Evolving a Mobile Learning Software Product Line for the Teaching of Programming through an Industry Practitioner Perspective

Anderson S. Marcolino*, Ellen F. Barbosa†
*Department of Engineering and Exact – Federal University of Paraná (UFPR)
Palotina (PR), Brazil – Email: anderson.marcolino@ufpr.br
†Institute of Science and Computational Mathematics – University of São Paulo (ICMC–USP)
São Carlos (SP), Brazil – Email: francine@icmc.usp.br

***Abstract*—The process that unifies theoretical techniques from academia with the practical expertise from industry has been considered an interesting approach to software development, since it can increase quality, reduce costs and allow a high level of reuse. In this perspective, we have worked on the integration of industry practices with academic research for evolving a software product line (SPL) for the development of mobile applications in the context of the teaching of programming. The first version of the proposed SPL was evaluated in a previous study and, as a result, we noticed a lack of integration of practitioners in the performed evaluation. Thereby, this work aims at investigating if the expertise of software industry practitioners can somehow improve the SPL proposed and, additionally, identifying which strategies can be adopted to complement the academic theories. Based on this goal we discuss: (i) the evolution of the SPL conceptual architecture under an industry practitioners' perspective; (ii) the proposed improvements based on the industry professionals' point of view; (iii) the representation of the improved SPL architecture adopting an SPL modeling approach; and (iv) the adoption of software industry strategies in the SPL engineering sub-processes, based on quickly development process in small development companies.**

I. Introduction

The widely use of information and communication technologies (ICTs) has providing significant changes in the educational process, resulting in emerging learning modalities. In general, teachers have adopted such technologies both as a supporting mechanism in the classroom as well as to promoting the continuity of studies in the students' homes. Mobile learning (m-learning) is one of these learning modalities, enabling the learning at any place and time through mobile devices. Moreover, since mobile devices are considered as low-cost technologies, they have become part of the routine of young, adult and elder people worldwide, creating a propitious scenario for their adoption in the educational process [7, 14, 15].

On the other hand, the development of educational applications, or just *apps*, can hinder the adoption of learning modalities based on ICTs, due to the several issues involved, such as the need of introducing pedagogical and didactic features in the solutions for supporting both teachers and students; and the need of assuring the low-cost of developing a software that really caters teachers and students [7]. An example of domain that still needs to overcome several issues regarding teaching and learning refers to programming foundations [21].

The majority of mobile *apps* regarding the teaching of programming foundations supports informal learning and does not provide an adequate level of feedback [15]. Informal learning refers to the process of learning without support of a human mediator, e.g., a tutor [7]. So, the *app* itself is responsible for providing feedback. This feedback, however, does not exactly show the students' limitations; consequently, teachers still need to monitoring the students, in person, through the learning process, being the main mediator. Additionally, the small-sized keyboards and the few adoption of sensors and touchscreen interactions may demotivated the learners to program on them. In fact, there is a need for researches that consider the adoption of interactions seen in social network and *apps* games for providing real m-learning experiences, besides to identifying the features that improve the learning on this modality [4, 15].

In this perspective, in our work we have investigated the use of the software product line (SPL) approach for supporting the development of m-learning *apps* for the teaching of programming. The several issues and concerns involved in the educational area and in the development of learning *apps* can be benefited from this methodology, since the teachers can decide which features they expect in the *apps* for supporting them in classrooms. That is, through an SPL it is possible to select features of content and learning activities, or only activities. Besides, the *apps* can integrate different means of interaction and formats of learning content and activities [12, 14].

Marcolino and Barbosa [14] proposed an SPL conceptual architecture of such line, conducting the three first sub-processes from the SPLE (Software Product Line Engineering) [19]: product management, domain

requirements engineering and domain design. The SPL conceptual architecture [14] was evaluated by 31 experts from academia (software engineers, programming teachers, m-learning experts and mobile developers), suggesting the architecture design is feasible. However, almost all participants were from the academic area. This can hinder the development process of the proposed architecture, due the lack of an analysis provided by industry practice point of view [2, 5].

Considering this scenario, this work aims at answering three main research questions: (1) *Can the proposed SPL architecture model be somehow improved by the analysis of software industry practitioners?*; (2) *Which points of our SPL architecture, proposed based on academic theories, need improvement according to the software industry practitioners (and their techniques)?*; and (3) *What software industry strategies can be applied in the development of the proposed SPL and in their features for m-learning apps*?

The investigation of these questions allowed us to present: (i) the evolution of the architecture; (ii) the analysis and discussion of the improvements; (iii) the representation of the improved architecture with SMarty (Stereotype-based Management of Variability) approach [11]; and (iv) the discussion regarding the adoption of software industry strategies.

This paper is organized as follows. Section II summarizes the main concepts regarding m-learning and the teaching of programming, as well as the proposed m-learning SPL and the SMarty approach. Section III discusses the evolution of the SPL conceptual architecture process. Section IV presents the architecture represented with UML and SMarty. Finally, Section V shows our conclusions and perspectives for future work.

II. Background

M-learning and the Teaching of Programming: Mobile devices provide a more personal experience, besides being cheaper than desktops computers [4]. Moreover, they can be adopted as a way for carrying the classrooms lessons to the students' homes [15].

The number of educational *apps* has grown significantly on the last years. However, most of such *apps* support only informal learning [15]. Besides, there are still many issues to overcome for allowing the effective use of mobile devices in the education, in several areas, such as in the teaching of programming [1, 15].

In the last years, many countries have changing their curricula for including programming disciplines in their courses. However, despite the increasing interest in programming, the teaching of programming foundations still faces many problems [3]. Among them we highlight the need of improving the support for teachers in classrooms, to provide more accurate feedback to the students and to establish more engaging learning environments [1, 3, 15, 18].

Based on these issues and on the need of developing educational *apps*, an SPL to create m-learning *apps* for the teaching of programming foundations has been proposed [14]. This SPL is summarized next.

The M-Learning SPL: it follows the SPLE (Software Product Line Engineering) framework [19]. Composed of nine sub-processes, the SPLE describes the main activities and artifacts of each sub-process, facilitating the conduction of the SPL life-cycles, namely: (i) domain engineering; and (ii) application engineering. Marcolino and Barbosa [14] specified each phase in the creation of a m-learning SPL, its conceptual architecture and the proposed improvements in the first three SPLE sub-processes. The M-Learning SPL Conceptual Architecture Excerpt available on https://bit.ly/3e5mVVk shows an excerpt of the SPL conceptual architecture.

The M-Learning SPL Conceptual Architecture Excerpt, the m-learning *apps* architecture presents the mobile client application, that 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*.

The Domain design differs from design of single systems mainly because it incorporates configurations mechanisms into the SPL reference architecture to support the variability of the SPL. These mechanisms are discussed in [14], being considered in the final SPL solution. Furthermore, it allows the development of an adaptable architecture for inclusion of new features.

The resulting SPL conceptual architecture [14] was evaluated without considering the industry practitioners. Therefore, a reevaluation of the proposed SPL in a practitioners' perspective is still need. Additionally, this analysis may attenuate possible future problems in the development phase, besides to identify how the practitioners may improve the academic theories adopted in the conception of the initial SPL model.

The SMarty Approach: The SPLE framework designates the adoption of an orthogonal variability model [19]. However, this model adopts a specific graphical notation; also, the creation of such models requires specific modeling tools. To minimize the inclusion of a new notation, the SMarty approach was adopted in the conception of the SPL models [11].

In short, SMarty supports the representation and management of variability in UML models. It supports use case, sequence, class, activity and component UML models in an UML 2 profile, named *SMartyProfile*. UML profile allows an easier integration of their notation in any modeling tool that enables UML profiles import.

SMarty also provides a set of guidelines to help users in its application, entitled *SMartyProcess*.

SMarty has shown good results from a set of experimental evaluations [11]. Moreover, as the approach allows the management of their stereotypes in a profile, the addition of new syntactic elements is facilitated [11], enabling the constant evolution of the line core assets.

III. Evolving the SPL Architecture

Despite the satisfactory results obtained in the SPL architecture evaluation, this can imply in a possible bias in the evaluation, particularly if we considered an industry perspective [5, 14]. Other issue refers to the results regarding to *conceptual aspects*, *infrastructure aspects* and *Software Oriented Architecture (SOA) aspects*. Comparing *general aspects* and *educational domain aspects* percentage results indicate that such items form the architecture *partially complies* their roles, suggesting possible issues to be reanalyzed.

We also identified the need of evaluating the architecture and the requirements catalog together, since the individual evaluation of them may imply on issues such as imprecise understanding of the software solutions. If the *domain design* and *domain realization* SPLE subprocesses are conducted without proper analysis, the cost and effort in refactoring the project and the subprocesses involved can be high.

Finally, there is a need for investigating the benefits in the integration of professionals' expertise with the techniques coming from academia. Particularly, it is aimed the identification in what aspects the techniques from academia fail and what strategies from industry may improve the development process of the presented SPL, mainly in small development companies. Thereby, aiming at addressing these issues, a reevaluation of the proposed architecture was conducted.

Methodology and Execution: The methodology applied to support the new analysis of the conceptual architecture follows an industry protocol in which the requirements and needs are analyzed for the architecture considered. This protocol was created and adopted by the practitioners who were volunteers in this analysis. Such protocol have been used in the development of their portfolio of software products.

The industry protocol involves the process of rigorous analysis of: (i) each requirement (functional and non functional) independently, for the identification of its possible impact as well as the development technique to be adopted; (ii) the sets of requirements according to their characteristics (i.e., if all of them are requirements related to educational activity, they will be integrated in a same set of requirements) and the expected and unexpected behaviors, when combined with other sets of requirements; (iii) the architectural design decisions for each set of requirements, and then, for the whole solution; (iv) the technical debt issues; and (v) the human resources available in the development team as well as the time available.

Then, the following steps were conducted: **1.** the previous architecture model and its publication were analyzed; **2.** the requirements catalog [13] was discussed considering the ideas of the first architecture model and the learning goals for reducing the problems [21] in programming domain considering m-learning *apps*; and **3.** each package/layer in the conceptual architecture was studied and restructured when necessary.

The analysis took one month, where two practitioners from the industry supported the evolution of the model. Both had, in average, eight years of expertise in mobile development in industry. One of them also had three years of expertise in SPL and the other one had five years of expertise in development of hybrid *apps*, i.e., for mobile and web platforms.

We highlighted that, even with a small number of software industry practitioners, the interaction they provide is fundamental to the improvement of the proposed academic theories. In our scenario, an attempt to balance the reduced number of participants was the time spent in the reanalysis: one month was employed to conduct it. Finally, it is also important to noticed that the participants collaborated with the development of the UML models with the SMarty approach.

Discussions and the Evolved Conceptual Architecture: After the execution of each previous step, the conceptual model was evolved, as shows Figure 1.

The *Domain Engineering*, *Application Mechanism* and *Application Engineering* do not changed [14]. They remained the same, except for the way on the *apps* are generated. Such modification implies in most of the changes in the m-learning *apps* architecture.

For users, the *Application Mechanism*[14], which supports the creation of the m-learning *apps*, had small modifications: they selected the desired features creating an *app* configuration. Such configuration receives the educational content according to the features selected previously. Moreover, the complete configuration is then related to a group of users (e.g., students registered in a class), determining which features will be available to be used when the user logs in the *app*. However, with such changes, the feature selection made by the users with such permission (teachers or administrators) will be stored in a database (or metadata files) instead of integrating the source-code of a mobile *app*. This software architecture strategy is called multitenancy.

The multitenancy architecture refers to an architecture strategy in which a single software solution is executing on a server, serving multiple groups of users who share common access in such instance of software.

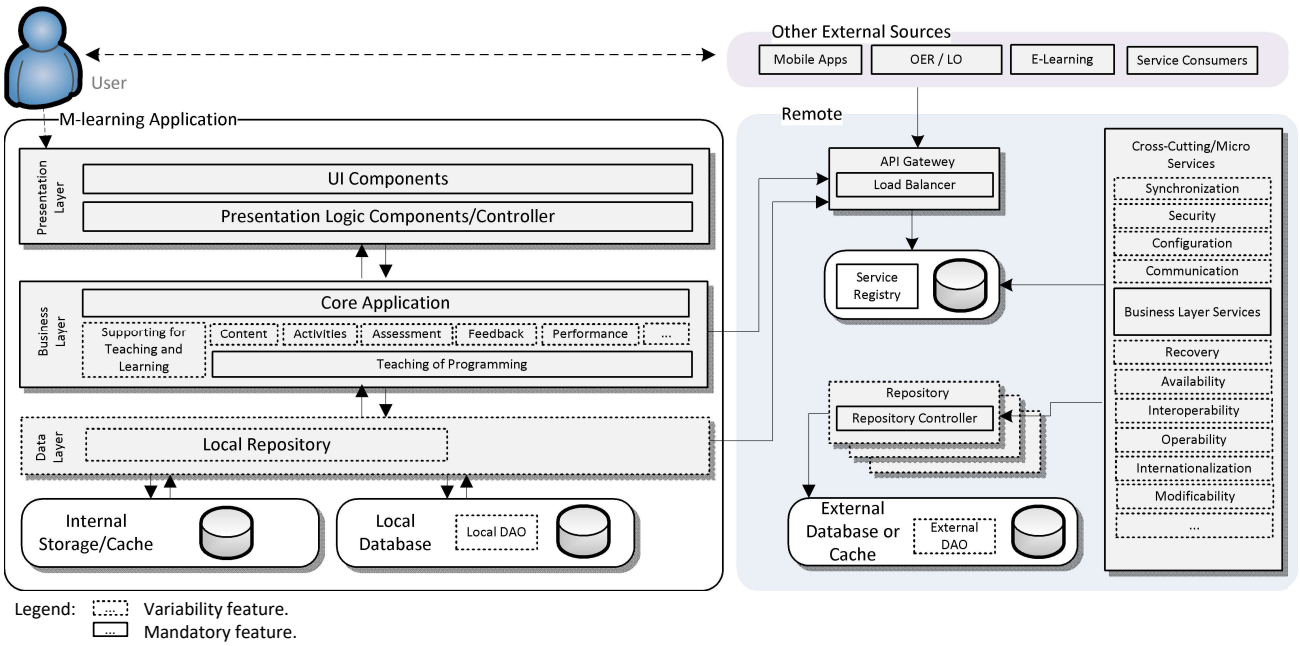

Figure 1. Evolved M-learning SPL Conceptual Architecture.

Each of these groups is called tenant. In the perspective of cloud computing models, the multitenancy approach is perfectly adherent to the concept of Software as a Service (SaaS) [9] in which the software is composed by multiple services available in different (or not) servers, being consumed by different *apps* or interfaces.

The change in the application mechanism implies in a delivery of a complete *app* instance (presentation and business layers), no matter the selected features. However, at the end, the variabilities will be solved in runtime based on the selected features, providing only features and services related to the current user.

In the multitenancy proposal, the source-code and database packages are installed on a single server, simplifying the release and scalability processes [9].

At this point, some questions arise:

Q1 – Why this decision was made? Imagine the hypothetical scenario where a teacher generated an *app* configuration for two classes to be used in one semester. After finishing the period, the teacher becomes responsible for more four classes in the next semester, but now he/she wants to use the *app* with a new feature, which was not selected in the previous semester. Until this point, there is no worries. The teacher can generate another *app* and adopt it in his/her new classes. However, what will happen with the data from the previous *app*? Additionally, if the teacher decides to change his/her choices for the configuration of the *app* in the middle of semester, there will be a need for creating a new *app* and installing it again in each student's device.

A final issue refers to the number of *apps* to be controlled based on several teachers and classes in the same institution. The data for the educational administrator will be always broken in parts, making harder the process for managing and for improvement learning and teaching processes, mainly by the specificities of each *app*. Those are only two of several implications that were identified, leading to the proposed changes.

Therefore, based on multitenancy strategy, these issues are avoided, allowing an easier creation of new configurations and the management of collected data.

Q2 – Do not these changes imply in significant modifications in functionalities of the *apps*? Yes, mainly about the connection dependency. Adopting the multitenancy strategy implies in a permanent connection, since such connection will be required for the presentation layer to show only selected features for the current user. However, cache and local database can be used to allow the usage of the *app* when a connection is not available. Such issue is one of the reasons for a new update in the architecture: the adoption of repository pattern, discussed later in this section.

Q3 – What are the main trade-offs of these changes? Positive factors: one instance in a server for multiple users, centralizing scalability issues and simplifying the management of releases. **Negative factors:** Higher development effort, mainly to maintain and to use per tenant data allowing the customization of the *apps*. Furthermore, multitenancy increases the inherent risks and impacts in applying a new release version.

As there is a single software instance serving multiple tenants, an update on this instance may cause downtime for all tenants even if the update is requested and useful for only one tenant [9]. However, this issue can be mitigated through the sending of messages for the users, warning them about the period of instability.

Nevertheless, the adoption of multitenancy implies in a new challenge: ensuring the correct execution of *apps* in devices in geographic areas where network connection is not available. To reduce this concern, as mentioned in **Q2**, the repository pattern was adopted as a solution. However, before understanding this pattern, it is important to make an observation related to the pedagogical and didactic concern: the constant feedback in *apps*.

According to Marcolino and Barbosa [14], there is a problem in the feedback provided by mobile *apps* regarding the teaching of programming. As such *apps* support more informal than formal learning, feedback features are given only for one of the parts: teachers or students. Thereby, the need of a more concise feedback for students to learn by themselves and also to notify teachers about the students' performance is a mandatory pedagogical and didactic feature to support formal learning. The feedback allows the improvement of the curricula, changes on didactic strategies adopted in classroom, changes on teaching topics and other improvements [13]. Thus, the adoption of a mechanism to store and retrieve the data of the learning process is fundamental, meeting the need for the adoption of mechanisms to enable it, as repository pattern enables.

The repository pattern avoids the directly access of data from business logic layer, services, etc. It reduces the duplicate code, weak types of business data, the difficulty in centralizing data from cache and in several data bases, and the inability to easily test the business logic. The pattern manages several data repositories, common in a microservices architecture and it also enables caching and storing data in mobile *apps*. Based on such benefits, the repository pattern was selected to deal with the multitenancy concerns, integrating a DAO (Data Access Object) strategy when appropriate, and easing the manage of microservices data – other update in the proposed SPL architecture.

The repository is shown in Figure 1. It is available in two perspectives: (i) directly in the client *app*; or (ii) in the server side. The *local repository* can performing the persistence through internal storage, i.e., private data in the device memory; or in local database, i.e., structured data in private database; and with network connection, i.e., calling a microservice. On the other hand, the server side repository is used only by microservices and performs its persistence through an external storage, e.g., in a repository as a database or cache.

Among other benefits encompassed by the adoption of repository pattern, we highlighted: (i) the pattern allows the adoption of different persistence strategies for different contexts, such as relational databases, non-relational databases, cache and services; and (ii) the *apps* may adopt managers easing the integration and encapsulation of the dependencies strategies.

Ultimately, the microservices concept was adopted, performing the integration of the SPL for the development of *apps* as a SaaS [17].

The adoption of m-learning *apps* has several benefits (Section II). However, the management of data, the availability of new contents, learning activities and other elements presented in a learning environment in the small-sized mobile screens and its native keyboards may not be a good way for conducting such tasks. In this perspective, providing alternative manners to access such functionalities in different platforms may improves the user experience and, microservices support this strategy.

Microservices allow the use of different platforms and technologies as their consumers. The responsibility is uncoupled from the business layer, which has the function of accessing the microservices available in one or more servers, providing this functionality for the presentation layer. Analyzing the SPL concept, the microservices meet and facilitate the process of management and definition of variabilities. They provide a set of fully uncoupled services, but highly collaborative. Each service implements a set of narrowly, related functions; for instance, an *app* might consist of services such as the grades management service, the users management service, and so on.

Services communicate using either synchronous protocols, such as HTTP/REST (Representational State Transfer), SOAP (Simple Object Access Protocol), or asynchronous protocols, as AMQP (Advanced Message Queuing Protocol). Also, they can be developed and deployed independently [17]. As each service has its own database, the data consistency is improved. Adding the repository standard, this will be an even more robust solution. And, considering the multitenancy architecture, just the features selected in advance will be available and visible for each tenant/user.

Based on these changes, Figure 2 depicts the new version of the conceptual m-learning *app* architecture. Comparing it with the first version, we can notice another difference: the relation between *other external sources* with the *app*. This meets what was discussed about the multitenancy and microservices strategies. Such external sources will benefit from these adoptions, since they are related with the *API Gateway*, its *load balancer* and the *service library*, which will allow the consumption of the microservices by other platforms or *apps*, improving the reuse of the non dependent features from the SPL domain.

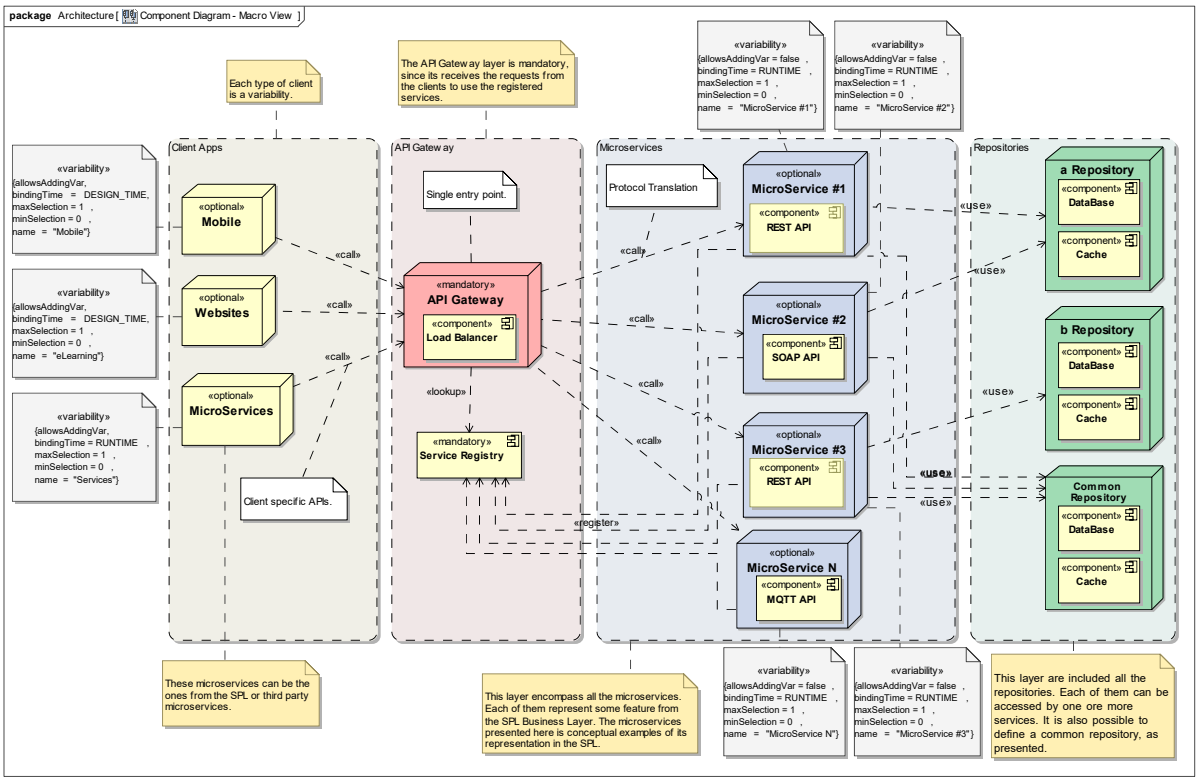

Figure 2. M-learning SPL Component Model.

The extra *service layer* was removed and the *core application* component replaced the *application facade/interface*. Now, the *business layer* has a relationship with the *API Gateway*, which balances the requests among the different tenants and *apps* to the microservices, which are, in turn, registered in a database managed by the *service registry*. Every microservice, which represents a feature in the SPL and is available to be selected by users, has a register entry in the *service registry*. Also, the *service registry* makes the register available for the *API Gateway*, responsible to redirect each requisition received from the *business layer* or *external sources* for the respective microservice.

Additionally to the benefits of adopting microservices, we highlight: (i) they are independent; if a microservice fails or becomes unavailable, the others will continue to function; (ii) they can be implemented adopting different technologies; (iii) they may have one or more repositories; and (iv) microservices can be encapsulated, functioning only with the complexity from the business domain, what ease the development, since the complexity is distributed among such domains.

On the other hand, there are also some limitations: (i) it is hard to separate business domains; considering the proposed SPL, however, this problem does not occur since m-learning requirements catalog solve this concern [13]; and (ii) the deployment of several microservices and the hardware maintenance requires more efforts than monolithic systems.

We also highlight some API Gateway benefits: (i) it is adopted as a facade layer aiming at facilitating the clients' accesses to the microservices; and (ii) it is an "intelligent" layer that can balance the load of requisitions and the caching information. However, one disadvantage is the creation of a single point of failure. If the unique API gateway fails, the entire *app* will not be executed as expected.

Having the macro view of the conceptual SPL architecture updated, the *domain design* subprocess is retaken; its models are presented next.

IV. SPL Architecture with SMarty

The SPL methodology differs from the development of singular software by the postponement of project decisions. The features are included in the software by convenience and by the need of stakeholders. In our proposed SPL, the variabilities are concentrating in the business layer features [14].

The integration of the teaching of programming domain, mobile platform and teaching and learning processes presents several features. Most of them need to be combined to provide a richer learning environment. The

several features considered in the proposed SPL come from the analysis of 81 *apps* and software systems in the programming domain (goo.gl/EvzUpB). Consequently, the catalog was represented as a feature model, depicting the relation among features, variabilities, variation points, variants and their constraints. As previously mentioned, the SMarty approach was selected to represent this in UML diagrams.

Among the domain design subprocesses, the logical view is represented through the feature model [10]. The developmental view can adopt the package diagram, component and class diagram. Figure 2 shows the component model of the SPL.

Component Architecture: The component diagram depicts how components are connected to form larger components or software systems. They are used to illustrate the structure of arbitrarily complex systems [20]. The SPL has four layers: client apps, API gateway, microservices and repositories.

Client Apps layer: this layer shows the apps, platforms, websites, services or any potential consumer of the microservices. The main clients considered in the research are mobile apps, followed by the websites. These clients have just the view and controller of the *app*. All the business logic is inserted in the microservices. However, they are the main way to connect all the desired features of the tenant/user, previously selected in the application mechanism. Additionally, all clients are considered optional variabilities, since they may or may not be part of the SPL.

In the representation, the clients are tagged as ≪optional≫ and each has an UML comment to depict additional information about them. These variables are solved in design time. Each instance of the client is called *API Gateway* node from the next layer.

API Gateway: this layer identifies the available microservices registered in a database through the mandatory service registry component. The API Gateway node, which is also mandatory, looks up for the microservice requested by the client in the service layer. If the microservice is found, the requisition is broadcast for the respectively microservice in the next layer. Additionally, the load balancer observes the behavior and quantity of requests from the clients, balancing the load of data traffic and ensuring that all requests will be answered, controlling them in a row. This component is also responsible for increasing the server resources; i.e., it provides the elasticity of the cloud resources [17, 20].

Microservices layer: this layer encompasses all the microservices, although each one can stay hosted on different servers. The microservices have specific functionalities, from non functional to functional ones, and each represents the features depicts in the feature model. They are instances of each selected feature in the application mechanism. The model presents four samples of services. Everyone has a component that expresses the type of supported protocol and, based on this protocol, each one receives requests directed by the API Gateway.

Also in this layer, each microservice is tagged as an optional variability, but following the features model; some of them will be represented as exclusive (≪alternative_XO≫) or be mutually exclusive among the other selected microservices. For instance, educational activities can be available following a fixed order or be performed by convenience; therefore, these two features regarding the availability of educational activities follow the SPL mutually exclusive constraint. On the other hand, there are features that require others, such as the learning monitoring and performance, that needs feedback features to support both teachers and students.

The other behavior of the microservice layer is the possibility for microservices to make calls among them, when they depends on the process of another microservices, removing the need of processing from the client. These calls are also performed by the API gateway.

Repositories Layer: in this layer all repository structure is presented. They can be on different servers, not being mandatory for some services. The repository is responsible for the management of the persistence process, providing interfaces in DAO model, if convenient, for enabling the access of the data in their registered database. They also coordinate the cache usage.

Architecture Class Diagram: The next architecture artifact from the SPLE framework is the class diagram (https://bit.ly/2zz4Pw1). For this representation, as each microservice represents a feature or a small set of them, the class diagram was divided in one representation for each existing artifact. This decision was made based on some issues, particularly: (i) the reduced number of developers, (ii) the reduced time for the development of each feature before its development, (iii) the facility to manage each diagram of the architecture as an artifact from the SPL core asset; and (iv) the eXtreme Programming (XP) agile process with the Test Driven Development (TDD) adopted for the conduction of domain realization and domain testing SPLE subprocesses. We highlight that the adoption of XP and TDD has been investigated in order to provide evidence of their possible benefits in the SPL context.

The class diagram (https://bit.ly/2zz4Pw1) shows the content feature microservice represented on a class diagram. The diagram considers two instances of clients: a Mobile App and a Web Site. Both call the content API Gateway, which looks up in the Service Registry to identify if the requisition for such microservice can be answered. For this, the respective called microservice needs to be registered on the queried database and be identified with the support of a repository. After the

API Gateway localize the service through its registered URI, it executes the customer request and returns the retrieved content data.

Three variants are represented in the class diagram. The two clients (*MobileApp* and *WebSite*), which both are solved in design time; and the *Content Microservice*, which is solved in runtime. The content microservice is considered a variable because some *apps* can be developed only to provide activities for the students, not being mandatory for every *app*. The difference between the binding time of the variabilities is related to the type of variable. The clients need to have their presentation and business logic layers developed before, since they will consume the available microservices accordingly to the permission of features selected for users on the *app* generation mechanism. The availability of the microservices, in turn, will occur at runtime.

Lessons Learned: Based on the tasks and decisions made in this work, two lessons can be highlighted:

(1) Evaluating the requirements catalog before the first conceptual model architecture without an integration of both could be resulted in future problems in the development phase. This was made because of the order of research phases. Nevertheless, this threat was mitigated based on the practitioner's analysis.

(2) Microservices improve the reuse and, in some cases, can be adopted in other software programs independently of the domain. Additionally, the reduced size of functionalities expressed on them enables the creation of a greater number of configurations of *apps*. In this perspective, other microservices can be easier integrated into our solution and even our microservices can be reused in other contexts. However, the evolution of the line will need more attention, since the communication among these microservices requires more efforts and tests to guarantee that they will work properly.

Related Work: Comparing the SPL SOA models proposed in the literature with the SPL presented in this research, we can notice that our models are more feasible to be adopted. Gunther and Berger present a development process for an SPL service oriented. The authors describe through a WSDL (web service description language) the proposed constraints for the services [6]. However, the domain used to show their proposal is a web store. Furthermore, it is necessary to study the WSDL to understand the architecture proposed.

Istoan, in [8], presents an SPL for creating software oriented *app*, more specifically, a process based on BPMN (Business Process Management Notation) to allow the development of *apps*. Despite the potential of the proposal, the presented models are restricted to feature models and BPMN diagrams, and the domain is related with building automation context.

Aiming to identify service oriented projects with SPLE, Mohabbati et al [16] conduct a systematic mapping study. From the 81 primary studies identified, only one was related with educational area. This study, however, did not present enough discussions about the proposed architecture to make it adoption feasible.

Finally, aiming at identifying SPLs in the specific domain of teaching of programming, Marcolino and Barbosa [12] also conducted a systematic mapping study. Only 10 SPLs were identified and just one integrates SOA. This study was adopted in the first version of our SPL (Figure 1).

The discussed models (Figure 2) were developed to present a more feasible view of an architecture project, easing its adoption or modification by other researchers. Additionally, the contributions obtained by the analysis conducted by practitioners allowed the integration, not only of SOA, but also of other industry strategies that support the selected educational domain. However, considering the adopted strategies, it is possible to generalize such models to be used in other learning domains.

Research Questions:Based on the analysis conducted with the practitioners, it was possible to answer the three research questions:

1. Can the proposed SPL architecture model be somehow improved by the analysis of software industry practitioners? Yes. The industry practitioners' expertise was significant to improve our proposal. It is important to mention that both practitioners are graduate in technological area, what implies that their knowledge is also founded on academic theories. On the other hand, considering the needs of stakeholders in the software industry, they also developed the capability to integrate different approaches and methodologies, as seen in the discussions presented. It was possible thereby to evolve the main points that were evaluated in a more low-grained models.

2. Which points of our SPL architecture, proposed with basis on academic theories, need improvement according to the software industry practitioners? Mainly the points related with installation, update, availability, and data and users management. As such features are related with all requirements, it drives improvements in the process of how the *apps* can solve the SPL variabilities in runtime, being adapted according to the selected features.

3. What software industry strategies can be applied in the development of the proposed SPL and in their features for m-learning *apps*? The integration of multitenancy, SOA, repository pattern and API gateway strategies allowed for proposing a robust architecture that satisfies all the requirements from the teaching and learning domain, including the specific requirements of pedagogical and didactic concerns. Additionally, the presented models may be easily

adapted for different learning domains. On the other hand, the adoption of only one API gateway and the need to create mechanisms to provide the execution of the *app* when the Internet connection is not available become challenges to be overcome.

Finally, it was possible to identify that the integration of practitioners in projects developed based only in academic theories can bring significant improvements to the solutions, ensuring the reduction of efforts in correcting possible mistakes occurred in design phases. Additionally, in comparison with other proposals from literature, our SPL architecture provides a detailed and easier model to be adopted in other researches, mainly because for their representation well-known notations were adopted.

V. Conclusions and Future Work

In this paper we discussed the evolution of an SPL conceptual architecture for the development of m-learning *apps* for the teaching of programming, based on the analysis of software industry practitioners. The first version of our model considered just theoretical and academic views in its evaluation. To provide a more practical perspective, two practitioners were integrated in our research, giving contributions to the improvement of the proposed SPL architecture.

Based on the evolved model, the component architecture diagram was presented. The model encompasses the multitenancy concept, microservices and repository pattern; the content feature was represented with UML diagram and the SMarty approach was used to exemplify the representation of each set of the features of the SPL.

All decisions and models proposed lead to a possible solution for some problems faced on the domain of the teaching of programming through m-learning *apps*. Furthermore, considering the practitioners' perspective, it was possible to answer three questions that guided this research, actually showing how important is the partnership with industry into creation of more complete solutions, and easier to be adopted.

As future work we highlight: (i) analyze the results in a qualitative study, with teachers of programming, in order to define the initial set of features to be firstly developed; and (ii) evaluate the new version of the SPL conceptual architecture to assure its potential for the development process, as well as to investigate the generalization of such models, allowing their adoption in other researches and domains.

Acknowledgment

The authors acknowledge the Brazilian funding agencies: CAPES (Process DS-8907173DT and Procad 071/2013), FAPESP (Process 2014/03389-9) and CNPq.

References

[1] V. C. Oliveira A. and P. C. A. R. Tedesco. Ensino e Aprendizagem de Programação para Iniciantes: uma Revisão Sistemática da Literatura focada no SBIE e WIE (*in portuguese*). *Brazilian Symp. on Inf. in Educatio*, 2012.

[2] M W Chiasson and E Davidson. Taking industry seriously in information systems research. *Mis Quarterly*, 2005.

[3] C. Duncan, T. Bell, and Steve T. Should your 8-year-old learn coding? In *Proceedings of the 9th Workshop in Primary and Secondary Comp. Ed.*, pages 60–69, NY, USA, 2014.

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

[5] Fáber D Giraldo, Sergio España, and Óscar Pastor. Evidences of the mismatch between industry and academy on modelling language quality evaluation. 2016.

[6] S Günther and Thorsten B. Service-oriented product lines: Towrads a development process and feature management model for web services. In *SPLC (2)*, pages 131–136, 2008.

[7] Vo Ngoc Hoi. Understanding higher education learners' acceptance and use of mobile devices for language learning. *Computers & Education*, 146, 2020.

[8] P Istoan, J Jézéquel, and Gilles P. *Software product lines for creating service-oriented applications*. PhD thesis, Irisa Rennes Research Institute, 2009.

[9] Rouven K, Cf Momm, and S Kounev. Architectural concerns in multi-tenant saas applications. *Closer*, 12:426–431, 2012.

[10] Frank J. Linden, Klaus Schmid, and Eelco Rommes. *SPL in Action: The Best Industrial Practice in Product Line Engineering*. Springer, Berlin, 2007.

[11] A. Marcolino, E. OliveiraJr, and I. M. de Souza Gimenes. Variability Identification and Representation in Software Product Line UML Sequence Diagrams: Proposal and Empirical Study. In *2014 Brazilian Symp. on Soft. Eng., Maceió, Brazil, Set. 28 - October 3, 2014*, pages 141–150, 2014.

[12] A. S. Marcolino and E. F. Barbosa. Linhas de Produto de Software no Domínio Educacional: Um Mapeamento Sistemático (*in portuguese*). *Brazilian Symp. on Inf. in Education*, 1:239–249, 2015.

[13] A. S. Marcolino and E. F. Barbosa. Towards an m-learning requirements catalog for the development of educational applications for the teaching of programming. *In 2016 IEEE Frontiers in Education Conf., 2016.*, pages 1–5, 2016.

[14] A. S. Marcolino and E. F. Barbosa. Towards a software product line architecture to build m-learning applications for the teaching of programming. *In 50th Hawaii International Conf.e on System Sciences, 2017, Kauai (HI)-EUA.*, pages 6264–6273, 2017.

[15] A. S. Marcolino and E.n F. Barbosa. Softwares Educacionais para o Ensino de Programação: Um Mapeamento Sistemático (*in portuguese*). *Brazilian Symp. on Inf. in Education*, 1:190–199, 2015.

[16] B Mohabbati, M Asadi, D Geviera, M Hatala, and H. A. Muller. Combining service-orientation and software product line engineering: A systematic mapping study. *Information and Software Technology*, 55(11):1845 – 1859, 2013.

[17] Sam Newman. *Building Microservices*. O'Reilly Media, Inc., 1st edition, 2015.

[18] Paul Piwek and Simon Savage. Challenges with learning to program and problem solve: An analysis of student online discussions. 2020.

[19] K. Pohl, G. Bockle, and F. Linden. *SPL Engineering Foundations, Principle, and Techniques*. USA: Springer-Verlag, 2005.

[20] Roger Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, Inc., NY, USA, 8 edition, 2015.

[21] D. M. Souza, M. H. S. Batista, and E. F. Barbosa. Problemas e Dificuldades no Ensino de Programação: Um Mapeamento Sistemático (*in portuguese*). In *Revista Brasileira de Informática na Educação.*, 2015.