A Software Product Line Methodology for Development of E-Learning System

Faheem Ahmed[1], Imran A. Zualkernan[2]
[1]*College of Information Technology, UAE University, Al Ain, UAE*
[2]*Computer Science and Engineering, American University of Sharjah, Sharjah, UAE*
f.ahmed@uaeu.ac.ae, izualkernan@aus.edu

Abstract: *Software product line has become one of the most promising software practices with the potential to substantially increase the development productivity in the software industry. Learning objects can be defined along the three dimensions of pedagogy, technology and the domain. We put forward a methodology to develop a software product line of E-Learning systems. This model identifies and elaborates the essential phases and activities. The model is divided into two phases, Domain Engineering and Application Engineering. Domain engineering phase consists of two views, Product Line Infrastructure View and E-Learning Analysis View. Application Engineering phase has Product Line Application View and Core Assets Development View. Various activities related to each view are identified to enhance the reuse development process for E-Learning software product line. The methodology demonstrates the use of explicit variability definition in a learning object at various levels of the model including Site, Structure, Skin, Services, Space Plan and Stuff. The methodology is validated against a commercial e-learning course in Six Sigma.*

1. Introduction

Software product line aims at curtailing the concept of "reinventing the wheel" in software development. It accentuates on consolidating the software assets in prescribed and a systematic way rather than an ad hoc and need to know basis. Software product line provides likelihood to accommodate the changing needs of the customers into new products by competently using software assets and allows capturing market segments for profitable business. Definition of software product line terminology has been widely explored by researchers [1, 2, 3, 4] to narrate an in-depth philosophy behind this approach. Synonyms of software product line terminology have also been widely used in Europe, for example "Product Families", "Product Population" and "System Families" etc. [5, 6]. The economic potentials of software product line have long been recognized in software industry [5, 7]. Software product line engineering is gaining popularity in the software industry. Some of the potential benefits of this approach include cost reduction, improvement in quality and a decrease in product development time. Software organizations are improving business operations such as technology, administration, and product development process in order to capture a major portion of the market share to be profitable. One of their major concerns is the effective utilization of software assets, thus reducing considerably the development time and cost of software products. Many organizations that deal in wide areas of operation, from consumer electronics, telecommunications, and avionics to information technology, are using software product lines practice, because it deals with effective utilization of software assets. Software product lines are promising, with the potential to substantially increase the productivity of the software development process and emerging as an attractive phenomenon within many organizations that deal with the software development. Several studies have been done, COPA [8], FAST [9], FORM [10], KobrA [11] and PuLSE [12] etc. to elaborate the software product line process. Independent work carried out in software reusability, object-oriented, and software architecture has reached a point at which many activities can be integrated to yield a new coherent approach to product-line integration. Traditional software life-cycle models do not encourage reusability within their phases. A product line can be built around *E-Learning system* by analyzing the products to determine the common and variable features. The product structure and implementation strategy around *E-Learning system* prepares a platform for several products, which aligns with the concept of software product line.

Recently, software development trends have caused single product development to evolve into "software product line architecture" (SPLA) which integrates lines of resulting products. The main objective of software product line is to reuse the architecture for successive product development. Clements [13] defines the term "software product line" as a set of software intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment and are developed from a common set

of core assets in a prescribed way." The software product line is receiving an increasing amount of attention from software development organizations because of the promising results in cost reduction, quality improvements and reduced delivery time. Clement et al. [14] report that SPL engineering is a growing software engineering sub-discipline and many organizations, including Philips®, Hewlett-Packard®, Nokia®, Raytheon®, and Cummins®, are using it to achieve extraordinary gains in productivity, development time, and product quality.

Since the acceptance of object-oriented paradigm in the early 1980's, concepts of software architectures have evolved significantly. As Garlan and Perry [15] point out, traditionally software architecture includes the structure of the components of a program or system, their interrelationships, and the principles and guidelines governing their design and evolution. However, more recently, software architecture is being restructured towards a SPLA, where the focus is not on single product development but rather on multiple product development. In a SPLA, all of the products share the same architecture. Pronk [16] defines SPLA as a system of reuse in which the same software is recycled for an entire class of products, with only minimal variations to support the diversity of individual product family members. According to Jazayeri et al. [17], SPLA defines the concepts, structures and textures necessary to achieve variation in the features of diverse products while ensuring that the products share the maximum amount of parts in the implementation. Mika and Tommi [18] point out that SPLA can be produced in three different ways: from scratch, from an existing product group or from a single product. Hence, software product line architecture is an effective way to minimize risks and to take advantage of opportunities such as complex customer requirements, business constraints and technology.

Digital learning objects are a complex amalgamation of learning content, pedagogy and technology and represent the building blocks of any online course. While there are many views on what a learning object is [19] [20], in practice, a learning object typically contains learning objectives, reusable information objects and formative and/or summative assessments [21]. The reusable information objects can range from images, text or videos to games or simulations. Assessment objects, on the other hand, range from simple multiple-choice questions to adaptive testing techniques. Much like earlier days of object-oriented design, most discussions on learning objects have revolved around micro-level reuse; how to re-use a learning object in a different learning context. In the mean time, significant success has been achieved in macro-level reuse in the context of traditional object-oriented design. In specific, one promising area for macro-level re-use has been that of product-line engineering [13]. The commonality and variability characteristics of digital learning objects makes a clear case of software product line architecture development, which can be used to come up with multiple product development based on business case engineering.

1.1 Research Motivations

Digital learning objects are composite structure of learning content, pedagogy and technology. The use of the Internet further accelerates the popularity and significance of learning objects design and implementation at an unprecedented rate of growth. Conceptually different products using digital learning objects share commonality and variability up to certain extend. Software product line provides an opportunity to explicitly identify commonality and variability first at the architecture level and later at the implementation. Figure 1 shows the systematic view of generic product line architecture.

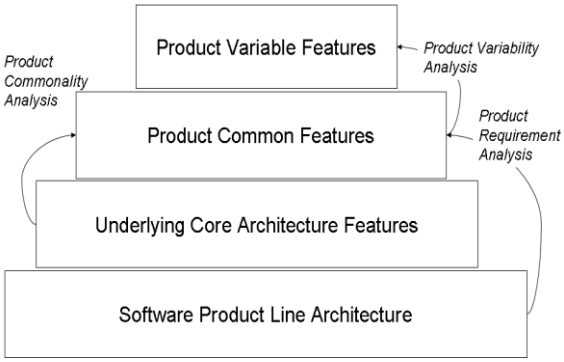

Figure 1: Software Product Line Architecture: A Structural View

A well-established requirements management activity for the software product line assists in understanding the scope and boundaries of the products to be developed and it helps in establishing the underlying core architecture features in terms of functionalities and their structure. Product line requirements deal with features or functionalities common to all the products belonging to that family. Product line requirements are composed of a constant

and a variable part. The constant part comes from product line requirements and deals with features common to all the products belonging to the family. The variable part represents those functionalities that can be changed to differentiate one product from another. This activity defines the variable part of the product requirement. The product commonality analysis provides a set of features that are common to all products, whereas product variability analysis identify explicit variation points where we can introduce changes to develop new set of products. Since the popularity of both the use of digital learning objects and software product line, there is a need to establish a process methodology for the learning objects to make use of software product line in order to get benefits of the product line approach in terms of cost, quality and reduction of development time. The objective of this study is to present a methodology to establish a software product line for E-learning system. The methodology will concentrates in identifying core architecture features of digital learning objects along with explicit definition of commonality and variability.

Given the phenomenal success and popularity of both software product line and E-learning system's development paradigms, we assert that a process guidance model can help to identify and understand the activities and tasks that need to be undertaken in order to successfully develop a software product line for E-learning system. In order to address this gap, we propose a model of developing software product line based on E-learning by incorporating several concepts that characterizing various aspects of software product line and learning objects. The proposed model identifies the interdependency of various activities of software product line and learning objects and describes different ways of exploiting the relationships between those activities in order to guide the process of developing learning objects based software product line. It should be clarified that such a process guidance model will not aim to replace existing software product line development and maintenance models and frameworks such as reported in [1] [22]. Rather, this model complements those frameworks for establishing and maintaining software product line in E-Learning context. Since Software architecture and its related

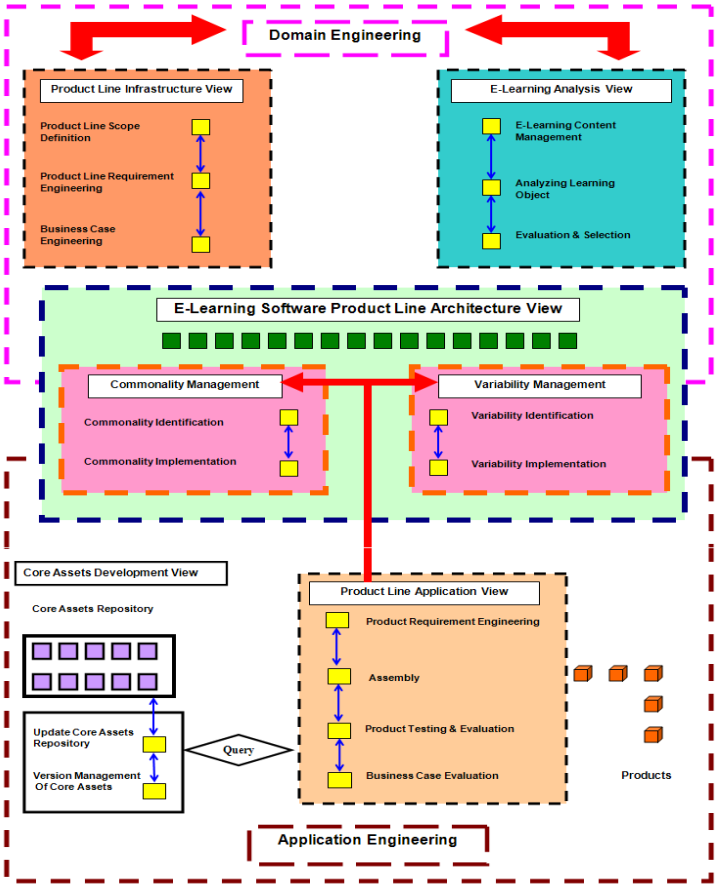

Figure 2: The Software Product Line Development Methodology for E-Learning System

issues are considered of paramount importance in the successful development and maintenance of a software product line [1] [23], this model emphasises the vital role of software architecture in developing E-learning based software product line.

2. Software Product Line Methodology for E-Learning System

The methodology shown in Figure 2 describes a complete life cycle for a software product line development for E-Learning system starting from conceptualization, initiation and development. This process methodology focuses on and encourages software reuse from repository within an application domain. The process methodology has two phases, *Domain Engineering* and *Application Engineering.* The *Domain Engineering* consists of two views, *Product Line Infrastructure View* and *E-Learning Analysis View*. Similarly the *Application Engineering* has two views, *Product Line Application View* and *Core Assets Development View*. Each view describes the development process with respect to its perspective and identifies the core activities to be performed in that view.

2.1 Domain Engineering

The *Domain Engineering* phase of the proposed methodology helps in establishing an infrastructure for software product line and constructs a core assets repository for product development. During the *Domain Engineering* phase, we initiate *Product Line Infrastructure View* and *E-Learning Analysis View.* The iterations in the activities of both views provide feedback to one another. The aim is to generate a core assets repository and a base line software product line architecture, which fulfills the product line requirements and meets the production constraints. In *Domain Engineering* phase we concentrate on the "E-Learning" and carry out activities in both the views.

2.1.1 Product Line Infrastructure View

Product Line Infrastructure View involves the activities related to conceptualization and initiation of software product line within an organization. This view performs activities that establish an infrastructure for a software product line. The *Product Line Infrastructure View* constantly provides feedback to *E-Learning Analysis View* for effective definition, identification, evaluation, selection and catalog/storage of E-learning content management. Software product line scope definition activity iteratively provides feedback to E-learning contents' definition and identification activity in *E-Learning Analysis View.* This way it ensures that all the material for content management are consistent with the scope of product line. Product line requirements deal with features or functionalities common to all the products belonging to that family. The requirement engineering for product line gives feedback to analyzing learning object activity in the *E-Learning Analysis View* to generate a candidate list of learning object's presentation that meets the product line requirements. The identification of business cases helps in evaluating identified learning objects in *E-Learning Analysis View* in order to meet the production criteria and the product requirements.

2.1.2 E-Learning Analysis View

E-Learning Analysis View is responsible for building up a core assets repository for the e-learning based software product line and provides base line information for the software product line architecture by specifically dealing with commonality and variability management. It communicates with *Product Line Infrastructure View* to generate content management. Initially the *E-Learning Analysis Engineer* identifies potential contents from the software product line requirements and scope. The definition and identification process yields a number of potential contents that can be used in the development of various products in a software product line. Those contents need to be evaluated at the individual level as well as at the product line level before they are selected for use in a software product line development. The selected contents are cataloged and stored in the repository with enough information so that they can be easily traced and retrieved as and when required for assembly. The *E-Learning Analysis View* uses support methodology to analyze, evaluate and select the contents. In this paper we are using the ED^2 Model [24] [25] for analysis, evaluation and selection of the contents discussed in detail in later part of this paper. The support methodology to analyze, evaluate and selection provides foundations for the commonality and variability selections in the software product line architecture.

2.2 Application Engineering

In the *Application Engineering* phase of the proposed methodology shown in Figure 2, actual

products are developed using software product line architecture and core assets repository. In this phase, activities of the *Product Line Application View* interact with the activities of the *Core Assets Development View* to produce required products. The application engineering phase provides feedback to domain engineering phase through business case evaluation to accommodate changing needs of the product line.

2.2.1 Product Line Application View

Product Line Application View interacts with *Product Line Infrastructure View* to identify potential business cases to capture market segment. The *Product Line Application View* generates the product requirements of the potential business case and provides feedback to *E-Learning Analysis View* to find out contents to be used in the product development. Product requirements are composed of a constant and a variable part. The constant part comes from product line requirements in the *Product Line Infrastructure View* and deals with features common to all the products belonging to the family. The variable part represents those functionalities that can be changed to differentiate one product from another. This activity defines the variable part of the product requirement. The assembly activity involves the development of product. The product requirements guide the assembly process to get feedback from the query activity of *Core Assets Development View* to find out those potential contents suitable to be assembled in order to produce the product. In product testing and evaluation, products developed from software product line are tested to analyze whether they meet the product line testing and evaluation criteria or not. Business case evaluation compares the proposed business case strategy with the outcome of the development and deployment process of products.

2.2.2 Core Assets Development View

Core Assets Development View is responsible for providing required components from core assets repository for developing products. *Core Assets Development View* interacts with *Product Line Application View* to receive product. In the query activity of the *Core Assets Development View*, components are searched from the core assets repository in order to develop the product. A well-catalogue core assets repository reduces the efforts to trace the suitable components for assembly. The product requirements serve as an input to the query activity, and continuously traversing core assets repository yields the required components, exactly matched, partially matched or not matched. The components, after adaptation, generate versions, which are documented in this activity. A comprehensive version management and dependency link strategy for components and products in the software product line engineering provides us with vital information about components and products having a relationship of composition and utilization. A software product line develops an initial core assets repository in the *Domain Engineering* phase. As a product line gets matured in its lifecycle, new core assets or even new versions of existing core assets are produced, which must be added to the core assets repository so that they can be reused in later products. The core assets repository is dynamic and continues increasing its size with the addition of new core assets.

2.3 Software Product Line Architecture

The proposed model emphasizes the importance of developing a software product line architecture based on e-learning product. The junction of *Domain Engineering phase* and *Application Engineering phase* produces a suitable product line architecture based on existing e-learning components. Overall *Software Product Line Architecture* can be produced in three different ways; it can be developed from scratch, it can be based on the existing product group, or it can be built based on a single existing product [7]. The proposed methodology emphasizes the approach of developing *Software Product Line Architecture* based on a single existing product. The junction of *Domain Engineering* phase and *Application Engineering* phase produces the *Software Product Line Architecture* based on the first product developed. The *Domain Engineering* phase provides product line requirements. The *Application Engineering* phase accommodates those requirements along with product specific requirements to establish the *Software Product Line Architecture*. The *Software Product Line Architecture* reflects the commonalities among the products and variation points where products differ from each other. All the resulting products from the product line share this common architecture. The iterative approach of methodology refines the *Software Product Line Architecture* after successive development of products.

3. The ED^2 Model: E-Learning Analysis View Support Activity

The ED^2 model for analyzing learning objects is presented in [24] [25]. The ED^2 model presents a comprehensive framework for thinking about

variability and change in any learning object. The model has two dimensions. The first dimension views variability in terms of pedagogy, technology and domain of learning. The second dimension looks at variability from the perspective of architectural layers of a learning object. The first dimension of the ED^2 model deals with the three dimensions of pedagogy, technology and the domain. The first source of variability is the domain of learning where different types and levels of knowledge may be included in a learning object. Similarly, the second source of variability is the technological basis of a learning object; one may decide to use an HTML-based learning object as opposed to a Flash-based one, for example. Finally, the pedagogy underlying the learning object can vary from tell-and-test to a socio-constructivist framework emphasizing the social nature of learning. In many cases, product families are established on top of a successful pilot product [26].

The second dimension of the ED^2 model deals with layers of change within a learning object. This dimension was originally conceived to think about layers of change in the field of architecture and later applied to learning objects [27] [28]. Brand [27] contends that a building can be viewed to consist of multiple layers that slide along each other. Each layer is designed to vary on a different time-scale. The layer that varies at the slowest speed is called the *Site*. This layer represents the physical location of a building and is consequently the most stable. The next level of variability is represented by the *Structure* of a building. The Structure may consist of the walls and roof of the building. In buildings, structures can last from 30 to 300 years and are hence less stable. *Skin* layer represents the exterior of a building and typically changes over a period of 20 years or so. The infrastructure inside a building represents the *Services* layers. This may include lighting, air-conditioning and plumbing etc. Services typically change over a life-span of decades. The *Space Plan* layer consists of the internal walls and the layout of the building. This layer can be changed every few years or so. Finally, the *Stuff* layer represents what is inside a building. For example, furniture represents one type of stuff. Stuff can be changed on a weekly or monthly basis. For a learning object, ED^2 combines the two dimensions as described below.

3.1 Site

Technologically, *Site* represents a choice of the lowest virtual machine being used to deploy a learning object. For example, one could choose Windows operating system, Java virtual machine or Adobe Flash virtual machine as the *Site*. Pedagogically, *Site* represents choosing an epistemological orientation towards learning. For example, one could choose an instructivist or constructivist pedagogy. From the domain of learning, *Site* represents choosing what constitute fundamental and immutable principles of a domain; homeopathic verses allopathic, for example.

3.2 Structure

Structure from a technology perspective, involves choosing learning design architecture like SCORM [29] or IMS-LD [30]. Pedagogically, the choice is about the nature of learning design; problem-based learning verses informal learning, for example. From a domain perspective, the choice is about embedding a particular domain ontology that emphasizes only particular views of a domain [31].

3.3 Services

Choices in technology-oriented services in a learning object may include authentication, login, tracking, archiving, and book-marking. Choice in pedagogical services, on the other hand, is represented by a level of understanding being delivered by a learning object. For example, one may use Bloom's taxonomy [32] to specify that a learning object delivers a "recall" level of understanding as opposed to "analysis." Domain services provide choices in learning objectives; what is to be learned in a learning object, for example.

3.4 Space Plan

Technologically, *Space Plan* of a learning object represents choices of personalization and customization. For example, the linear sequencing in SCORM [29] supports a flexible *Space Plan*. Pedagogically, the *Space Plan* represents a choice on the degree of adaptive-ness of the learning design; does the learning object use learning styles and preferences to determine learning paths for individual learners? Domain-wise, a *Space Plan* represents choices regarding multiple types (or roles) of learners; is the learning object designed for a technician, an engineer or both?

3.5 Skin

Technologically, *Skin* of a learning object represents choices in user-interface technologies; HTML, DHTML, XUP or SVG, for example. Pedagogically, *Skin* represents the choice of a learning template. For example, Cisco's methodology [21] specifies that teaching a concept requires a definition, an example and a non-example. Domain-wise, *Skin* represents choices in presentation styles and colors; each domain has its own presentation norms.

3.6 Stuff

Technologically, *Stuff* represents choice or selection of specific *assets* [29]; text, PDF files, Flash movies and Java scripts, for example. Pedagogically, *Stuff* represents choices about including specific instances of pedagogical primitives like objectives, topics, lessons, assessments, games, simulations, and activities. Domain-wise, *Stuff* represents choices in which of the specific definitions, concepts, processes, and principles to include in a specific learning object.

4. Application of the Methodology: A Case Study

We applied the proposed methodology software product line development to creating learning objects for the Six Sigma quality methodology [33]. As a part of implementing Six Sigma within an organization, the Six Sigma training needs to be delivered at various levels for various stakeholders. The level of training can vary from yellow-belt training to black-belt or master black-belt training. Intermediate levels of green belt and orange-belt have also been introduced. In addition to various levels of training, the Six sigma methodology has various variants like DMAIC (Define, Measure, Analyze, Implement and Control) or DFSS (Design for Six Sigma), for example. These variants can be further customized; some companies may choose to combine the Define step with Measure to just include MAIC. Another source of variability is the diverse set of conceptual and statistical tools that can be optionally used to implement this methodology; for example [34] lists about one hundred tools. The high variability in various aspects of a Six Sigma course makes it a good candidate for a SPL. A business cases for this product line is clearly justified by the need for various specialized versions of yellow belt, green-belt and black-belt courses in Six Sigma.

In the Domain Engineering we perform the activities of the *Product Line Infrastructure View* and *E-Learning Analysis View* to establish product line architecture. This can be achieved by highlighting the commonality and variability features in the products. The scope definition of product line yields that a six sigma course learning system that supports various courses and can be accessible on the WWW for delivery, simulation, discussion, and examination purposes. In order to carry out the task of product line requirements engineering based on the business case of "six sigma course product line" the activities in the *E-Learning Analysis View* starts analyzing the contents and the way the contents are presentable. The Analyzing Learning Object activities in the *E-Learning Analysis View* help in establishing the Figure 3. Figure 3 shows a simplified feature model the six sigma course learning object in terms of ED^2 model. The first variability is at the top level which is the training levels of six sigma courses of yellow, black, orange and green levels. The commonality exists at the six level of ED^2 model which are Site, Services, Structure, Space plan, Skin and Stuff. In other words, any Six Sigma course must include (and choose) aspects from each of these six. Commonality management in software product line architecture deals with all product aspects that are common across all the various ED^2's categories. The structure of ED^2 dictates that commonality should be high (less variability) at the lower layers (like the Site) and low (high variability) in the higher layers (like the Stuff). Commonality analysis ensures that the selections made for each layer of commonality according to the ED^2 model are mutually consistent. For example, since a pure Flash framework was fixed for the Site, it is consistent with the Stuff being constrained to use common Flash buttons; a use of Windows buttons for the Stuff, would obviously be inconsistent. Detailed example of commonality among successive products of six sigma learning objects are shown in Table 1 which is a result of the product line requirements engineering in the *Product Line Infrastructure View*.

The product requirement engineering activity in *Product Line Application View* provides information about as opposed to commonality; variability explicitly models what can be changed. It highlights in the core architecture where we can introduce changes to come up with new products based on the business case. Again, ED^2 provides a structured approach to identifying variability. The Figure 2 clearly highlights variations in features in terms of ED^2 model. For example, in case of Structure some products may use SCORM and others may use LD. Whereas in case of Site, some product may use HTML, DHTML, or XUP.

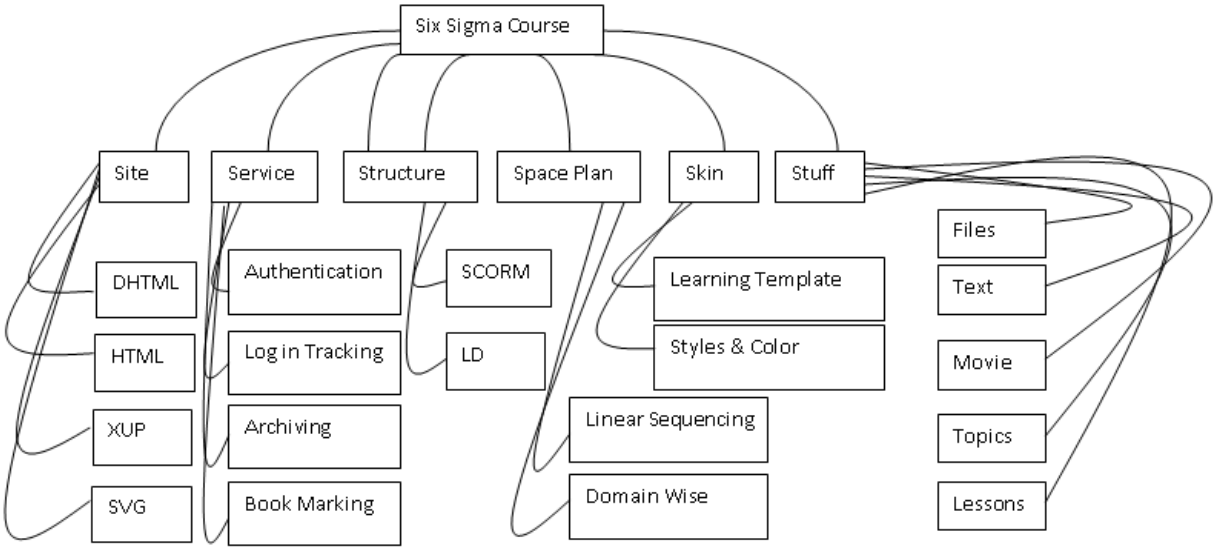

Figure 3: Feature Model of Six Sigma Course Product

Like commonality analysis, variability analysis also needs to ensure that there are no contradictions. For example, the variability in the *Services* layer (no audio) is consistent with the Flash engine's capabilities (to skip audio for a Tab) at the *Site* layer. Some more examples of variability analysis are shown in Table 2.

During *Application Engineering* phase product s are developed using the product line architecture. The Figure 4 and 5 show screen shots for a yellow-belt and orange-belt modules as a part of two successive products that are developed from the Six Sigma courses software product line. These two products consist of sets of Adobe Flash modules that constitute each course. The commonality is present in all the six factors of ED^2 model including Site, Structure, Services, Space plan, Skin and Stuff. Both courses use the same Adobe Flash Engine that interacts with a SCORM-based Learning Management System using and AICC interface to support authentication, tracking and book-marking services. Both products have the same look and feel in term of the various buttons and menus on the screen. Both products support pages and sub-pages and an audio interface. The variability arises in the depth of the presentation content. As Figure 3 shows, the yellow-belt course is mostly descriptive and introduced Six Sigma at a basic level. The orange-belt course, on the other hand, goes into the details of technical analyses like gage analysis. There is variability from a presentation perspective as well. For example, in Figures 4 and 5, the yellow belt example shows a simple table while, in the orange-belt example, the three graphs can be selected one at a time by the user by clicking on it. There is also wide variability in the two courses with respect to the pedagogy. While the yellow-belt course relies primarily on tell and test (knowledge level, in terms of Bloom's taxonomy), the orange-belt course uses scenario-based learning as well. For example, a culturally relevant example of how to select a mobile phone is used where the user can choose a particular mobile phone and try to understand the reasons behind their selection by using a factorial design approach.

Table 1: Commonality Identification Examples

Dimension	Commonality Identification
Site	Use the Adobe Flash-based engine for all products. This engine defines a common virtual layer for all the products. The engine needs to support interface to a Learning Management System. In addition, the product will use a traditional teacher-centric pedagogy.
Structure	All products will contain modules where each module will present objectives, followed by a mix of information and assessment items, followed by a conclusion. Each product will support multiple learning styles by including images, text and audio. Each product will be structured to include sub-pages in the form of Tabs. Traditional Six Sigma methodology will be used (excluding Lean concepts).

Services	All products will have login, book-marking and audio services. In addition, all products will have a help facility.
Space Plan	All modules will have pages and each page will have tabs for sub-items within a page.
Skin	All products will carry the same broiler plate with the company's logo. All products will have a forward-backwards button, an audio panel including on/off, pause and repeat buttons, a drop-down menu to access various pages of the product and a panel showing where the learner is (sub-page number) with respect to the complete module.
Stuff	All products will use common Flash buttons for forward and backwards, drop-down menus and Tab objects.

Table 2: Variability Identification Examples

Dimension	Variability Identification
Site	Arbitrary Flash movies can be embedded within the generalized Adobe Flash engine. The pedagogy can be changed to emphasize problem-based learning in a social context.
Structure	The content inside each Tab in a product can be an arbitrary image, text or another Flash movie. The depth of content presentation can be varied for yellow to black-belt.
Services	Audio can be skipped for some tabs. Similarly, tracking is optional. The learning objectives are different for yellow or black-belt.
Space Plan	The placement of text, images, and movies within a Tab can be varied. In other words, different configuration of text, images and movies can be used. The order of modules and sub-pages to include will depend on whether yellow or black-belt training is being delivered.
Skin	Colors and look & feel can be changed.
Stuff	All the content including text, images and specific audio is variable. The specific stuff to include will depend on which of the various levels of training is being imparted.

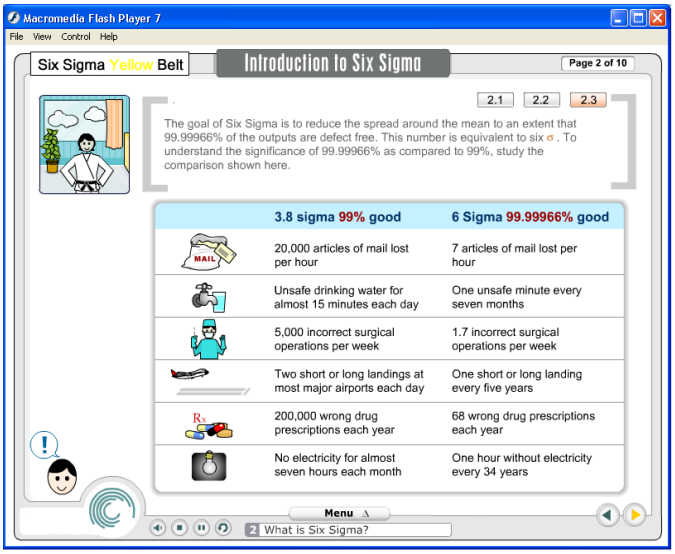

Figure 4: Screen shot of one module in a yellow-belt course

Source: w.knowledgeplatform.com

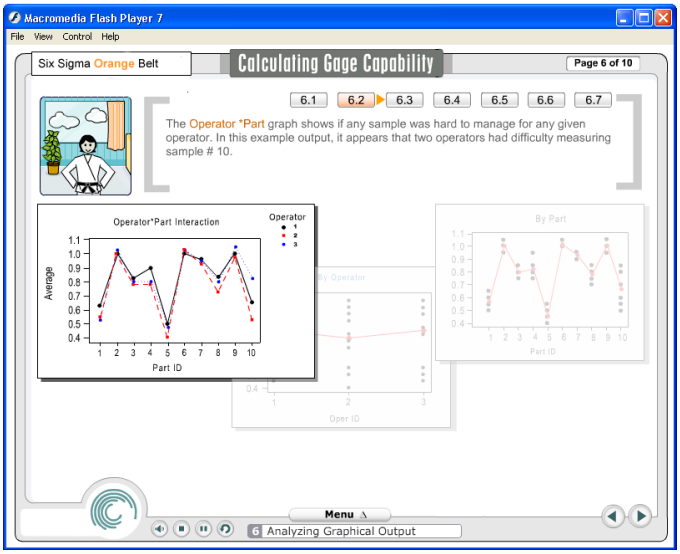

Figure 5: Screen shot of one module in a black-belt Course

Source: www.knowledgeplatform.com

5. Conclusion

Design and development of digital learning objects is gaining popularity due to significantly large increase in online-learning. Software products line engineering curtails the development time and further avoids reinventing the wheel in software development. The objective of this study was to investigate the use of product line approach in developing digital learning objects and put forward a methodology. The proposed

methodology highlights various activities of software product line and E-Learning system development. The model integrates the concept of software product line and learning object to come up with a prescribed way of establishing a software product line for E-Learning system capable of producing multiple products within an application domain. In order to validate the model, we developed a software product line for six sigma application domain, which reveals that productivity in terms of cost, time and quality would be increased if we follow the proposed methodology. Additionally, the methodology provides an efficient way of integrating the approaches of software product line and E-learning system development process.

References

[1] Clement P. and Northrop L., Software Product Lines: Practices and Pattern, Addison Wesley, 2001.

[2] M.L., Implementing Product-Line Features with Component Reuse, Proceedings of 6th International Conference on Software Reuse: Advances in Software Reusability, pp. 137-152, Lecture Notes in Computer Science, Springer Verlag, 2000.

[3] Griss D. M. Weiss and Lai C. T., Software Product-Line Engineering: A Family-Based Software Development Approach, Addison-Wesley, 1999.

[4] Griss M.L., Product-Line Architectures, G. T. Heineman and W. L Councill (Eds.) Component-Based Software Engineering, pp. 405-419, Addison-Wesley, 2001.

[5] Linden F. van der, Software Product Families in Europe: The Esaps & Café Projects,

IEEE Software, Vol. 19, No. 4, pp. 41-49, 2002.

[6] Ommering R.V. (2000) Beyond Product Families: Building a Product Population, Proceedings of the Conference on Software Architectures for Product Families, Lecture Notes in Computer Science, Springer - Verlag, pp.187-198.

[7] Buckle G.; Clements P.; McGregor J.D.; Muthig D. and Schmid K., Calculating ROI for Software Product Lines, IEEE Software, Vol. 21, No. 3, pp. 23-31, 2004.

[8] P. America, H. Obbink, J. Muller, and R. van Ommering, "COPA: A Component-Oriented Platform Architecting Method for Families of Software Intensive Electronic Products,". Denver, Colorado: The First Conference on Software Product Line Engineering, 2000.

[9] David M. Weiss and Chi Tau Robert Lai. Software Product-Line Engineering: A Family-Based Software Development Process. Addison-Wesley, 1999.

[10] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh, "FORM: A Feature-Oriented Reuse Method with Domain- Specific Reference Architectures," Annals of Software Engineering, vol. 5, 1998, pp. 143 - 168.

[11] Atkinson, C., Bayer, J., Muthig, D., Component-Based Product Line Development: The KobrA Approach, Proceedings, 1st International Software Product Line Conference,2000, pp.289-309.

[12] Bayer, J.; Flege, O.; Knauber, P.; Laqua, R.; Muthig, D.; Schmid, K.; Widen, T.; and DeBaud, M.;, (1999) PuLSE: A Methodology to Develop Software Product Lines, Proceedigns of the 5th ACM SIGSOFT Symposium on Software Reusability, pp. 122-131.

[13] Clements, P. C. (2001). “On the importance of product line scope,” in: Proceedings of the 4th International Workshop on Software Product Family Engineering, pp. 69-77.

[14] Clements, P. C., Jones, L. G. Northrop, L. M. & McGregor, J. D. (2005). “Project management in a software product line organization,” IEEE Software, vol. 22, no. 5, pp. 54-62.

[15] Garlan, D. and Perry, D. (1995). “Introduction to the special issue on software architecture,” IEEE Transactions on Software Engineering, vol. 21, no. 4, pp. 269-274.

[16] Pronk, B. J., (200) "An interface-based platform approach," in: Proceedings of the 1st Software Product Lines Conference, pp. 331-352.

[17] Jazayeri, M., Ran, A. and Van der Linden, F. (2000). Software Architecture for Product Families: Principles and Practice, Addison Wesley.

[18] Mika, K. and Tommi, M. (2004). "Assessing systems adaptability to a product family," Journal of Systems Architecture, pp. 383-392.

[19] Wiley, D., Ed. (2000). The Instructional Use of Learning Objects, (Available online at http://www.reusability.org/read/) [Accessed November 14, 2008]

[20] McGreal, R., Ed., 2004, Online Education Using Learning Objects. Open and Distance Learning Series, Routledge/Falmer, London.

[21] Cisco. (2003). Reusable Learning Objects Authoring Guidelines: How to Build Modules, Lessons and Topics, Cisco Systems, Inc., San Jose, California.

[22] F.v.d. Linden, K. Schmid, and E. Rommes, Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering. 2007: Springer.

[23] J. Bosch, Design & Use of Software Architectures: Adopting and evolving a product-line approach. 2000: Addison-Wesley.

[24] Zualkernan, I. A. (2006). "Stability Analysis of Learning Objects in Engineering Education," in Proceedings of the 2nd International Conference on Engineering Education, Kuwait.

[25] Zualkernan, I. A. (2008). Eighteen Design Decisions, Computer Science and Engineering Department, Technical Report # TR-02-08, American University of Sharjah.

[26] M. Korhonen and T. Mikkonen, Assessing Systems Adaptability to a Product Family, Journal of Systems Architecture, No. 50, pp. 383-392, 2004.

[27] Brand, S. (1994). How Buildings Learn: What Happens After They're Built. New York: Penguin Books.

[28] Gibbons, S., Nelson, J. and Richards, R., 2000, "The nature and origin of instructional objects," In D. A. Wiley (Ed.), The Instructional Use of Learning Objects, (Available online at http://www.reusability.org/read/) [Accessed November 14, 2006]

[29] IMS-SCORM, 2005, IMS Content Packaging Use Case Descriptions, Version 1.2, (Available at http://www.imsglobal.org/content/packaging/cpv1p2pd/imscp_usecv1p2pd.html) [Accessed November 14, 2008]

[30] IMS-LD, 2008, IMS Learning Design Specification, (Available at http://www.imsglobal.org/learningdesign/) [Accessed November 14, 2010].

[31] Gomez-Perez A., Fernandez-Lopez, M. and Corcho, O. (2004). Ontological Engineering, Springer, London.

[32] Bloom, B. S., Ed. (1956). Taxonomy of Educational Objectives: Book 1, Cognitive domain. New York: Longman.

[33] Pyzdek, T. (2003). The Six Sigma Handbook, McGraw-Hill, New York.

[34] George, M. L., Rowlands, D., Price, M. and Mazey, J. (2005). The Lean Six Sigma Pocket ToolBook, McGraw-Hill.