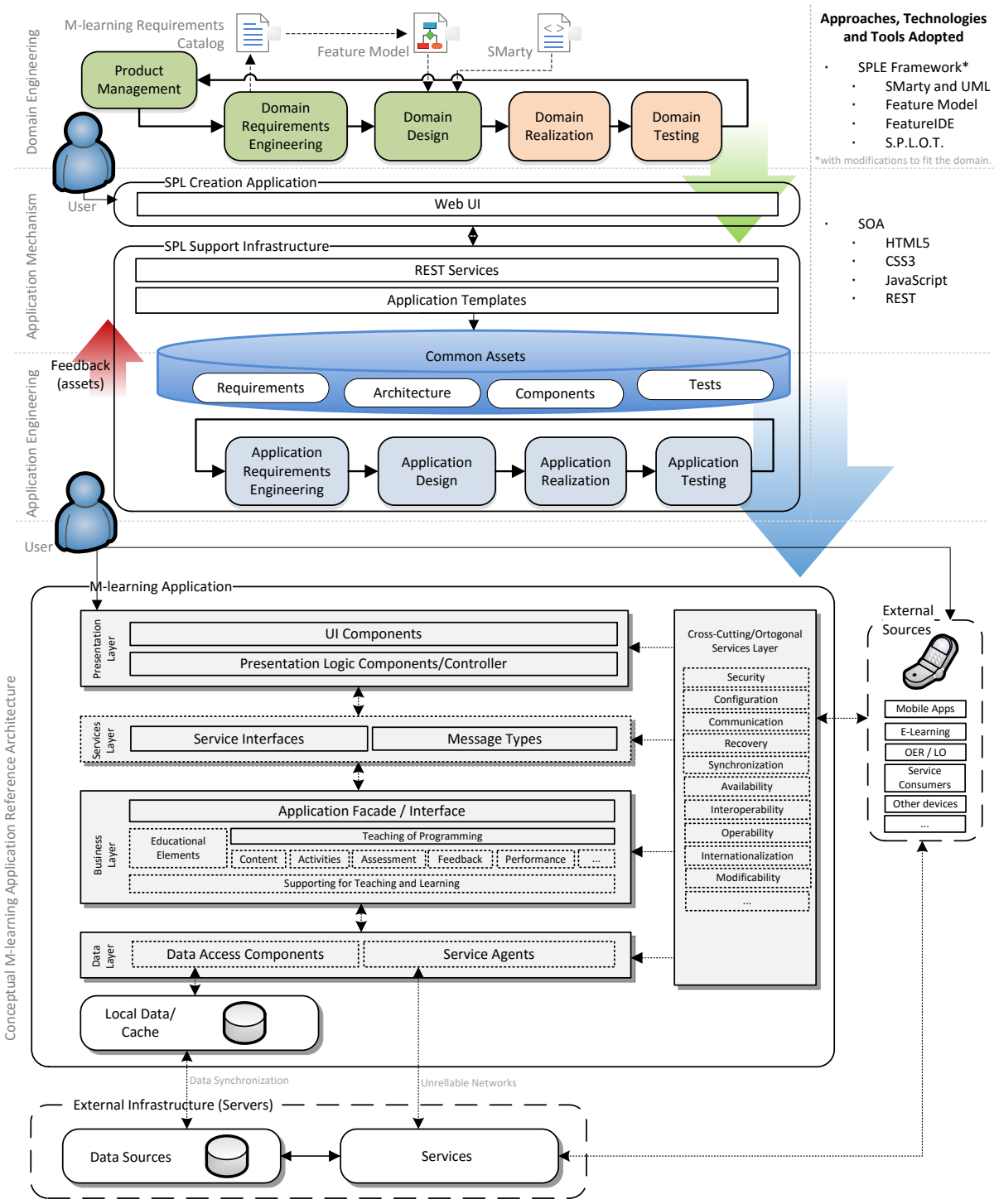

Figure 2. Macro View of the Architecture of M-Learning Application and SPL Application Generation.

in this layer, which avoids the exposition of details of the internal processes or the business rules used in the application. *Service interfaces* expose the business logic implemented in the application for potential consumers.

The main challenge is the definition of resources to be available to clients. Moreover, the way such services will behave and relate with the business layer (educational and programming domain features) in a more isolated way must be designed.

Business Layer: this layer encompasses both the educational domain issues and the teaching of programming domain issues. The *application facade/ interface* receives the process requisitions and redirects them to the desired educational sub-packages. The idea is to provide a simplified interface to the business logic components often through the combination of multiple business operations into a single operation, and facilitating the use of the business logic and reducing dependencies.

In the educational context, the logic layer is responsible for each supporting mechanism to content, activities, assessments, learning feedback and performance.

The *supporting for teaching and learning* sub-package receives data collected in the *UI components* and uses them for conducting a business process according to a preestablished order. It deals with the availability of the different educational mechanisms (content, activities, assessment, feedback and performance) and their relationships among them and with the educational goals.

The *educational elements*, on the other hand, group the entities responsible for encapsulating the business logic and data necessary for representing real world elements, such as learners, teachers, classes, courses. They provide access to the business data and related functionalities, validating the data within the entity and encapsulating the logic application for ensuring consistency and implementation of business rules and behavior. Intelligent tutoring and other support entities are also included in this sub-package.

Finally, the other units are directly related to the *teaching of programming* sub-package, responsible for including the teaching of programming domain, describing the way it will be taught to learners, the strategies, representations, mechanisms and necessary tools that provide the correct solution to stakeholders and achievement of the educational goal.

When the user selects the desired features in the *web UI* for creating the application, tips related to each selected programming feature and the problems in the teaching of programming that may be mitigated by a specific feature are given. Additionally, considerations of learning theories, also included in our m-learning requirements catalog, can be used for tracing the desired educational goal and presented to users for supporting the process of application creation.

Data Layer: this layer abstracts the logic required for the access to the data stores, which can occur in either a local data (client side application), or a cache, or directly in a remote data base synchronized with features from *cross-cutting/orthogonal services layer*. They centralize common data access functionalities to facilitate the application for configuration and maintenance. The different components for those tasks are included in the *data access components* sub-package.

Finally, sub-package *Service Agents* implements data access components necessary for communicating the application with the services in the *external infrastructure*. It isolates the varying requirements for calling services from the application and converts the data from the application to be used for the services.

The main challenge in the components is the guarantee of the data's quality traffic to be supported by cross-cutting elements and the respective service approach adopted. Additionally, the processes to monitor and to store data is also a challenge. Based on a big number of learners' access, the infrastructure for storing data must be available and capable to provide all the simultaneous accesses needed.

Cross-Cutting/Orthogonal services Layer: this layer represents all non-functional requirements and the features related to technical issues, as security, configuration, communications (protocols, technologies), mobility and mobile devices specificities, availability and internationalization. It also includes the communication with *external sources*, such as external applications, other learning systems, as e-learning or even t-learning systems, with other systems that may consume the same services from the application, and repositories of learning objects (LO) or/and open educational resources (OER).

The challenge related with the cross-cutting components, or orthogonal services – when the application is a service-based, is how to encapsulate them. As they are cross-cutting, the encapsulation of these components is not a trivial task and, for the majority of them, it is nearly impossible since they deal with specificities of quality that are spread for all the other components. Furthermore, considerations of quality of services must be included, based on a priority order defined by the stakeholders.

Additional Considerations: The architecture can support the development of m-learning applications for both formal or informal learning, including or not services, justifying the variable packages and sub-packages in Figure 2. When only a service-oriented architectural style (SOA) is considered, the *business layer*, *data layer* and the most components of the *cross-cutting/orthogonal services layer* must be allocated on the server side. In other words, such layers and their components must be available only on the server side that is being accessed by the services layer. When the stakeholders select features without services, all the functionalities will be stored at user mobile device.

The different abstraction levels of the diagrams can lead to a better understanding of each package and set of components in their respective sub-package. The models are modeled for the execution of the next SPLE sub-processes, namely *domain realization* and *domain testing*, which will allow the generation of m-learning

applications to be tested. Additionally, prior to the development of the components and the supporting infrastructure, the technologies of front-end and back-end must be identified and other m-learning applications with open source-code must be considered to be candidates in the extraction of their components for our SPL.

Next we show a preliminary evaluation of the model through an online questionnaire (qualitative analysis).

B. Questionnaire with Experts

To perform the next phases of the development of the proposed SPL, the model of the architecture should be approved by the main stakeholders involved, i.e., software engineers, programming teachers, m-learning experts and mobile developers. Thus, an online questionnaire in a checklist format was applied with the stakeholders, called from now as participants.

A total of 31 participants answered the questionnaire. 23 were programming teachers, and eight were software engineers (and had never taught programming before). Among the 23 teachers, three were m-learning experts and four were mobile developers. They have 5.2 years of expertise in average.

24 questions were proposed based on [7]. Each question could be answered according to the following: (1) totally complies; (2) partially complies; (3) does not comply; and (4) indifferent. The complete qualitative study protocol and results are available at http://goo.gl/Ll3o8D. The questions were divided in seven groups: *general aspects*, *conceptual aspects*, *infrastructure aspects*, *modeling aspects*, *SOA aspects*, *educational domain aspects*, and *m-learning domains aspects*. All questions were previously analyzed through a pilot execution.

The majority of the participants (52%), totally complied that the conceptual model architecture and its educational and mobile learning concepts were feasible. 42% indicated that the model partially satisfied the domain, 2% indicated that the model does not comply, and 4% remained indifferent. We point out that the indifferent answer was indicated when the participant did not have the expertise to answer an specific question.

Based on the answers of all participants, we observe that all the seven evaluated aspects represented in the architecture model were considered totally satisfied, as shown in Figure 3.

Considering the answers by stakeholders' expertise, as teachers of programming, software engineers, mobile developers and m-learning experts, a different perspective can be also analyzed. For the 16 teachers of programming, the most of aspects were considered totally complied, with one exception for the *modeling aspects*, showing a statistical tie for totally and partially complied answers.

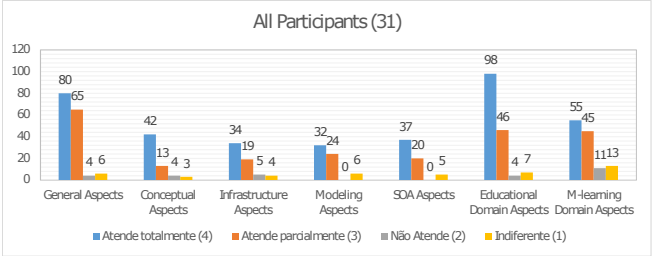

Figure 3. Evaluation Results by Set of Questions.

In the perspective of software engineers (8), the *infrastructure aspects* were totally complied, *conceptual* and *modeling aspects* had a statistical tie between the totally and partially complied answers. The other aspects were considered partially complied.

Mobile developers (4), as the teachers of programming, considered all the aspects totally satisfactory. Finally, m-learning experts' answers (3) showed a statistical tie. They considered the *infrastructure aspects* as totally and partially complied. The other aspects were totally complied.

Despite the small number of participants, they represent the roles of the stakeholders that we intend to support with our SPL. Further evaluations with other methods will be applied with next models of the architecture in order to allow its concise development.

V. Conclusions and Future Works

Based on different approaches and methodologies, in this study we described the SPLE framework and the adaptations made for the inclusion of specific mechanisms, allowing the conception of the SPL architecture model for the development of m-learning applications for the teaching of programming. The choices and approaches adopted can support project and design decisions for the conception of SPLs in other domains.

The main challenges, as seen in each layer of our proposal, are related with adaptation of interfaces, functionalities and strategies for the effective adoption of mobile devices in the teaching of programming.

Based on a questionnaire applied with the main stakeholders, the architecture model was considered feasible in all the aspects that the high model represents, thus allowing the conduction of the next phases of the SPLE framework.

As future work, we highlight: (i) model the architectural UML diagrams discussed in this study; (ii) validate these UML models with experts; (iii) conduct the *domain realisation* phase, in which we should develop the components of our architecture and evolve the SPL for allowing the conception of our first applications; (iv) develop the proposed mechanism to facilitate the generation of products by non-technical users; and (v) include

new components, extracted from other m-learning applications, to populate the SPL core assets.

Acknowledgments

The authors acknowledge Brazilian funding agencies – Fapesp (Processes 2013/07375-0 and 2014/03389-9), CAPES (Procad 071/2013) and CNPq – for the financial support, and thank Dr. Timo Käkölä and the anonymous reviewers for their valuable suggestions to improve the quality of the paper.

References

[1] H. Ahn and S. Kang. A comparison of software product line architecture design methods from the practicality viewpoint. *Korea Conf. on Soft. Eng.*, 2009.

[2] V. C. O. Aureliano and P. C. A. R. Tedesco. Ensino e Aprendizagem de Programação para Iniciantes: uma Revisão Sistemática da Literatura focada no SBIE e WIE. *Simp. Brasileiro de Inf. na Ed.*, 2012.

[3] R. Capilla, J. Bosch, and K.C. Kang. *Systems and Software Variability Management: Concepts, Tools and Experiences*. Springer, 2013.

[4] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2002.

[5] C. Duncan, T. Bell, and S. Tanimoto. Should your 8-year-old learn coding? In *Proceedings of the Workshop in Primary and Secondary Comp. Education*, pages 60–69, New York, USA, 2014.

[6] V. Falvo Junior, N. F. Duarte Filho, E. OliveiraJr, and E. F. Barbosa. Towards the Establishment of a Software Product Line for Mobile Learning Applications. *Int. Conf. on Soft. Eng. and Knowledge Engineering*, 1:678–683, 2014.

[7] N. F. D. Filho and E. F. Barbosa. A contribution to the establishment of reference architectures for mobile learning environments. *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje*, 10(4):234–241, Nov 2015.

[8] K.J. Harms, D. Cosgrove, S. Gray, and C. Kelleher. Automatically Generating Tutorials to Enable Middle School Children to Learn Programming Independently. pages 11–19, 2013.

[9] A. Jalil, M. Beer, and P. Crowther. Pedagogical requirements for mobile learning: A review on mobilearn task model. *Journal of Int. Media in Education*, (1), 2015.

[10] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Carnegie-Mellon University, November 1990.

[11] C. W. Krueger. Easing the Transition to Software Mass Customization. In *Soft. Product-Family Eng.*, volume 2290, pages 282–293. Springer, 2002.

[12] E. Lavrischeva and A. Ostrovski. General disciplines and tools for e-learning software engineering. *Communications in Comp. and Inf. Science*, 347 CCIS:212–229, 2013.

[13] F. J. Linden, K. Schmid, and E. Rommes. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer-Verlag New York, Inc., 2007.

[14] A. Marcolino, E. OliveiraJr, I. M. S. Gimenes, and E. F. Barbosa. Empirically Based Evolution of a Variability Management Approach at UML Class Level. In *Annual Comp. Soft. and Applications Conf., Vasteras, Sweden.*, pages 354–363, 2014.

[15] A. S. Marcolino and E. F. Barbosa. Linhas de Produto de Software no Domínio Educacional: Um Mapeamento Sistemático. *Simp. Brasileiro de Inf. na Educação*, 1:239–249, 2015.

[16] A. S. Marcolino and E. F. Barbosa. Softwares Educacionais para o Ensino de Programação: Um Mapeamento Sistemático. *Simp. Brasileiro de Inf. na Educação*, 1:190–199, 2015.

[17] E. OliveiraJr, I. M. S. Gimenes, and J. C. Maldonado. Systematic Management of Variability in UML–based Software Product Lines. *Journal of Universal Comp. Science*, 16(17):2374–2393, 2010.

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

[19] N. Rubens, D. Kaplan, and T. Okamoto. E-Learning 3.0: Anyone, Anywhere, Anytime, and AI. In *New Horizons in Web Based Learning*, pages 171–180. Springer, 2014.

[20] M. Sarrab. M-learning in Education: Omani Undergraduate Students Perspective. *Procedia - Social and Behavioral Sciences*, 176:834 – 839, 2015. Int. Educational Tec. Conf. Chicago, IL, USA.

[21] SEI. Software Engineering Institute A Framework for Software Product Line Practice. http://www.sei.cmu.edu/productlines/, 2016.

[22] D. M. Souza, M. H. S. Batista, and E. F. Barbosa. Problemas e dificuldades no ensino de programação: Um mapeamento sistemático. In *Revista Brasileira de Inf. na Ed.*, volume 24, pages 1–14, 2016.

[23] T. Thüm, C. Kästner, F. Benduhn, J. Meinicke, G. Saake, and T. Leich. Featureide: An extensible framework for feature-oriented software development. *Science of Comp. Prog.*, 79:70–85, 2014.

[24] UNESCO. Mobile learning for teachers global themes, 2012. Accessed in 10 out. 2013.