A Domain Engineering for Interactive Learning Modules

Danilo L. Dalmon and Leônidas O. Brandão

Instituto de Matemática e Estatística, Universidade de São Paulo
Department of Computer Science
Rua do Matão 1010, Cidade Universitária, São Paulo, SP - Brazil
Email: ddalmon@ime.usp.br, leo@ime.usp.br

Anarosa A. F. Brandão

Escola Politécnica, Universidade de São Paulo
Department of Computer and Digital Systems Engineering
Email: anarosa.brandao@poli.usp.br

Seiji Isotani

Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo
Laboratory of Software Engineering
Email: sisotani@icmc.usp.br

For creating educational systems, each developer usually applies different approaches to specific situations, including advanced software engineering techniques. In the case of systems that provide interactivity-intense assignments, problems during their development include difficulties to manage component repositories and the absence of a systematic process to support code reuse. Interactive Learning Modules (iLM) are systems designed in this context, which also provide key functionalities to facilitate the teacher's work. To address the problems faced during theirs development, a Domain Engineering is proposed to build iLM. In order to define the iLM software family, this paper presents the core features of existing systems, describes the method used to produce an application framework and how to instantiate it. Restructuration of existing iLM using the proposed Domain Engineering is reported with initial high gains in productivity and system quality.

Classification: D.2.2 Design Tools and Techniques; D.2.11 Software Architecture; D.2.13 Reusable Software; K.3.1 Computer Uses in Education.

Keywords: Educational Software; Domain Engineering; Application Framework; Interactive Learning Modules.

1. INTRODUCTION

The problem of developing educational software is not new. Nicolson and Scott (1986) analyzed educational software development cycles and reported that the technology developed is often inadequate for both teachers and learners. Educational Software are tools created to be used by students and teachers, and they are expected to improve students' knowledge about some subject and/or help teachers to produce better educational content (Tchounikine, 2011).

In order to establish a clear goal for any educational software, the literature points out the importance of interactivity to increase students' motivation and learning outcomes. For instance, many authors address the improvement in students' understanding through their interactivity with the learning content (Isotani and Brandão, 2008, Kortenkamp, 2004, Sims, 1997, and Ruthven, Hennessy and Deaney, 2007). Also, interactivity through quick feedback for each student's response is important to keep them motivated (Hentea, Shea and Pennington, 2003, Hara and Kling, 1999).

Interactive Learning Modules (iLM), formerly known as e-LM (Rodrigues, Brandão and Brandão, 2010), are educational software that focus on these interactive features and the main characteristics of which are: running in web-browsers, presenting authoring tools, promoting interactivity-intense assignments and

providing communication with Learning Management Systems (LMS) under a simple protocol (Dalmon, Isotani, Brandão and Brandão, 2011a).

Developing educational software is expensive and time-consuming. Moreover, to produce new systems that provide interactivity demanded by teachers and educators may imply higher costs, related to specifying, developing and testing the system with the final users, teachers and students. An approach to reduce costs and improve maintainability during software development is code reuse. Nevertheless, without a systematic process to support code reuse, this approach could be ineffective, even considering a small group of people that share almost the same knowledge. This was occurring in our research group, in which five iLM were developed: *iGeom* (Isotani and Brandão, 2008), *iGraf* (do Prado, 2006), *iCG*, *iComb* (Eisenmann and Brandão, 2009) and *iVProg* (Kamiya and Brandão, 2009) without the adoption of a systematic software process and people organization.

In addition, the development of educational systems has some differences from other specific-area software in some aspects (Tchounikine, 2011), which makes code reuse difficult (Spalter and van Dam, 2003). These differences include components with numerous granularities, self managed repositories, ill-defined quality criteria and social issues, such as team organization and project management. Many attempts, including applying software engineering techniques, were made to solve these problems (Roschelle, Kaput, Stroup and Kahn, 1998, Spalter and van Dam, 2003, Ateyeh and Lockermann, 2006), but some issues still remain, such as systematic use and reuse of components available in repositories, limited interactivity provided, and underlying pedagogical approaches that prevent teachers from using systems in broader educational contexts.

In order to overcome many of these problems, a Software Product Line (SPL) approach was chosen for developing the iLM software family. The intended outcome of this research is to provide a Domain Engineering for improving the development of these systems. This technique defines team organization and a systematic development process to improve reuse and quality of a group of similar software (Clements and Northrop, 2001). Domain Engineering can be used to: (i) design the refactoring of a set of existent iLM; to (ii) develop new iLM; and to (iii) add new domain-independent features to them. An application framework was used to manage variability in Domain Engineering design and implementation (Dalmon et al., 2011a).

Our main goal is to report the development of a Domain Engineering for a specific family of educational software, the iLM, in order to highlight lessons learned and development results. It is not our goal to contribute to the SPL field of research. The work presented is inserted in a research project the long term goal of which is to develop and to evaluate a SPL for iLM. This evaluation will be possible after several years of SPL use so there will be data about software quality, development, maintenance and evolution under the influence of this technique. Currently, the first version of Domain Engineering is completed and Application Engineering is starting, which provides data about the initial influences of the SPL technique on the maintenance of existing iLM.

The text is organized as follows. Section 2 presents some approaches from the literature to the development of educational software, highlighting the main challenges focusing on code reuse. Section 3 describes existing iLM as a product family. The method and the development of the Domain Engineering are detailed respectively in sections 4 and 5. Section 6 reports current Application Engineering results and a first evaluation. The paper ends with some conclusions and future directions.

2. EDUCATIONAL SOFTWARE DEVELOPMENT AND CODE REUSE

Developing high quality educational software is expensive, time-consuming and is usually done in exceptional contexts, such as research projects at universities, by students without professional experience (Spalter and van Dam, 2003). Studies indicate that, in some cases of highly computer-managed learning, it can take more than 100 hours of programming for each hour of instruction (Aleven, Mclaren, Sewall and Koerdinger, 2008). For being influenced by different factors in design and development in comparison with regular enterprise systems, educational software developers elaborated specific techniques for code reuse. In this section, we present some related work that report several of these factors and techniques.

Software used for educational purposes can be divided into three groups (Mor and Winters, 2007): (a) those intended to organize and to present content, such as LMS; (b) those intended to provide assignments, such as tutors and simulators; and (c) those not intended to be educational despite being used as such, such as text editors or Internet search engines. This categorization serves mainly to restrict the context of this analysis. This paper relies on the group of educational software which provides interactive assignments.

In fact, we are interested in the development of systems that provide domain-specific features to support teachers' everyday work, as well as interactivity-intense assignments to students. By domain-specific features we mean functionalities that use domain specificities in system design in order to improve learning. By interactivity-intense assignments we mean the possibility of interacting with the assignment through specific types of interactivity, e.g. manipulate, construct and support (Sims, 1997, Tang, 2004). Manipulate interactivity allows the modification of domain-specific objects to simulate behavior and then make conjectures and tests. Construct interactivity allows the creation of objects before manipulation or simulation. These types of interactivity are essential to constructivist problem-based learning approaches (Albanese and Mitchell, 1993, Savery and Duffy 1995), which we value the most. Additionally, the support type of interactivity, which feedbacks students on their performance, provides many benefits to teachers and students (Tang, 2004). The feedback could be "right or wrong" information (Isotani and Brandão, 2008) or could be a more sophisticated dialog, such as those provided by Intelligent Tutoring Systems (Aleven et al., 2008).

A literature review shows several approaches to facilitate code reuse during the development of educational software. In the context of the ESCOT Project (Roschelle et al., 1998, Roschelle, Digiano, Chung, Repenning, Tager and Treinen, 2000), the task of authoring educational components was divided among teams of developers and teachers. Therefore, code reuse was based on the search for environments and authoring tools for assembling software components, which resulted in using a JavaBeans (Oracle, 2012) component repository to create small applications to a broad extent in middle and high-school curricula. Problems related to the development of educational software and their usage were reported by Spalter and van Dam (2003). Among others, they have identified problems related to team organization, the lack of programming skills and interoperability among components. Moreover they have pointed out issues related to the growth of intellectual property and problems of software usage for pedagogical purposes in schools. To overcome such problems, Spalter and van Dam (2003) have proposed a family of educational software called *Exploratories*. Each *Exploratories* application was built to support a specific knowledge domain, such as the circulatory system in Biology, and was (recursively) composed of code components in order to address interoperability and flexibility issues. However, the goal of independent sub-applications available on the web, where a regular teacher would assemble them for classroom use, was still far away.

In the same direction of using components and repositories to foster reuse, Bote, Hernandez, Dimitriadis, Asensio, Gomez and Vega (2004) worked with IMS Learning Design – IMS-LD (IMS, 2003) specification to make components interoperable and grid-based techniques to build a family of collaborative educational systems to provide domain-specific assignments to a computer architecture course. They adopted an approach in which the teacher could enter his or her students' data and tailor the tool by using grid scripting to assemble IMS-LD components. On the contrary, our approach tries to reduce the amount of teacher work by removing the need to learn complex technologies, such as grid scripting.

An alternative method to foster reuse is to design a development process to create flexible components that can be reused in different educational contexts (Douglas, 2001, Muzio, Heins and Mundell, 2002, Boyle, 2003, Polsani, 2003). Thus, the whole system is reusable instead of just parts of the code. Generally they are supported by repositories of Learning Objects (LO) and other educational systems (Richards, Mcgreal, Hatala and Friesen, 2002, Nash, 2005). Pankratius, Stucky and Vossen (2005) proposed the reengineering of existing disconnected LO using Aspect Oriented Programming – AOP to reuse code during LO development. AOP was used to describe original LO and how to use them, or parts of them, to create new LO. These approaches solve many reuse problems, provide benefits to programmers and designers, but they are far from the teacher's everyday work.

The example of LO used by Pankratius et al. (2005) is related to digital information products. This kind of LO fits in the group of educational systems that intend to organize and present content. A significant effort has been spent on developing techniques to increase the quality of digital information products and to reduce

the effort to develop them by reuse. These products do not have the same educational goals or computational structure as assignment-based educational software such as iLM. Nevertheless, some contributions from these works can be extended to the context of developing assignment-based systems, such as methods to foster code reuse and lessons learned from facing difficulties with managing people and setting requirements.

IMS-LD specification was also used to create a family of similar tools using a model-driven generative approach (Dodero and Díez, 2006). A feature model was used to list the available features that were mapped into models through IMS-LD files, which could be shared as LO. Additionally, the same specification was used during the instantiation of a framework for reengineering presentation of educational software (Choquet and Corbière, 2006). This framework models learning scenarios that can get existing systems to create new and more organized ones.

AOP was also adopted to provide reuse of content-intense systems (Ateyeh and Lockermann, 2006). In this work, AOP was applied to woven aspects similarly to those of a SPL, with domain engineering and course engineering (which has the role of application engineering in a SPL) detached. Domain engineering has been modeled using an ontology, and in each step there were content and didactics components. A courseware authoring tool was developed to produce LO, such as LMS modules and digital information products. Oberweis, Pankratius and Stucky (2007) proposed the full use of SPL to the creation and maintenance of digital information products. In the case of e-Learning Systems, Ahmed and Zualkernan (2011) reported a SPL that is structured to create systems that model entire courses on a given subject, using components repository and a process based on different views of assets.

In addition to code reuse, there are different educational software design proposals that consider contributions from other aspects, for instance, from content experts, teachers, instructional designers, institution practices, curriculum and usability (Mayes and Fowler, 1999, Douglas, 2001, Muzio et al., 2002, Sampson and Karampiperis, 2006). Furthermore, the related literature also highlights other aspects. An educational system should not be user-centered, in which the "user" has a domain knowledge and uses it with the system, but learner-centered, in which the "user" aims to develop a domain knowledge by using it (Quintana, Krajcik, Soloway and Norris, 2002). This implies deep graphical user interface and content structural changes. Interactivity is very important to educational software, and different types of interactivity must be considered to foster different types of learning (Tang, 2004), on which the whole system is based. Also, the tool must be in line with teacher's instructional practices, and not establish new ones (Hinostroza, Rehbein, Mellar and Preston, 2000), since this could alter the system most basic requirements.

Finally, the literature presents many uses of software engineering techniques to enhance educational software development. These approaches contributed to our research with lessons learned and challenges to be faced. They propose the development of educational software only intended to present content while they do not support interactive assignments which is a key feature for such systems. Nevertheless, it is possible to learn from these approaches how to support code, architecture and process reuse. Unfortunately, few of them consistently report pros and cons, evaluation methods, or differences in development contexts (systems size, team size, team experience), limiting knowledge capitalization (Tchounikine, 2011).

In order to contribute to the field, our proposal adopts the SPL approach to deal with code, architecture and process reuse during the development of a family of educational software that is intended to provide domain-specific assignments with different types of interactivity, e.g. manipulate and construct interactivity (Sims, 1997). Such a family is described in the next section.

3. THE SOFTWARE FAMILY OF INTERACTIVE LEARNING MODULES

Our research group develops educational software intended to improve the learning and teaching processes mainly by facilitating teachers' work and providing interactive assignments to students. These systems, called iLM, were developed following some guidelines to be compatible with a communication protocol of LMS (Rodrigues et al., 2010), but they were also conceived as single applications.

Prior to this work, five systems were built within the iLM family: *iGeom*, an Interactive Geometry System (Isotani and Brandão, 2008); *iGraf*, a system to study mathematical functions and graphics (do Prado, 2006); *iCG* is a system that emulates a computer with an embedded compiler; *iComb*, an educational counting

system (Eisenmann and Brandão, 2009); and *iVProg*, a visual programming system (Kamiya and Brandão, 2009). Fig. 1 shows some of their screens.

Although their development had followed some guidelines, they did not follow a systematic development process. For instance, *iGeom* has been developed since 2000. Currently, with five developers working on *iGeom*, it is at version 5 and it has 46,000 lines of code. The first deployment of *iComb* occured in 2007 and it had 7500 lines of code written by two programmers. At present, the second version is being developed by a third developer. Therefore, their maintainability has become a hard task and the adoption of software engineering techniques is imperative for the group research and development success.

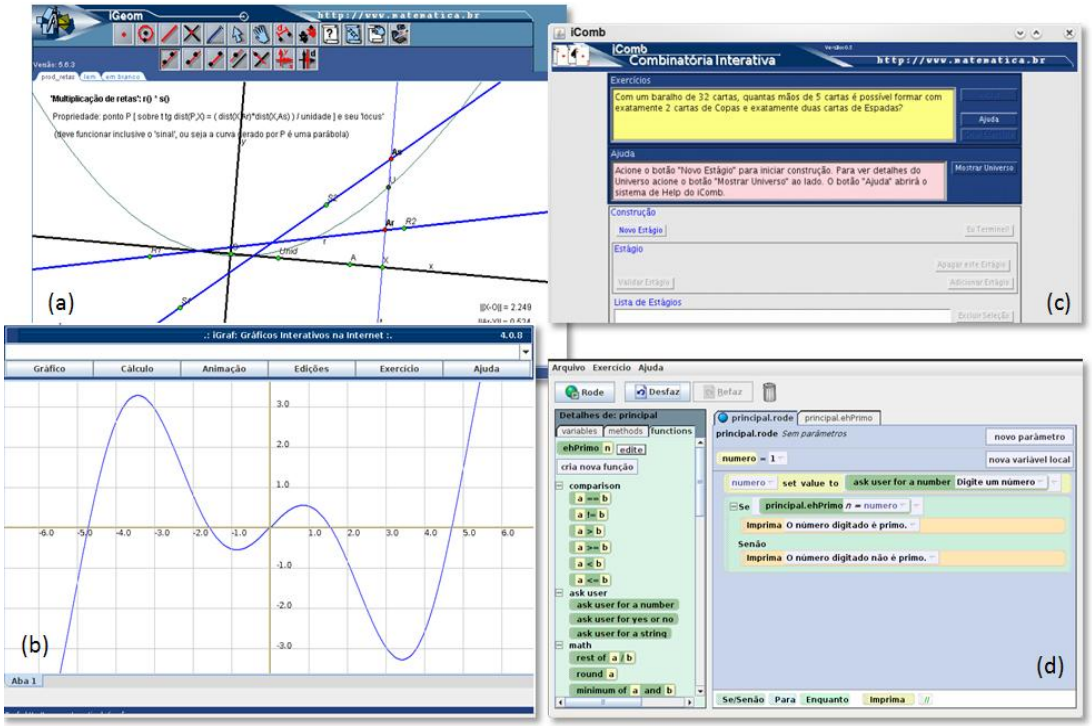

Fig. 1: Screenshots from iLM: (a) iGeom, (b) iGraf, (c) iComb and (d) iVProg.

In order to decide what software engineering technique to adopt, a systematic analysis of existing iLM was conducted and some commonalities among them were observed so that we could characterize them as a family of similar software products. Thus, we decided to apply a SPL approach for the development of new iLM and the refactoring of existing ones.

During the systematic analysis of iLM, their core features were identified (Dalmon et al., 2011a):

- *Run in web-browsers* – by running in web-browsers, iLM are platform independent and can be used in classrooms as well as in distance education.
- *Communication with LMS* – under a simple protocol, iLM communicate with LMS in order to (a) open assignments files and (b) send the student's assignment solution back to the LMS.
- *Authoring tools* – to allow the creation of assignments and the choice of a pedagogical approach. Usually, educational software adopts an underlying pedagogical approach that may limit the systems flexibility and content reuse for teachers.
- *Interactivity-intense assignments* – systems of the iLM family provide interactivity-intense assignments since users can create or modify domain-specific objects within assignments to simulate a behavior and then make conjectures and tests.

Besides the core features, we had identified some optional features exploited by the learning and teaching processes. Although these features were included in the SPL core, we kept them optional.

- *Automatic checking of assignments* – as an additional interactivity feature, feedback is one of the most important features in educational software (Hentea et al., 2003). In assignment-based systems, such as iLM, an automatic assessment feature can help both teachers and students. By using an iLM with such a feature embedded in an LMS, it is possible to store all the students' results, and students can profit by checking whether their solution is correct. This raises students' motivation and release teachers from the task of checking their assignments (Isotani and Brandão, 2008).
- *Intelligent tutoring* – Intelligent Tutoring Systems – ITS bring many benefits to educational practices (Koerdinger and Corbett, 2006). By using ITS features during the authoring of iLM assignments, teachers can increase the amount and quality of feedback interactivity provided by the system. In spite of being suggested, this feature is under development for an existing iLM (Dalmon, Brandão and Brandão, 2011b).

After specifying and defining the iLM software family, we detail the scope of the SPL approach in the next section.

4. DOMAIN ENGINEERING METHOD FOR ILM

This section presents the method used to develop the Domain Engineering of a Software Product Line (SPL) for iLM in order to allow a discussion of its influence on the development process in next sections. A SPL consists of methods and tools to define and to guide the creation of iLM, fostering code, architecture and process reuse (Clements and Northrop, 2001). It is composed of two parts, Domain Engineering in which code, architecture and process are developed for reuse, and Application Engineering, which makes use of code, architecture and process from Domain Engineering to develop iLM. The main goals of applying this technique for developing iLM are to reduce the effort during systems creation and maintenance, to improve systems quality and to provide systematic processes for design, implementing, testing and documenting.

The SPL approach affects four aspects of development: business, architecture, process and organization (Linden, Schmidt and Rommes, 2007). Business sets the main goals of the products that will be created by the SPL. Architecture defines the systems code structure, and where code reuse is more evident. Process establishes the methods, activities and steps along the development; and organization sets the tasks attribution to developers. Our focus lies on the architecture and on process aspects. The business aspect is defined as the approach taken by the research group, which is mainly to contribute to the community by providing open source tools, while the main goals of the iLM family is helping teachers and students. The organization aspect is defined by delegating tasks to graduate students and programmers with scholarships. Currently, the group consists of seven people, one for each iLM, one for the SPL, and two newcomers as additional programmers for *iGeom* and *iVProg*.

The method applied to the Domain Engineering for iLM is a slightly modified PLUS method (Gomaa, 2004). It was chosen mainly considering the team's experience in UML-based software design and the availability of documentation, as there is no SPL expert in the team. PLUS states that the features of the aimed systems must be listed and classified as core (compulsory), variant (that varies from a range of possibilities) and optional (that can be chosen or not). Therefore, during application engineering, developers must choose which possibility of each variant feature and whether the system will have the optional features. Later, during the analysis step, classes must also be classified as core, variant and optional, which must reflect the previous features classification.

The change in PLUS was made in order to adapt the method to the iLM domain, because of the variability encountered among existing systems. Features are categorized as core, alternative or optional, but classes are only categorized as core or variant. This is due to the fact that in our systems each class only implements one feature, thus they can only be of types core or variant (mainly user interface and domain model classes), no matter whether the feature is core or optional. This simplifies implementation of individual components. Additionally, when core and optional features have only alternative behaviors, such as user interfaces, they are not considered alternatively in order to simplify the design and to separate user interface implementation from the other classes.

Moreover, the Spiral model of software development process (Bohem, 1986) was integrated to the Domain Engineering process, to define how the PLUS method would be applied. Fig. 2 depicts a schematic view of the used process. First, we defined the Domain Engineering scope, by describing the system family, the core requirements, core and optional features that will be provided as the preliminary design. Then, an iterative development step, including software analysis, design, implementation and testing, is performed to build the first Domain Engineering prototype. With this prototype, it is possible to start the development of iLM in Application Engineering. Whenever application engineering sets new requirements and features for Domain Engineering, it restarts an iterative development step, in order to provide new prototypes, or releases.

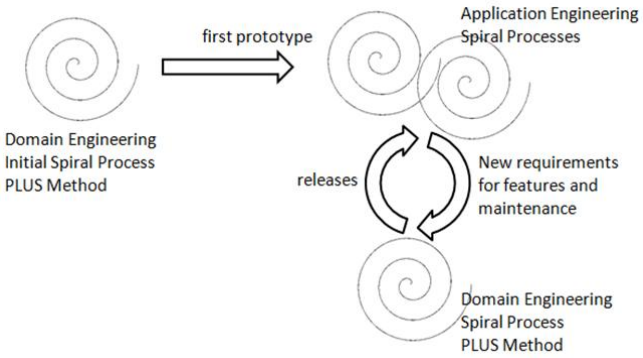

Fig. 2: Schematic view of the process used: SPL with PLUS method and Spiral models.

In this context, the PLUS method utilization started by listing the non-functional requirements of existing systems, then a feature analysis was made, categorizing the features into core or optional. Later, some desired features were added and categorized. The next step was use cases description for all features. During the analysis phase, UML class diagrams were created, categorizing classes into core or variant for each feature, followed by UML sequence diagrams for each use case. During design, groups of classes within features were organized into components, before implementation and testing.

An application framework was chosen for the Domain Engineering implementation. In this case, application engineering consists of the framework instantiation and specialization. Variability management at code level is performed with two types of variability points in the framework for building iLM: (a) groups of components that form an optional feature must be chosen, or not, to be part of the system; and (b) variant classes within components that define specific behaviors of the system by inheritance. Therefore, during application engineering, developers must choose which features the system will have, and the behavior of each of them by inheriting abstract classes of the framework.

In the next section, we describe the development of Domain Engineering, represented by the first version of the framework, and the detailed methods for application engineering.

5. DEVELOPMENT OF DOMAIN ENGINEERING FOR ILM

This section presents the development of Domain Engineering for building iLM. According to the Spiral process and PLUS method described in the previous section, it was conducted in four steps: requirements analysis, feature analysis, use case analysis and an iterative step of design, implementation and testing, resulting in the framework prototype. The four steps will be detailed next.

5.1. Requirements Analysis

During the preliminary design and the requirements phase, regarding the framework development, the analysis of desired and existing requirements resulted in the list below. Application engineering considers additional requirements, which are described later.

- *Documentation* – a Domain Engineering framework must be well documented so that it is possible to maintain it during domain application and to use it easily during engineering application.
- *Extensibility* – the framework for building iLM must be capable of absorbing new features and specific features of iLM as the family of systems grows.

- *Maintainability* – existing iLM have poor maintainability. One of the main goals of using Domain Engineering approach is to improve the code quality, which is key in this work.
- *Open source* – our project will follow an open source approach in order to provide free tools to educational institutions and to allow a community to develop iLM.
- *Platform compatibility* – they must be compatible with any operational system in order to be used by any educational institution.
- *Resource constraints* – some educational institutions in developing countries have outdated computational resources; hence, iLM must be lightweight and demand low processing power.
- *Reusability* – as Domain Engineering, the framework most important requirement is code reusability.
- *Testability* – to ensure maintainability and reusability, the framework code must be highly testable.

Existing iLM explicit requirements were only platform compatibility, resource constraints and open source. The additional requirements defined in this research mainly reflect systematic reusability in a software life cycle longer than two years.

5.2. Feature Analysis

The features of the iLM family were defined by mapping functionalities of existing iLM and some additional ones. Fig. 3 shows a feature model with the core and optional features described in section 3, divided into two levels of sub-features.

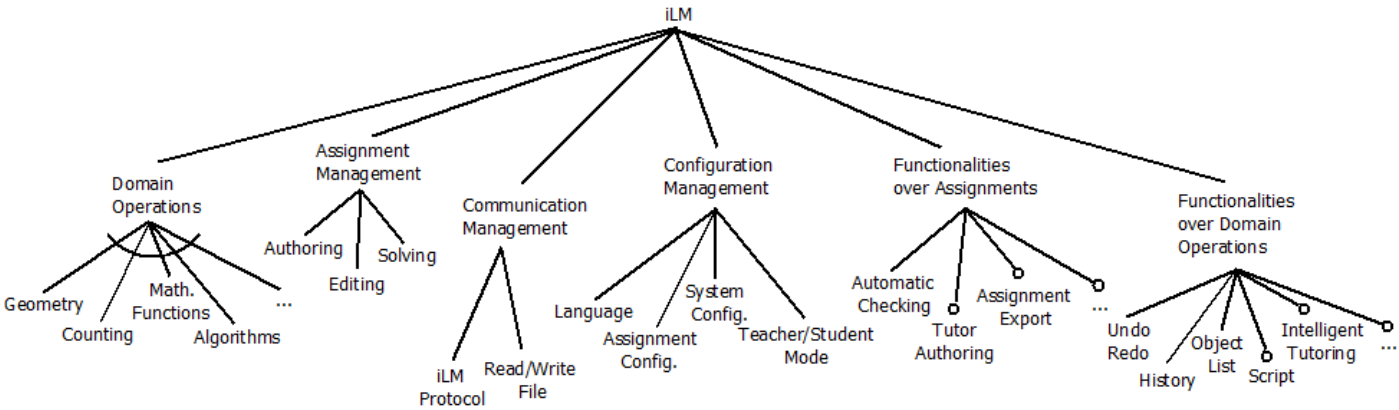

Fig. 3: Feature model of the framework for building iLM.

This diagram follows the FODA, *Feature Oriented Domain Analysis* (Kang, 1990) notation, in which an arc between features means *alternative* features and a circle means *optional* feature. An iLM has all mandatory features (which are provided by the framework), one of the alternative features (which is the iLM specific domain) and may have one or more optional features. A brief description of the sub-features is presented.

- *Domain Operations* – contains all domain-specific functionalities that must be defined during application engineering. It is an alternative feature; each iLM thus has a different implementation, such as geometry in *iGeom* and counting in *iComb*. This sub-feature implements domain-specific manipulate and construct types of interactivity.
- *Assignment Management* – the management of assignments is performed by this feature. It allows the user to author and edit assignments as a teacher, and to solve an assignment as a student. The modeling of assignments is domain independent, which allows this feature to be mandatory and provided by the framework.
- *Communication Management* – provides the functionality for opening and saving files and for communicating assignment data through the iLM protocol. These tasks can be entirely implemented during Domain Engineering, thus it will be independent of the specific data being transferred.

- *Configuration Management* – this feature manages the system configuration, with parameters such as language and teacher or student versions (to enable or disable assignment authoring). Configurable behaviors specified during application engineering can also use this feature.
- *Functionalities over Assignments* – this feature has operations over assignments that are being solved by the student or authored by the teacher. Examples are automatic checking of the student's solution (which is provided by the framework, even though considered as optional) and exporting to images. Other functionalities may be included, such as exporting assignments as Learning Objects standard packages.
- *Functionalities over Domain Operations* – this feature manages the user's domain operations during an assignment. It provides domain-independent functionalities that facilitate or improve manipulate and construct types of interactivity. For instance, it has sub-features of undo and redo, domain operations history, domain object list (which are mandatory) and others, such as scripts and intelligent tutoring.

There are many feature interactions among mandatory features, which are omitted in Fig. 3 for simplicity, and all of them depend on the existence of the *Domain Operations* feature. For instance, authoring and editing assignments (from *Assignment Management* feature) depend on reading and writing files and on the configuration of the teacher or student version. Since the framework handles all interactions among mandatory features, during Application Engineering, developers of iLM do not need to consider them. In the case of optional features, they can depend on each other (as *Intelligent Tutoring* depends on *Tutor Authoring*), but no mandatory or alternative feature depends on them.

All subsequent development adopting the PLUS method is highly dependent on this feature division. Next in the paper, feature means one of the groups of sub-features listed above. Use case analysis and each step during prototype development is also divided into these sub-features.

5.3. Use Case Analysis

For designing the framework, the use case analysis was conducted for each mandatory or core feature (since it will not provide alternative or optional features). As its result, five use case diagrams were created. Each use case, depending on its sub-feature, is classified as core or optional.

Fig. 4 shows, as an example, a combined use case diagram for the *Assignment Management* and *Functionalities over Assignments* features. Core sub-features are depicted in **bold** and optional sub-features in *italic*. We briefly describe its use cases below:

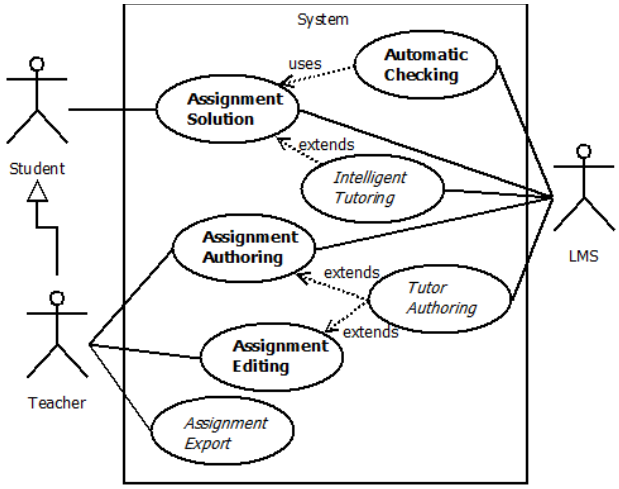

Fig. 4: Use case diagram for the *Assignment Management* and *Functionalities over Assignments* features.

- *Assignment Solution* – the user playing the role of student opens the iLM with an assignment to be solved; the assignment file is provided by the LMS server which embeds the iLM.
- *Intelligent Tutoring* – in the case of an iLM and an assignment with intelligent tutoring features, when the file is opened, the tutor is also executed. After finishing the assignment, it sends tutoring data to the LMS server.

- *Automatic Checking* – when the student wants to check whether his or her solution is correct, if available, the automatic checking is launched. It sends the assignment result to the LMS which embeds the iLM. The result can be stored for further analysis.
- *Assignment Authoring* – the user opens the iLM playing the role of teacher. All features for assignment authoring are available to this role. The assignment file can be stored in the LMS server.
- *Tutor Authoring* – if the iLM has intelligent tutoring features, while authoring an assignment, the user can set the tutor behavior.
- *Assignment Editing* – as a teacher, the user can open assignment files to edit from the LMS server page. The available features are the same as in the *Assignment Authoring* use case.
- *Assignment Export* – the teacher may also, when available, export the assignment authored to standard specifications such as IMS-LD (IMS, 2003) or SCORM (Advanced Distributed Learning, 2004).

According to the adopted development process, having finished the first iteration of software analysis for Domain Engineering, the next step is to develop its first version, as described in the next section.

5.4. Framework development

After the use case analysis, the application framework development was composed of an iterative phase of software analysis, design, implementation and testing. The Domain Engineering of a SPL intends to provide code, architecture and process for reuse. The framework provides the code and architecture, while the method for its use provides a systematic reusable process for developing iLM. This section describes the iterative development steps, and then the results from each of the development iterations, including a discussion about the influence of the used method and some lessons learned.

Software analysis takes data from requirements and creates a coarse structure of the software. In this case, each use case diagram was analyzed individually in order to generate separated UML class diagrams. We define code components that had a one-to-one relation with features. Variability management was considered by choosing which classes had variant behaviors and which were constant. All diagrams had variant classes only on user interfaces, except the one modeling the *Domain Operations* feature.

In the last iterations, variability management had little impact on implementation, as the architecture got more concise and analysis was reviewed mainly to solve problems faced during implementation or to add more flexible or simpler project choices. The PLUS method was of great importance during conception but not during refinement. This may be due to the relative small size of the project and the number of people involved.

The design phase transforms the coarse architecture from analysis into a well defined set of communicating components. This was done using design patterns (Gamma, Helm, Johnson and Vlissides, 1995 and Gomaa, 2003) in the first iterations and refactoring techniques later (Fowler, Beck, Brant, Opdyke and Roberts, 1999 and Kerievsky, 2004). The main product of this phase, obtained in the last iterations, was a component architecture that reflected feature variability in a good manner. This was allowed by the PLUS method and refined design decisions incrementally, which were used to restrain variability management only to the analysis and design phases, making it more transparent during implementation. Consequently, implementation could deal with each component individually, no matter how it would be used during application engineering. It is important to highlight that this design level was not attained until the last iterations, and was a result of several implementation efforts and iterations.

The implementation phase generates code from the designed architecture, and was a key phase to identify when to start new development iterations. Testing phase was conducted along implementation with a small time offset. Because of design decisions, considerations over variability management reduced over time, together with the difficulty in programming.

Development took four main iterations, most of them dedicated to design the framework. At the first iteration, design was under specifying, which hindered implementation. In the second iteration, design activities took longer but the result was an over specifying design, which also impaired other phases. By the

third iteration, design was mature and most of the implementation was achieved. Finally, during the fourth iteration, some design flaws were corrected and the method for using the framework was simplified. For testing the framework, an iLM was created to serve as an example. This iLM-example has no pedagogical objectives and its functionalities were defined in order to instantiate the flexible points of the framework.

The proposed method to use the developed framework also matured throughout the iterations by getting more and more simple. Here, the PLUS method aided to model variability points which were implemented as hotspots (Fayad and Schimidt, 1997), such that the instantiation of different of these flexible points were similar. The framework started with many hotspots with no relation to each other and, as design was refined, hotspots were merged and were organized into three groups: domain model, domain user interface and plug-ins of functionalities over assignments and over domain operations. To document this method, we developed three manuals, in ascending order of technical depth: (a) How to instantiate the framework; (b) How to extend the framework with plug-ins; and (c) How the framework works internally. The most used was (a), which has a step-by-step nature, providing a systematic process for developing an iLM. Moreover, maintenance and evolution issues were considered throughout the manuals and code documentation.

Based on our experience in applying our process to a set of different learning systems, we have identified the most important characteristics of the framework, namely iterative design and implementation, a powerful method, good design decisions and improved by feedbacks from developers. Iterative design and implementation made explicit errors early, reducing the amount of rework and turning the fixing task easier. A powerful modeling method, such as PLUS, guided the early design, was over-used sometimes, and then turned almost transparent in later iterations with the help of good design decisions, which made implementation simpler. Lastly, documenting and the users' opinion made visible some design that could be enhanced.

5.5. Framework current state

The application framework that implements the Domain Engineering has a component architecture, as shown in Fig. 5. This architecture can be divided into two groups: (a) the user interface, with the two components in the upper part of the diagram; and (b) the features, with six components, one for each feature in the feature diagram. Core components are shown in **bold font**, the variant components in *italic and underlined*, and the optional ones in comic sans font. The framework currently provides all core components, some abstract classes for the variant components, and some examples of components for the optional ones.

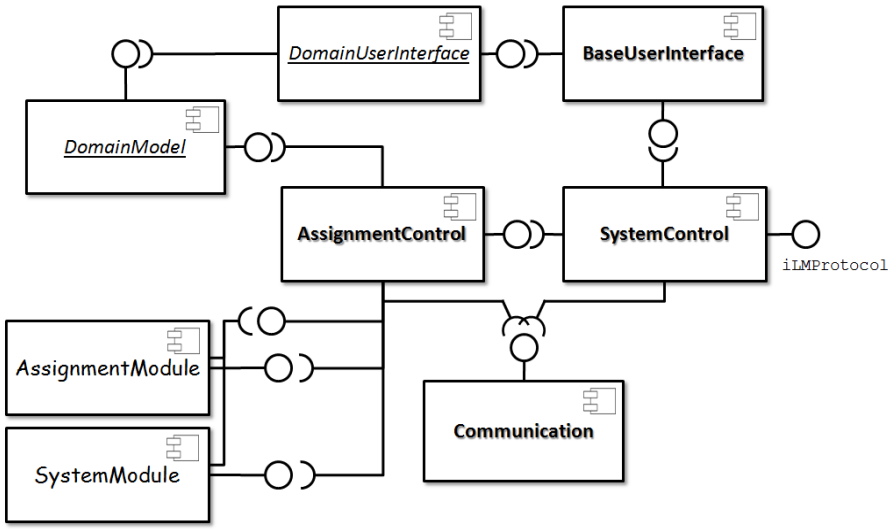

Fig. 5: Application framework's component architecture.

Each of the components is detailed as follows:

- *DomainUserInterface* – is composed of abstract classes that are Java Swing components, which must be inherited and specialized to provide domain specific behavior. The framework also provides functionalities to create buttons and to trigger domain operations.

- *BaseUserInterface* – this is an empty frame with basic and domain-independent functionalities that is responsible for initializing and for showing the *DomainUserInterface* component. Another responsibility is the management of each module user interfaces (the plug-ins).
- *DomainModel* – the component that provides the model on which the domain operations must be created. It is a group of abstract classes that model domain operations, objects and functionalities.
- *AssignmentControl* – a component for the management of assignments, which is also responsible for managing plug-ins. The main control class connects assignments with plug-ins, with the communication component and with the domain model.
- *SystemControl* – is responsible for initializing the whole system and for managing the configuration of features. It also provides the public interface that is used by LMS for external communication.
- *AssignmentModule* – is the component responsible for setting the plug-in architecture to modules that provide functionalities over domain operations. In addition to the provided architecture, the framework has three modules: undo/redo, history of domain operations and a list of domain objects. These modules can be used as examples or basis for the development of new plug-ins.
- *SystemModule* – is the component responsible for setting the plug-in architecture to modules that provide functionalities over assignments. The framework also provides an example of system module: the automatic checking module.
- *Communication* – this component has the functionalities related to communicating with other systems, such as saving and reading files, opening assignments and sending evaluations to LMS.

The current state of the framework has around 5,000 lines of code within 50 classes and 12 interfaces. Each component is a Java Package. All the features were been implemented, except for some details of teacher and student versions and other configuration parameters. The code is available as open source at http://ccsl.ime.usp.br/redmine/projects/ima/files.

Regarding the framework evolution and extension, the method proposes the inclusion of plug-in based functionalities, as AssignmentModule and SystemModule components. A manual was created to document the method in detail for this development. The proposed systematic method for using the framework to develop an iLM is presented in next section.

5.6. Method for instantiating the Framework

Application Engineering is the process of using the products of Domain Engineering to develop systems of the family, in this case, iLM. This section details the method of instantiating the application framework and using the systematic process provided by Domain Engineering.

First, one must know which specific domain the iLM will provide to teachers and students. Among the iLM features, the application framework provided by Domain Engineering does not have the domain operations and can be extended by functionalities over domain operations and assignments. Hence, in order to develop an iLM, in this phase one must define, within the chosen domain, what the domain operations are and if it will use plug-in based functionalities.

Examples of domain operations, in existing iLM, are: in *iGeom*, creating points, lines, circumferences, moving points, editing objects, reflecting objects to an axe; and in *iGraf*, creating functions, curves, points, editing these objects, animating them, etc. Moreover, there are two types of plug-in based functionalities: domain-specific and domain-independent. Domain-independent plug-ins, modules such as undo/redo and automatic assessment, are considered framework extensions as they can be developed during Domain Engineering. Nevertheless, it is possible that a certain domain needs specific functionalities, such as specific intelligent tutoring and specific export engines (e. g., exporting *iVProg* programs to programming languages). For this reason, domain specific plug-ins must be developed during Application Engineering.

Different cookbooks are proposed for each alternative or optional component in Fig. 5. In the case of *DomainModel* component, each domain specific operation must be modeled as a class inheriting `DomainAction` class, which is a *Command* design pattern (Gamma et al., 1995). These operations manipulate

objects of classes inheriting `DomainObject` class. The receiver of `DomainAction` objects is always the *Singleton* concrete `DomainModel` object, the abstract class of which is also provided by the framework. `DomainModel` implements all operations, it is the core of the iLM domain.

In the case of plug-ins, the modules of AssignmentModule and SystemModule components, implementation is different for each kind. If they implement the `AssignmentModule` interface, they are *Observers* of `DomainActions` and/or `DomainObjects`, so they must do something when an operation is triggered and/or an object is created or destroyed. If they implement `SystemModule`, its functionality manipulates an Assignment object, which has data of all `DomainActions` and `DomainObjects`, but also of propositions, other modules, etc. These modules also have access to communication functionalities provided by the framework.

DomainUserInterface is the remaining component. Its design can be produced as a standalone domain-specific user interface, which must be put together using the *Strategy* pattern. In the end, the connection of this user interface with the framework is made by inheriting the `DomainGUI` class. This class has some methods that were designed to connect the interaction mechanisms of the user interface with the `DomainAction` objects. Thus, for instance, when a button is pressed in `DomainGUI`, a `DomainAction` object is triggered, which calls a `DomainModel` method to execute the functionality.

Another step in designing *DomainUserInterface* component is the user interface for authoring assignments, which is also based on the *Strategy* pattern. The `AuthoringGUI` abstract class provided by the framework must be inherited to allow the final user, usually a teacher, to author domain-specific assignments for this iLM. This step is simply allowed by inheriting `AuthoringGUI` class and by implementing its abstract methods, which are specific for each datum of the `Assignment` object.

In summary, the suggested method for Application Engineering using the framework provided by Domain Engineering consists of: defining domain operations and objects, which are implemented as `DomainAction` or as plug-ins, and designing two user interfaces, the domain and the authoring one. In the end, the complete iLM consists of the framework, the chosen plug-ins, the objects of the domain model, and the specific user interfaces. This method is described in detail in the manuals provided with the framework and is also available (in Portuguese) with its code at http://ccsl.ime.usp.br/redmine/projects/ima/files.

6. RESULTS AND DISCUSSION

As mentioned in Section 1, the evaluation of complete results concerning an SPL and its products is a long term task. Nevertheless, since an SPL can be divided into Domain Engineering and Application Engineering, some intermediate results can be analyzed during each of the aforementioned phases. In fact, Domain Engineering may be analyzed through the work products that will be generated by following its associated process, the use of its underlying architecture and the provided code. Moreover, such analysis may be provided by the description of Application Engineering for the iLM SPL, or even part of it, associated with some metrics to evaluate what is already done.

Therefore, we analyze process, architecture and code reuse, and an estimative on quality and maintainability for each system, following some pre-established criteria. Such criteria consider that some existing iLM would be adapted to the proposed Domain Engineering in order to achieve the long term goals of improving quality for maintainability and evolution, and that some iLM would be built from scratch. Thus, process reuse would be analyzed by interviewing the developers before and after the use of Domain Engineering. Architecture reuse would be analyzed by comparing the new structure of the system with the previous one. Code reuse would be analyzed by evaluating how much of the code (%) could be discarded from the old system. Quality and maintenance could be estimated by requirements and maintainability analysis.

In this section, we describe how Application Engineering for the existing iLM is being conducted. We present current products of the development process and we discuss the corresponding results by highlighting some lessons learned. Currently, each iLM is being rebuilt or adapted in order to work with the framework. Also, a new iLM to support Genetics teaching and learning is in its early stage of development, and it has used the proposed method from the very beginning. Depending on the iLM specificities and developers'

profiles, their adaptation to the proposed Domain Engineering follows a different path towards the integration to the framework. The current stage of adopting the Domain Engineering to the existing iLM is described as follows.

iGeom is being adapted through disentangling the code, mainly between the user interface and the geometry model. The separation of concerns was made in order to make the system architecture compatible with that of the framework. Since *iGeom* is the oldest and the largest (46,000 lines of code), maintenance issues were common and justified the effort of restructuring. Two programmers with professional experience in programming are working on *iGeom*. They are refactoring the older version of *iGeom* by applying our Domain Engineering approach, good practices of programming and professional tools. Thus they have enhanced the development process and its products. Currently, *iGeom* is being integrated to the framework code.

With *iGraf* and *iComb*, adaptation is similar to *iGeom*, but in smaller scales. *iGraf* has around 15,000 lines of code (one third of *iGeom*) and was being restructured even before the Domain Engineering development due to maintenance issues. As the framework reached stability, this restructuring was adapted to follow its design. This task is performed by one developer with only academic experience in programming and it is now being finished in order to be integrated to the framework. On the other hand, the adaptation of *iComb*, with half the size of *iGraf*, also started with a non-experienced programmer and was delayed due to administration issues. The current stage is still at the beginning of separation of concerns and domain model specification.

iVProg is being rebuilt from scratch rather than adapted. Reasons for this includes the fact that *iVProg* was created based on Alice (Carnegie Mellon University, 2012), a well established educational software, that was simplified and adapted to be an iLM for 2 years. The current version of *iVProg* has around 37,000 lines of code, most of them as Alice's legacy, which makes it very hard to maintain and to modify. In the last three months, one programmer with no professional experience is developing the new *iVProg* version, rebuilding it from scratch using Domain Engineering for the family. The domain model and the domain user interface of this new version are ready, now with good and clean design as guided by Domain Engineering. There was gain in productivity not only due to the use of Domain Engineering but also due to the reuse of the domain-specific feature design. The integration of the new *iVProg* to the framework code has just started.

In this context, process reuse analysis resulting from interviews with the developers shows that they came from ad-hoc development, in which requirements were defined at any time and features were designed directly on the code, to a guided development stated by the process provided by Domain Engineering. After the beginning of the restructuration process, they knew beforehand what to do, and it was possible to know where they were going in the long term. Consequently, development was more organized and satisfactory.

As far as code reuse is concerned, we can only estimate it since the refactoring of our learning systems is not yet completed. Still, architecture reuse has had the most important impact on the developers' work. The percentage of code reuse decreases with the system size; *iGeom* has around 35% of code for domain independent features and system structure, while *iComb* has over 45%. *iVProg*, due to its legacy code structure, cannot be analyzed this way. The modularity of these systems increases when their architecture are restructured in order to be compatible with the framework, and the responsibilities of each component in the overall architecture gets clearer. A key point stated by *iGeom* and *iVProg* programmers is that with the provided architecture, they do not need to spend time thinking of how the system code should be organized, and this organization does not need to change over time due to including specific features.

The system quality may be evaluated through the requirements that were fulfilled. Since requirements that are common to the iLM family are provided by Domain Engineering, system quality that relies on them can be estimated by testing the framework. These tests are currently being conducted to assure this quality. Moreover, architecture reuse can affect the quality of domain-specific features but, since they are implemented with a previously tested structure, the tasks of specification, design and debugging are benefited. Maintainability is mainly dependent on documentation, architecture and code quality. On the one hand, for common structure and features, they are provided directly by Domain Engineering, being reviewed by all programmers that use it, raising its quality iteratively. On the other hand, they are guided by Domain

Engineering, which also allows the iterative revision. These considerations, added to the fact that restructuring leads to better quality systems from the initial stage of development, also contributes to making maintenance easier.

7. CONCLUSIONS

Interactive Learning Modules (iLM) have the potential to improve teaching and learning quality by providing adequate technology that meets teachers' and students' needs in both classroom and distance education settings. However, the development of this software family is complex and time-consuming, mostly because the lack of methods and processes that help its development. To overcome this problem, we proposed a Domain Engineering within a Software Product Line (SPL) approach to define and to guide the iLM development process.

To characterize the iLM software family, we conducted an initial analysis of five iLM developed to enhance learning in different domains (Geometry – *iGeom*, Mathematical Functions – *iGraf*, Counting – *iComb* and Programming – *iCG* and *iVProg*). This initial result helped to identify common and desired features that are essential to keep the functional consistency among them and it provided the foundations to build an application framework. With this framework, Domain Engineering provides documentation to foster process, architecture and code reuse during the development of iLM.

The main contribution of this work is the centralization of knowledge about the iLM family as the result of Domain Engineering, in the proposed application framework. Before that, knowledge was spreaded among existing systems, now detailed specifications and code can be shared among all iLM with the support of our framework. This is important in order to increase standardization among common functionalities of existing and new iLM and to facilitate the understanding of the systems developed in our research group by others. The proposed Domain Engineering provides code, architecture and process reuse, and code reuse usually gets more attention. Besides the importance of code reuse, our results show that architecture and process reuse have great influence on systems development, mostly if they were ad-hoc implemented before.

Research on evaluating the use of software engineering methods and techniques for the development of educational software is at its early stages. This work intends to disseminate the idea of using and evaluating these methods and techniques by showing preliminary results of the evaluation of applying a SPL approach to develop or to refactor iLM. In the long term, we expect to provide continuously better evaluation of the software engineering techniques used, while improving the educational software development process.

Future work consists of finishing the adaptation and rebuilding of existing iLM. After that, we could better analyze code reuse provided by Domain Engineering. Also, new iLM are being created from scratch using Domain Engineering, which can generate new development results as well as new data for studying its influence. As a long term goal, maintenance and evolution of existing iLM will be monitored in order to evaluate maintenance gains.

8. ACKNOWLEDGMENTS

Danilo L. Dalmon is supported by FAPESP under grant 2010/06805-2. This work was partially supported by FAPESP (2011/10926-2) and CNPq (550449/2011-6).

9. REFERENCES

ADVANCED DISTRIBUTED LEARNING, (2004): Shareable Content Object Reference Model 2004 Specification, www.adlnet.gov/capabilities/scorm accessed in December 2011.

ALBANESE, M. A. and MITCHELL, S., (1993): Problem-based learning: A review of literature on its outcomes and implementation issues, *Academic Medicine*, vol. 68, pp.52-81.

ALEVEN, V., MCLAREN, B. M., SEWALL, J., and KOERDINGER, K. R., Example-tracing tutors: A new paradigm for intelligent tutoring systems. *International Artificial Intelligence in Education Society*, 2008, pp. 105–154.

AHMED, F., and ZUALKERNAN, I. A., (2011): A Software Product Line Methodology for Development of E-Learning System, International Journal of Computer Science and Emerging Technologies, Vol. 2, pp.285-295.

ATEYEH, K., and LOCKERMANN, P. C., (2006): Reuse- and Aspect-Oriented Courseware Development, *Educational Technology and Society.*

BOEHM, B., (1986): A Spiral Model of Software Development and Enhancement, *ACM SIGSOFT Software Engineering Notes*, vol. 11, pp.14-24.

BOTE, M. L., HERNANDEZ, D. L., DIMITRIADIS, Y.A., ASENSIO, J. I. P., GOMEZ, E. S., VEGA, G. and VAQUERO, L. M. G., (2004): Towards Reusability and Tailorability in Collaborative Learning Systems using IMS-LD and Grid Services, *International Journal on Advanced Technology for Learning*, vol. 1, pp.129-138.

BOYLE, T., (2003): Design Principles for Authoring Dynamic Reusable Learning Objects, *Australian Journal of Educational Technology*, vol. 19, pp. 46-58.

CARNEGIE MELLON UNIVERSITY, (2012): Alice – An educational software that teaches students computer programming in a 3D environment. www.alice.org. Accessed 19-June-2012.

CHOQUET, C. and CORBIÈRE, A., (2006): Reengineering Framework for Systems in Education, *Educational and Technology*, Vol. 9, No 4, pp. 228-241.

CLEMENTS, P. and NORTHROP, L., (2001): Software Product Lines: Practices and Patterns (The SEI Series in Software Engineering), *Addison-Wesley Professional*.

DALMON, D., TANBELLINI, M., EISENMANN, A., NASCIMENTO, M., RODRIGUES, P., DO PRADO, R., KAMIYA, R., ISOTANI, S., BRANDÃO, A. and BRANDÃO, L., (2011): Interactive Learning Modules in Engineering Education and as a Motivational Tool for Middle and High School Students, *Proceedings of International Symposium on Engineering Education* – IGIP.

DALMON, D. L., ISOTANI, S., BRANDÃO, A. A. F. and BRANDÃO, L. O., (2011): Work in Progress - A Framework for Building Interactive Learning Modules, *Proceedings of Frontiers in Education*.

DALMON, D. L., ISOTANI, S., BRANDÃO, A. A. F. and BRANDÃO, L. O., (2011): Work in Progress - Enhancing Interactive Geometry Systems with Intelligent Tutoring Features, *Proceedings of Frontiers in Education*, 2011.

DO PRADO, R. and BRANDÃO, L.O., (2006): iGraf: Módulo de Aprendizagem para Ensino de Função na Web, *XIV Brazilian Symposium on Informatics in Education* (in Portuguese).

DODERO, J. M. and DIEZ, D., (2006): Model-Driven Instructional Engineering to Generate Adaptable Learning Materials, *Proceedings of International Conference on Advanced Learning Technologies*, pp. 1188-1189.

DOUGLAS, I., (2001): Instructional Design Based on Reusable Learning Objects: Applying Lessons of Object-Oriented Software Engineering to Learning Systems Design, *Proceedings of Frontiers in Education*.

EISENMANN, A. L. K. and BRANDÃO, L. O., (2009): iComb: um sistema para o ensino e aprendizagem de combinatória em ambiente Web, *XX Brazilian Symposium on Informatics in Education* (in Portuguese).

FAYAD, M., SCHIMIDT, D. C., (1997): Object-oriented application frameworks. Communications of the ACM, vol. 40, pp. 32-38.

FOWLER, M., BECK, K., BRANT, J., OPDYKE, W. and ROBERTS, D., (1999): Refactoring: Improving the Design of Existing Code, *Addison-Wesley Professional*.

GAMMA, E., HELM, R., JOHNSON, R., and VLISSIDES, J. (1995): Design Patterns: Elements of Reusable Object-Oriented Software. *Addison-Wesley*, Reading, MA.

GOMAA, H., (2004): Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures, *Addison-Wesley Professional*.

HARA, N., KLING, R., (1999): Students' Frustrations with a Web-Based Distance Education Course. *First Monday: Journal on the Internet*, 4(12). http://131.193.153.231/www/issues/issue4_12/hara/index.html

HENTEA, M., SHEA, M. J., and PENNINGTON, L., A (2003): A perspective on fulfilling the expectations of distance education, *Proceedings of Conference on Information technology Curriculum*, pp.160–167.

HINOSTROZA, E., REHBEIN, L. E., MELLAR, H. and PRESTON, C., (2000): Developing Educational Software: a Professional Tool Perspective, *Education and Information Technologies*, vol. 5, pp.103-117.

IMS (2003): IMS Learning Design Specification, *Boston: The IMS Global Learning Consortium, Final Specification*.

ISOTANI, S. and BRANDÃO, L. O., (2008): An algorithm for automatic checking of exercises in a dynamic geometry system: iGeom, *Computers and Education*, vol. 51, pp. 1283-1303.

KANG, K.; COHEN, S.; HESS, J.; NOWAK, W.; PETERSON, S. (1990): Feature-Oriented Domain Analysis (FODA) – Feasibility Study (Technical Report). *Software Engineering Institute, Carnegie Mellon University*. CMU/SEI-90-TR-21.

KAMIYA, R. H., BRANDÃO, L. O., (2009): iVProg - um sistema para introdução à Programação através de um modelo Visual na Internet, *XX Brazilian Symposium on Informatics in Education* (in Portuguese).

KERIEVSKY, J., (2004): Refactoring to Patterns, *Addison-Wesley Professional*.

KOERDINGER, K. R. and CORBETT, A. T., (2006): Cognitive Tutors: Technology bringing learning science to the classroom. In K. Sawyer (Ed.) *The Cambridge Handbook of the Learning Sciences. Cambridge University Press*.

KORTENKAMP, U and RICHTER-GEBERT, J., (2004): Using automatic theorem proving to improve the usability of geometry software, *Proceedings of MathUI* (Mathematical User Interfaces).

LINDEN, F.J. van der, SCHMIDT, K. and ROMMES, E., (2007): Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering, *Springer-Verlag New York, Inc. Secaucus*, USA.

MAYES, J.T. and FOWLER, C.J., (1999): Learning technology and usability: a framework for understanding courseware, *Interacting with Computers*, vol. 11, pp. 485–497.

MOR, Y. and WINTERS, N., (2007): Design Approaches in Technology-Enhanced Learning, *Interactive Learning Environments*, vol. 15, pp.61–75.

MUZIO, J. A., HEINS, T. and MUNDELL, R. (2002): Experiences with reusable E-learning objects From theory to practice, *Internet and Higher Education*, vol. 5, pp. 21–34.

NASH, S. S., (2005): Learning Objects, Learning Object Repositories, and Learning Theory: Preliminary Best Practices for Online Courses, *Interdisciplinary Journal of Knowledge and Learning Objects*, vol. 1, 2005.

NICOLSON, R.I and SCOTT, P.J., (1986): Computers and Education: the software production problem, *British Journal of Educational Technology*, 17: 26–35. DOI: 10.1111/j.1467-8535.1986.tb00494.

OBERWEIS, A., PANKRATIUS, V. and STUCKY, W., (2007): Product Lines for Digital Information Products", *Information Systems*, vol. 32, pp. 909-939.

ORACLE, (2012): Java SE Technologies. http://www.oracle.com/technetwork/java/javase/documentation/spec-136004.html. Accessed 19-June-2012.

PANKRATIUS, V., STUCKY, W. and VOSSEN, G., (2005): Aspect-oriented reengineering of e-learning courseware, *The Learning Organization*, vol. 12, pp. 457-470.

POLSANI, P. R., (2003): Use and Abuse of Reusable Learning Objects, *Journal of Digital Information*, vol. 3.

QUINTANA, C., KRAJCIK, J., SOLOWAY, E. and NORRIS, C., (2002): A framework for understanding the development of educational software. In *The human-computer interaction handbook*, Julie A. Jacko and Andrew Sears (Eds.). L. Erlbaum Associates Inc., USA, pp. 823-834.

RICHARDS, G., MCGREAL, R., HATALA, M. and FRIESEN, N, (2002): The Evolution of Learning Object Repository Technologies: Portals for On-line Objects for Learning, *Journal of Distance Education*, vol. 17, pp. 67-79.

RODRIGUES, P. A., BRANDÃO, L. O. and BRANDÃO, A. A. F., (2010): Interactive Assignment: a Moodle component to enrich the learning process, *Proceedings of Frontiers in Education*, pp. T4F1-T4F6.

ROSCHELLE, J., KAPUT, J., STROUP, W., and KAHN, T. M., (1998): Scaleable Integration of Educational Software: Exploring the Promise of Component Architectures, *Journal of Interactive Media in Education*.

ROSCHELLE, J., DIGIANO, C., CHUNG, M., REPENNING, A., TAGER, S., and TREINEN, M., (2000): Reusability and Interoperability of Tools for Mathematics Learning: Lessons from the ESCOT Project, *Proceedings of Intelligent Systems and Applications*, pp. 664-669.

RUNESON, P., HÖST, M., (2009): Guidelines for conducting and reporting case study research in software engineering, *Empirical Software Engineering*, DOI 10.1007/s10664-008-9102-8, pp. 131-164.

RUTHVEN, K., HENNESSY, S., and DEANEY, R., (2007): Constructions of dynamic geometry: A study of the interpretative flexibility of educational system in classroom practice, *Computers and Education*, vol. 51, pp. 297-317.

SAMPSON, D. G. and KARAMPIPERIS, P., (2006): Towards Next Generation Activity-based Learning Systems, *International Journal on e-Learning*, vol. 5, pp. 129-149.

SAVERY, J. R. and DUFFY, T. M., (1995): Problem Based Learning: An instructional model and its constructivist framework, In Constructivist Learning Environments: Case Studies in Instructional Design, Ed. Wilson, B.

SIMS, R., (1997): Interactivity: a forgotten art?, *Computers in Human Behavior*, vol. 13, pp. 157-180.

SPALTER, A. M. and VAN DAM, A., (2003): Problems using Components in Educational Software, *Computers and Graphics*, Vol. 27, pp. 329-337.

TANG, B. C., (2004): Interactive e-learning activities to engage learners – A simple classification. *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications*, pp. 4092-4097.

TCHOUNIKINE, P., (2011): Computer Science and Educational Software Design, DOI 10.1007/978-3-642-20003-8_1, Springer-Verlag Berlin Heidelberg.