

# Change Management: Modeling Software Product Lines Evolution

Samuel A. Ajila, Ph.D. MIEEE

Department of Systems & Computer Engineering, Carleton University,  
1125 Colonel By Drive, Ottawa, Ontario, K1S 5B6, Canada, [ajila@sce.carleton.ca](mailto:ajila@sce.carleton.ca)

## Abstract

*In a traditional software engineering approach (otherwise known as single-product approach), the software architecture is evaluated with respect to the requirements of that product alone. However, a product line approach requires the software designer to consider requirements for a family of systems and the relationships between those requirements. This paper examines three kinds of evolution processes: architecture, component, and product. In modeling software product lines evolution, four distinct change management processes are identified – product change process, component-set change process, framework change process, and product line change integration. These change processes share four strategies – change identification, change impact calculation, change propagation, and change validation. This paper also present an evolution modeling approach based on the dependency relationships structure of the various artifacts involved in product line families. The paper also shows how the relationships between the different artifacts in a product line is captured and used in change management.*

## 1. Introduction

A software product is the result of a set of activities, which appeal to various competence and knowledge. Each activity manipulates different kinds of artifacts (requirements, design, coding, testing, etc.) [1, 2]. Software product may be changed in order to introduce new user requirements, to correct errors in the requirements specification or design, and to introduce new machine or new environment. Evolution comes into play because of these changes. In general, change management is necessary because of software evolution. The main defy in change management activity is to ensure the consistency of all the assets within a product line. The result of software evolution is to satisfy the need of users as well as the product development team [1, 3].

A software product line [4, 7, 10] is a group of software systems sharing a common (not identical), managed set of features that satisfy a well defined set of needs (market, special mission or otherwise) and that are developed from a set of common core assets for a specific application domain. It is an “intermediate” level of reuse where components are used for subsequent product versions and for a family of software products. Modeling product lines provides substantial reuse advantages. The benefits of reuse includes [7, 10] – *reduce time to market, reduce product development costs, improved process predictability, increased product quality, achieve large-scale productivity gains, and increase customer satisfaction*. The individual evolution of a product in a software product line is affected by the evolution of the reuse components as well as the evolution of the rest of the assets involved in the product line architecture [4]. In traditional software engineering

(single product), a sizable amount of money and time is spent doing software product evolution. This is so because of the complicity involved in the maintenance of software. For software product lines, the evolution of assets in the product line is even more complex compared to the traditional approach. The reason is that most assets depend on multiple software systems [10].

This paper examines the problems associated with product line evolution, identifies four distinct change management processes in relation to product lines, and proposes a model for product line evolution.

The rest of this paper is organized as follows. Section 2 presents software product lines evolution. Section 3 discusses the change management processes. Section 4 presents a modeling approach to software product evolution and a conclusion is given in section 5.

## 2. The problem - Software Product Lines Evolution

In general, software evolution is a change process that covers all the life cycle of a software product. This may be caused by changes to requirements definition, functional specification and design, performance specification, system’s environment, and product implementation. Software product lines evolution is more complex than the traditional software product evolution, because most assets are relied upon by the multiple products [4, 9]. The dependency between the various assets in a product line is very complex due to the multiple artifacts involved and it may be difficult to maintain the status of the assets. Products in software product line are normally built for use by the customers and the use, in general, will lead to new requirements. New requirements may also emerge from the introduction of new technology or new environment. These new requirements change can affect both system functions as well as business goals on which the product is developed. There are two things to be done when a new set of requirements emerge:

- the first one is a business decision – is whether the requirements should be incorporated into the existing product or whether a new product should be developed, and
- the second is technical – if the decision is to incorporate the new requirements then what are the effects (consequences) of these new requirements change on the product, the product line architecture, the product line requirements, and the assets set. If the effects are limited to the product itself, the evolution is handled in the usual way as in traditional software development. If the effects span beyond the product to other artifacts then the evolution is more complex because of the inter- and intra-relationships between the various artifacts. The discussion in this paper is centered on the technical aspect.

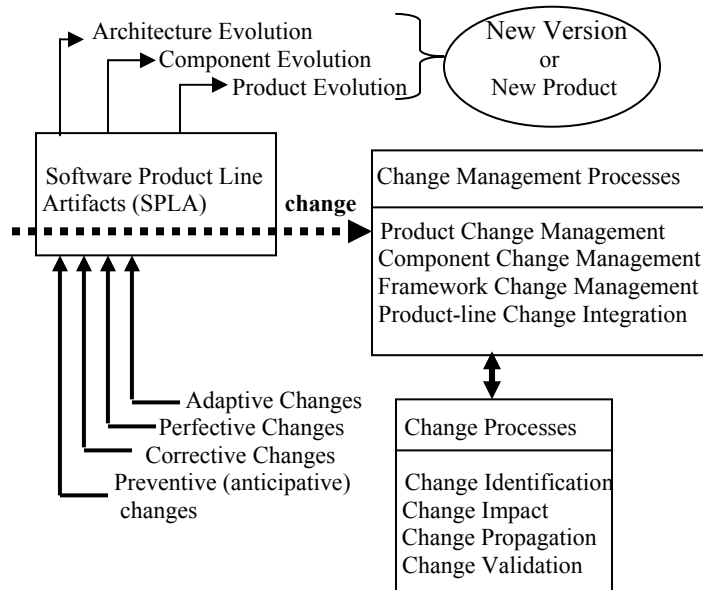


Figure 1.0 – Software Product Line Evolution

When the effects of change requests in the context of a product line go beyond the product itself to the other assets then we need to look at three different kinds of evolution – architecture, component, and product (cf. figure 1.0). The first major asset of the product line is the architecture. The main activity is to design an architecture that covers all the products including features that shear the products. The second set of artifacts is the components that have been clearly identified in the product line architecture. The third set comprises the products developed based on the architecture and the components. Therefore, any change to any part of these assets must be well managed in order to preserve the consistency of the assets base.

### 3. Change management

Change management is necessary because of software evolution. In this work, we identify four different types of changes that may affect software product line artifacts [1, 2] (cf. figure 1.0):

- *adaptive changes* – these kinds of changes deal with new environmental or technological requirements such as new support system or new machine.
- *perfective changes* – these deal with new performance requirements in order to enhance the quality of the products; improve functionality and changes to the business model. Perfective changes may be employed in order to introduce a new product (or a new version) into the market.
- *corrective changes* – diagnose and correct errors in specification, design, and development of the product or reaction to the apparition of errors in a particular product due to customer complaints.
- *preventive (anticipative) changes* – anticipate future changes and try to correct them in order to avoid major modification of the system.

In addition, four distinct change management processes are identified (cf. figure 1.0):

**Product change management** – a product is changed by adding new features or by extending the functionality of a product in order to support fully a standard. In some cases when a product is first released, it

may be decided that a subset of a standard will be supported. The standard is upgraded in subsequent releases of the product. When a change occurs within a product, the change is propagated to other assets within the product in order to ensure consistency.

- **Component change Management** – a new product may be introduced in order to support new functionality or to support an old functionality in a different way. This may be because of new hardware platform or new quality requirements. This can be achieved by adding a new domain specific component or by replacing an old component with a new one that will implement the additional functionality in a better way. When this occurs, the change is propagated within the component set of the product in order to assure consistency and maintain high quality.
- **Framework change management** - A company may decide to release a new set of products into the market based on an existing product line but this set is different in some aspects (product behavior, quality, etc.). In this case, the differences (in terms of behavior or quality) will affect many assets within the framework. Therefore, the changes generated are propagated to other assets within the framework to maintain consistency.
- **Product line change integration** – When a new product line is introduced into the software product line, changes within the software product line are propagated to all the product lines and integrated into the existing products generated from the product line architecture.  
The change management processes above share a set of sub-activities based on the change process strategies below (cf. figure 1.0):
  - **Change identification** – Identifies the changes to be made, determines the change type, and updates the history of changes. To do this, we need answers to the following questions:
    - What is the definition of the change?
    - What is the history of the change? (That is, what are the decisions that led to the change?)
  - **Change impact** – analyze the change type, calculate the impact of change on the affected parts using dependency relationships, and determine further changes that may be generated due to the impact. In effect, we need to answer the following questions:
    - What is the impact of the change on the product line artifacts?

- What is the execution pattern of the change?
- **Change Propagation** – propagate the effects of the change in the product line assets in order to ensure consistency within the assets base. That is,
  - In what way is the change going to be carried out?
  - Who is going to carry out the change?
- **Change validation** – to ensure consistency, the change must be validated.
  - It must be assured that all the needs for the change are satisfied.
  - The change must be carried out in accordance with the specification.
  - Determine the parts of the framework that may need to be regression tested after the change.

A valid product line change is *complete* if [3]:

- We can identify all other assets (i.e. *victims* of change) affected by the change;
- We can measure the consequences (i.e. *impacts*) of the change on these “victims”;
- We can trace the relationships between the changed asset and the rest of the product line families; and
- We can acquire knowledge on how to perform and manage the change.

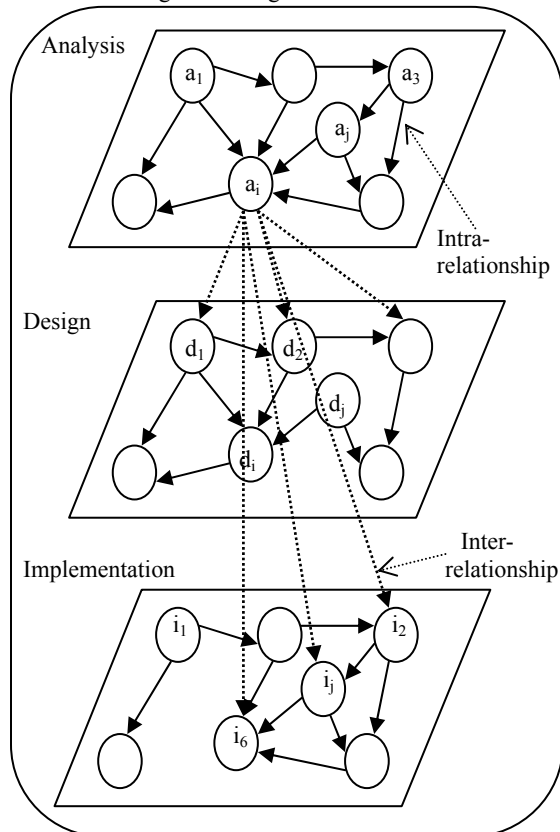


Figure 4.0 – Product life cycle phases

#### 4. Modeling Software Product Line Evolution

In this section, we present a modeling technique for software product line evolution. We start by defining a product life cycle and using dependency graph to show the relationships between the different assets within a product (figure 4.0).

The relationships between the artifacts within a life cycle phase are referred to as *intra-relationships* and those between the phases are called *inter-relationships*. The inter-relationships allow vertical traceability while the intra-relationships allow horizontal traceability. A product life cycle consists of *analysis*, *design*, and *implementation*.

Analysis represents specifications based on requirements. Using objects orientation and UML notations [5, 6], analysis model of a product consists of use *case diagram*, *sequence diagram*, *class diagram*, and *state chart* based on analysis object (figure 4.1a). The design model consists of sequence diagram, design class diagram, design objects, subsystem decomposition, and deployment diagram (figure 4.1b). The implementation implements an executable product based on the design specification.

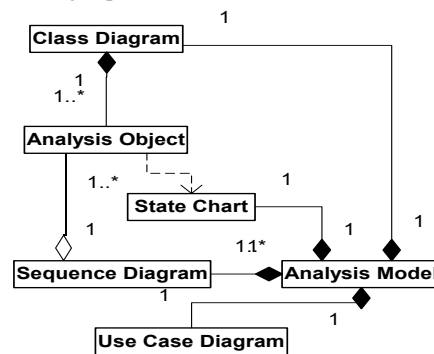


Figure 4.1(a) – Analysis Artifacts

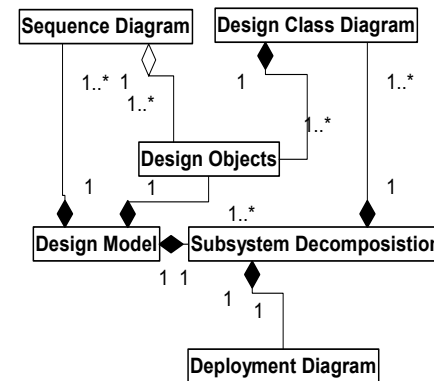


Figure 4.1(b) – Design Artifacts

We view each product in the product line as a two-dimensional plane consisting of a set of horizontal traceability and a set of vertical traceability (figure 4.0). The products are related to one another because of the shared common assets (figure 4.2).

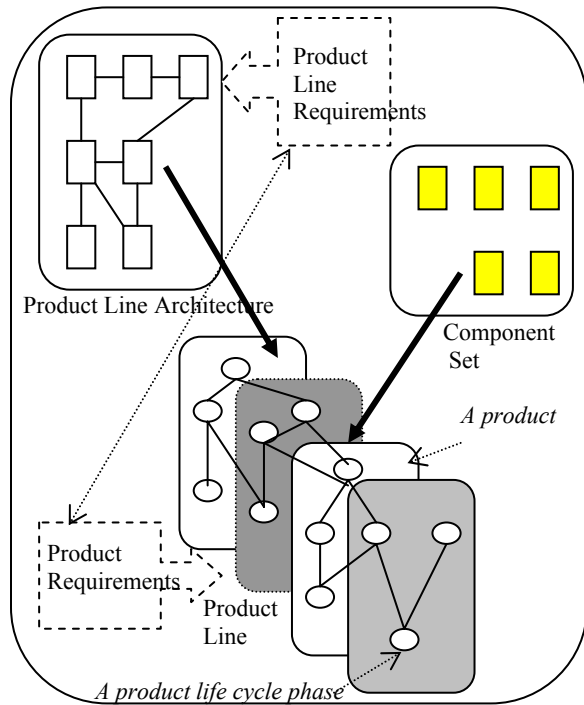


Figure 4.2 – Evolution of software product line

There are relationships between the product line architecture and the product line, and between the component set and the product line. Using the dependency relationships, we build a set of dependency structures as follow [8]:

DEPENDENCY-STRUCTURE-list =  
 {Product-line-Architecture-Dependency-list,  
 Component-set-Dependency-list,  
 PRODUCT-LINE-DEPENDENCY-list}

PRODUCT-LINE-DEPENDENCY-list =  
 {Product-Dependency-list<sub>1</sub>,  
 Product-Dependency-list<sub>2</sub>,  
 ...  
 Product-Dependency-list<sub>n</sub>}

PRODUCT-DEPENDENCY-list =  
 {Intra-Dependencies,  
 Inter-Dependencies}

The DEPENDENCY-STRUCTURE-list is then used to calculate the impact of change and propagation. The Software Product Line Evolution Process consists of the following (figure 4.3):

- **Change Process Manager** – The process manager sets the goals, regulates the change process, and handles all the dynamics of the effects of change. This includes activating the necessary operations based on the type of change and the consequences; verifies constraints attached to the execution of

impact analysis function and the propagation process; and gives assistance to the user by checking the consistency of the bases as well as recording the history of change.

- **Software Product Lines** – This consists of products, component set, product-line architecture, the set of core assets in the product line, and the relationships between them.
- **Core Assets Base** – This base consists of all the core assets, their properties, and historical factors (attached to each of the asset in case of reuse). In the base, all the assets have uniform representation in order to facilitate their interpretation and evolution.
- **Dependency-Structure-list Base** – This base consist of all the artifacts involved in the product line families and their relationships. These include product line architecture, component-set, and product-lines. It consists of a database of all the objects, a fact base of all the direct dependency relations, and a knowledge base containing predefined indirect relations and integrity constraints. The base is persistent because each “entity” has a unique internal identification.
- **Change Management Tool** – The change management tool is based on a model called “What-If” [1, 2]. *What-If* evaluates *à priori* the impact of object change in a software system. The model is generic and it covers the entire product life cycle. It is based on a “mixed software views” approach. In this approach, object change is explained in terms of life cycle view, combination of life cycle phases (sub-views), and the domain specific view. The domain specific view allows the specification of fine grain objects and the life cycle view allows the specification of medium and large grain granularities. Impact analysis is *à priori* in What-If model, which means that one can calculate the effects of a change before it is even carried out. This is particularly useful because if a new requirement is going to cost too much in terms of managing the changes that it is introduced into the product line, then, it is better to develop a new product line and treat the requirement as variability.

## 5. Conclusion

In traditional software engineering, a sizable amount of money and time is spent doing software product evolution. This is so because of the complicity involved in the maintenance of software. For software product lines, the evolution of assets in the product line is even more complex compared to the traditional approach. This is so because most assets depend on multiple software systems. In addition, multiple organization units and business models are involved, and these contribute to the complexity. In this paper, we have presented a modeling approach that will enhance the product-line evolution management. The change management activity encompasses the dynamics involved in handling changes and the effects on the product line families.

Our approach in this paper will aid both the developers as well as decision makers to handle changes effectively by

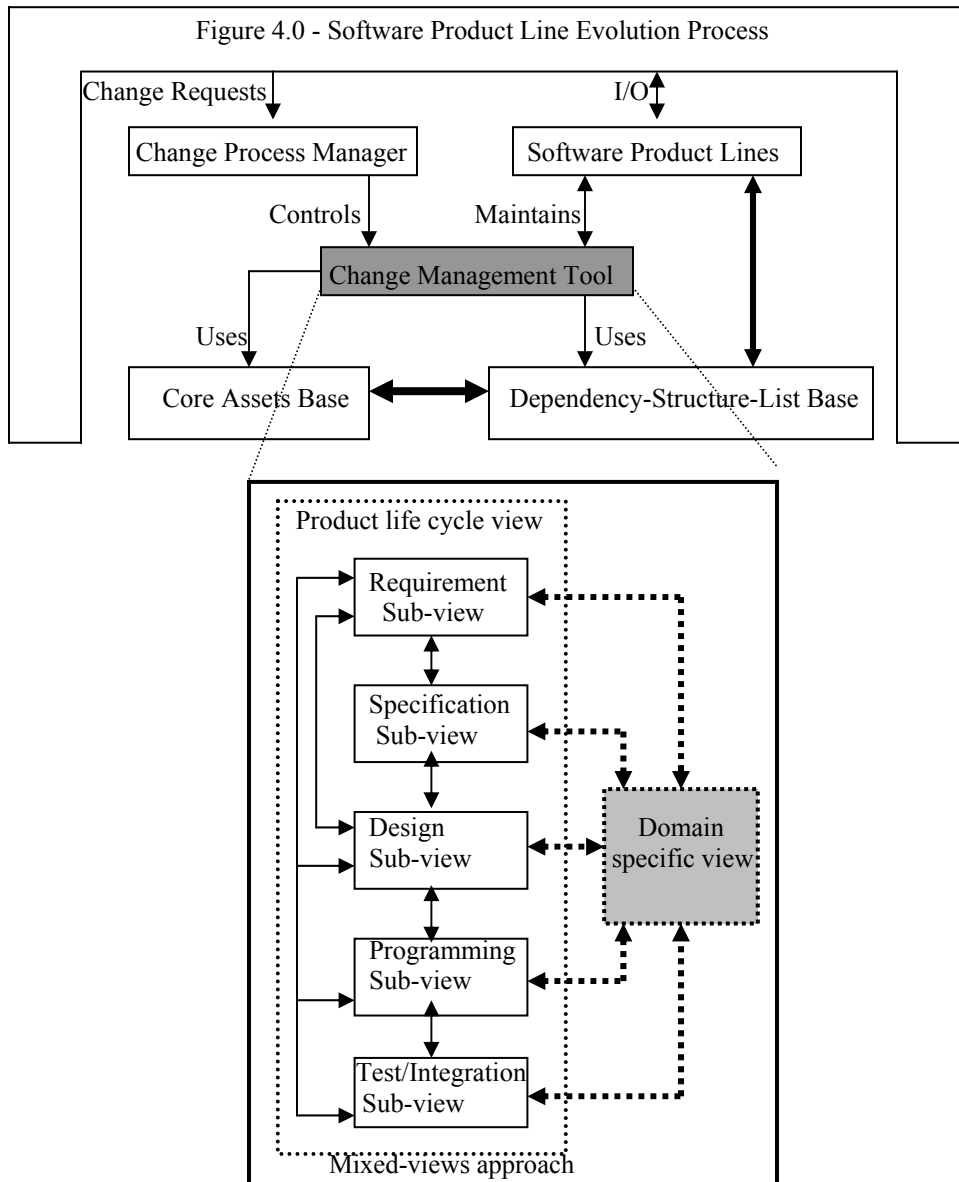


Figure 4.3 – Product Line Evolution Model

been able to calculate the impacts and propagate the effects of the changes. In addition, the approach in this paper will reduce substantially the cost of products maintenance. This is in line with the main goal of product line engineering which is, to reduce cost and increase customer satisfaction. Our approach in this paper did not include how to handle evolution when there is reuse.

## Bibliography

- [1] Samuel A. Ajila, Software Maintenance : An Approach to impact Analysis of Objects Change, In International Journal of Software - Practice and Experience, Vol. 25(10) , pp. 1155 – 1181, October 1995
- [2] S. A. Ajila, 'Dealing with Impact Analysis of Objects Change in a Distributed Team-Based Software Development: Basic Concepts and Perspectives', In Process support for Distributed Team-based Software Development (PDTSD'99) in collaboration with 3<sup>rd</sup> World Multiconference SCI'99 & ISAS'99, Vol. 2, 531 – 536, July 31 – August 4, 1999, Orlando, Florida, USA.
- [3] S. A. Ajila, 'Specification of Dependency Relations in Software Change Management', In Proceedings of Software Development Theory, Practice and Experience – Issues and Challenges for the 21<sup>st</sup> Century, University of Botswana in collaboration with UNU- International Institute for Software Technology, pg 48-61, September 2000, Gaborone, Botswana.
- [4] Jan Bosch, Design & Use of Software Architectures – Adopting and evolving a product-line approach, Addison-Wesley, 2000.
- [5] Bernd Bruege and Allen H Dutoit, Object-Oriented Software Engineering – Conquering Complex and Change Systems, Prentice Hall, 2000.
- [6] Grady Booch et al., The Unified Modeling Language User Guide, Addison-Wesley, 1999.
- [7] Paul Clements and Linda Northrop, Software Product Lines – Practices and Patterns, Addison-Wesley, 2002.
- [8] A Kaba, 'Des mécanismes pour l'évolution des procédé de développement de logiciels, Ph.D. thesis, Département de formation Doctrale en Informatique, Centre de Recherche en Informatique de Nancy (CRIN-CNRS now LORIA-CNRS), Institut National Polytechnique de Lorraine, Nancy, France, July 1996.
- [9] M. M. Lehman and L. A. Belady, 'Program evolution: Process of software change. APIC Studies in Data Processing No 27, 1985, Academic Press.
- [10] Stephen R. Schach and Amir Tomer, Development/Maintenance/Reuse: Software Evolution in Product Lines, In Software Product Lines – Experience and Research Directions, Edited by Patrick Donohoe, Proceedings of the 1<sup>st</sup> Software Product Lines Conference (SPLC1), pg 437-450, August 28-31, 2000, Denver, Colorado, USA.