I J C T A, 9(25), 2016, pp. 301-322 ISSN: 0974-5572

Comprehensive Configuration Management Model for Software Product Line

Elham Darmanaki Farahani* and Jafar Habibi**

ABSTRACT

In Software Product Line (SPL), Configuration Management (CM) is a multi-dimensional problem. On the one hand, the Core Assets that constitute a configuration need to be managed, and on the other hand, each product in the product line that is built using a configuration must be managed, and furthermore, the management of all these configurations must be coordinated under a single process. Therefore, CM for product lines is more complex than for single systems. The CM of any software system involves four closely related activities: Change Management (ChM), Version Management (VM), Build Management (BM) and Release Management (RM). The aim of this paper is to provide a comprehensive CM model comprising four main sub-models for all CM-related activities required for evolutionary based SPL system development and maintenance. The proposed models support any level of aggregation in SPLs and have been applied to Mobile SPL as a case study.

Keywords: Software Product Line, Configuration Management, Change Management, Version Management, Release Management, Build Management.

1. INTRODUCTION

SPL refers to a set of software-intensive‘ products that: (i) share a common, managed set of features, (ii) satisfy the specific needs of a particular market segment (domain), and (iii) are developed from a common set of Core Assets in a prescribed way [1]. SPL is a two-phase process. The first phase is Domain engineering which composes the general part or baseline of SPL. The second phase is the Application engineering of individual applications.

Change is a fact of life in any software system and can manifest itself in two ways. First, through Corrective Changes to fix the bugs that may be detected in the system. Second, via Effective Changes to deal with alterations in the requirements and environment of the system. To ensure that the changes are applied to the product in a controlled manner, a CM model is needed[2].

Change requests may come from both the system stakeholders and the system development team and may relate to a range of issues, namely: (i) existing requirements that have not yet been implemented in the released system, (ii) new requirements and bug reports from stakeholders, and finally (iii) new ideas for software improvement from the system development team. The processes of change identification and system evolution are cyclic and continue throughout the lifetime of a system. Change proposals should be linked to the system components that have to be modified to implement those proposals.

CM is, of course, an integral part of any software development activity and takes on a special significance in the product line context. Development of a single product is well understood. But when we have multiple products that share much of the same functionality, development complexity increases an order of magnitude

* Kish International Campus, Sharif University of Technology Tehran, Iran, *Email: efarahani@ce.sharif.edu*

** Department of Computer Engineering, Sharif University of Technology Tehran, Iran, *Email: jhabibi@sharif.edu*

compared to single product development. As has been mentioned in the future work section in [3] (which is the main reference for SPL)[4], it is difficult to make changes to a single system and much more difficult to a product line.

Applying any changes to SPLs is more complex than the traditional software product evolution because most Assets are relied upon by multiple products. 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.

In the case of a single product, the evolution of the product is in the time dimension. A single product is maintained as one evolving entity. In contrast, in SPL, products evolve independent of the components shared among them. Products and Assets have their own line of development. The evolution of Assets is said to be in the time dimension while the evolution of products is said to be in the space dimension [4]. Here, space refers to the product space in the product line. The Asset developers and the product developers may not be the same people. Even the developers of different products may not be the same people. Sharing of Assets across products creates a network of product and Asset dependency. A change in an Asset may affect multiple products as well as the Assets that depend on the changed one. For example, changes in the interface of an Asset could break products and other Assets using it. Four main challenges of CM in SPLs are as follows:

(1) The ChM process in SPL must control the changes to all artifacts under CM, especially the Core Assets. Due to the inter-dependencies between the Assets and the products, changes to either of them may affect the evolution of the other. On the other hand, since two products may share the same Core Assets, changing one of them may also affect the other.

(2) Additionally, the benefits of SPL come from the reuse of Core Assets. If changes to software artifacts are not well controlled, this may cause the Core Assets to deviate from the general architecture or the product may lose its connection with the Core Assets. Both cases will destroy SPL.

(3) Changes in customer requirements and errors detected in product variants may require the platform itself to be modified. Given that all the members of the product line will be affected by modifications made to the platform, it is often allowed to modify the platform independently for different market segments such that only the product variants that are immediately affected by the modification are tested and rebuilt. To achieve this, the platform is branched into a family of platforms.

(4) In an SPL there are multiple products which are simultaneously going through their own cycles of release and versioning. So RM in SPL is a complex process. Special releases of the system may have to be produced for each customer and individual customers may be running different releases of the system at the same time. This means that a software company selling a specialized software product may have to manage tens or even hundreds of different releases of that product. Their configuration management systems and processes have to be designed to provide information about which customers have which releases of the product and also what is the relationship between releases and product versions [2].

In addition to the above CM challenges, the CM of a software product involves four closely related activities: Change Management (ChM), Version Management (VM), Build Management (BM) and Release Management (RM) [5].

The main objective of this paper is to complete our previous ChM and VM models [5] and propose two new models for RM and BM in SPLs and to integrate these four models into a comprehensive CM model. The proposed models support all types of Assets in any SPL. We analyze these models in the context of one case study and show its powerful features. The rest of this paper is organized as follows. Section 2 defines the terminology used in this paper. Section 3 presents the background and latest research on CM in SPLs.

Section 4 describes each of the four proposed models for SPL, and section 5 analyzes the proposed models in the context of one sample case study and presents our solution in detail. Finally, the conclusion comes in section 6.

2. RELATED CONCEPTS

To avoid ambiguity and misunderstanding, this section defines the terminology used in this paper.

Component: A Component is the basic unit in CM. An example of a Component is a single file. A set of files that collectively perform a function or form an inheritance tree is also called a Component. A Component could be atomic or composite. However, in this paper, we treat both of them as the smallest configuration artifact[3].

Asset: An Asset is a collection of one or more Components. The relation between the Components and Assets is the part-whole relation. There are two types of Assets in SPL, Core and Variable[6].

Core Asset: A Core Asset contains a set of domain specific but application independent Components that can be adapted and reused in various related applications. Core Asset is one of the most important concepts in SPL. Generally, a Core Asset almost includes both the architecture of the products that will be shared in the product line architecture, as well as the Components that are developed for systematic reuse across the product line.

Variable Asset: A Variable Asset contains a set of Components that are specified for an application. It is produced for a specific application and is not involved in the Asset reusing process. The quality requirement (for example, reusability) of Variable Assets is not as high as Core Assets and the effort spent on their maintenance is less than the Core Assets[7].

Product: From a logical view, a product is a collection of Core and Variable Assets. SPL takes Core and Variable Assets as input and produces a product as the output. Different products share the same or similar Core Assets, but the Core Assets need to be adapted to meet the requirements of each specific product. A product can be logically considered to contain two parts, a Core part and a Variable part, which are built from the Core Assets and Variable Assets, respectively.

Feature Model (FM): Feature Models are simple, tree models that capture the commonality and variability of a Product Line. Each property of the problem space becomes a Feature in the model. This means that a Feature is a characteristic of a system from the point of view of some stakeholder. A Feature can be a requirement or a functional or non-functional (quality) characteristic that is dependent on the customer request. Relationships between a parent Feature and its child Features (or sub-Features) are categorized as: Mandatory (child Feature is required), Optional (child Feature is optional), Or (at least one of the child Features must be selected) and Alternative (xor) (at most one of the child Features can be selected)[1].

Corrective Changes: Corrective Changes are modifications made to an artifact in order to remove a residual bug while leaving the specification unchanged. A Corrective Change is important because, without it, the product cannot function correctly as specified in the functional requirement. A bug is an error, flaw, failure, or fault in a software product that causes it to produce an incorrect or unexpected result, or to behave in unintended ways. Most of the bugs arise from mistakes and errors made by people in either a program's source code or its design, or in frameworks used by such a program. Corrective Changes may lead to the creation of a new version of the product[8].

Effective Changes: Effective Changes are modifications made to an artifact to meet a new requirement for: (i) enhancing the quality of a product such as its security, performance and usability, (ii) improving its functionality, and (iii) dealing with changes to the business model. Effective changes may lead to the generation of a new version of the product.

Version: An instance of a configuration item (Asset, Feature, Feature Model, Product) that differs, in some way, from other instances of that item. Each version is assigned an identification number, consisting of a major number and a minor number. The major number is changed with any effective changes while the minor number is changed as a result of any corrective changes.

Baseline: A Baseline is a collection of Asset (Core and Variable) versions that make up a product. Baselines are controlled, which means that the versions of the Assets making up the product cannot be changed. This means that it should always be possible to re-create a baseline from its constituent Assets[6].

Mainline: A sequence of Baselines representing different versions of a Product.

Release: A version of a Product that has been released to customers for use.

3. BACKGROUND

In this section, we present the methods, models and proposals concerning the issues of CM in SPLs. Table 1 summarize the activities conducted in relation to CM in SPLs[3].

Clemente in [9] proposes the general CM model of SPL. In this model, the artifacts under CM include Core Assets, Variable Assets and product instances. Component is defined as the basic unit for CM. The proposed model lacks the capability of effective change management because it does not differentiate between corrective and effective changes.

Van Ommeringin [10] describes the Koala component-based software system and the conventional CM systems for implementing a set of products. In this approach, each new release of packages should be backward compatible. But the proposed solution is not a general solution and is built on Component-based Koala solution.

In [11] presents the specific requirements of CM for product lines and proposes to utilize CM systems as the basis for an automated product derivation process. His approach envisions systems that construct products from common artifacts and allows the change of these artifacts in the implementation of products. These changes can be later re-incorporated to the common artifacts in the product line in an automated fashion. However, he does not propose any ChM process model. He also suggests product release plans in SPL organizations to deliver products according to release schedules. He states that the key to managing product releases is in synchronizing the release of features in core assets with the product release schedules. In order to release a product at a given point in time, the core assets used in the composition and configuration of that product must meet or exceed the required functionality and quality for that product release. Product managers in this scenario are not managing teams of application engineers to meet their release schedules. Instead, they manage their release schedules by coordinating and negotiating with architects and core asset developers to make sure that the core assets meet or exceed the required functionality and quality. From the perspective of the core asset teams, their release schedules are driven by the consolidated requirements and schedules from all of the products.

Van Deursenin [12] describes a package-based product line which is similar to Van Ommering's and Krueger's approaches; however, this is not generic since it relies on the Koala technologies. In Krueger's approach, there's no separation between the core developers and the product developers. Since only the production is under Software Configuration Management (SCM), it does not allow for large numbers of independent product evolution branches that are maintained under SCM. In Van Ommering, Krueger and Van Deursen approaches [5], [10], forward propagation is automatic. Since changes occur in the Core Assets[6], products that use the latest Assets will receive the new changes. Furthermore, dependency among components and products are manually maintained.

In [13], Ajila et al. propose to utilize traceability information for supporting SPL evolution. The authors argue about the necessity of traceability relationships in SPLs in general and propose to use impact analysis

for determining a generalized set of changes. This set of changes should be monitored by interested stakeholders to assess the risk that they may pose to SPL. They claim that traceability is necessary for analyzing the impact of change in the evolutionary SPL environment. They also suggest a Dependency-structure that represents the dependency between artifacts. It consists of: (i) Change Monitoring Structure that defines the strategies for the evolution process and change scope, and (ii) Product Line Reference Space for controlling the structure during evolution. While their approach provides a traceability management scheme between various artifacts for SPL and its derived products, it requires extra documents and algorithms to establish traceability. These extra artifacts need additional efforts to develop and manage the traceability[14].

Taborda in [15] attempts to understand the relevance of current software management best practices by utilizing a simple meta-model to illustrate the impact that architectural complexity and reusable components have on software management patterns. The meta-model serves as a heuristic device and supports the view that products based on a framework of reusable software components pose new challenges and have to be managed simultaneously at a number of different levels. This provides a rationale for the Release Matrix, a generalization of a software release plan, as a technique for managing software product lines. The Release Matrix has practical applications for tracking the evolution of complex component-based systems and has been shown via the model to be a natural consequence of increased architectural complexity and component reuse.

In [9] define the problems that arise in SCM for SPL and propose very general solutions to lessen their impact. In Staples' proposal, generic SM systems will suffix.

In [11] present an evolution-based CM model for SPL. They provide the two sectional models (production and product domains) to improve the previously proposed model. However, they do not present any versioning model for SPLs.

In [16] introduce a configuration tool to develop and maintain resources of the SPL-based systems and multiple products derived from SPL. Additionally, they present the collision case when SPL and its products evolve separately and describe how to deal with their separate evolution using the proposed configuration tool. They utilize several source code CM tools (e.g., Subversion [9]]) for managing SPL and its derived products. This approach does not provide the resource versioning scheme for building SPL-based systems composed of massive number of software artifacts. Furthermore, they do not tackle diverse resources in developing SPL and its products.

In [17] Mohan and Ramesh suggest some practices for ChM in SPL but they do not consider VM or any ChM method in SPL.

In [18] present an approach for CM in an SPL. The approach, which is added to the traditional CM practices, manages information, including changes, about assets and products ensuring that they are exactly what they were intended to be. The proposed CM system supports product releases. Whether there are formal, annual releases or almost continuous releases including nightly builds, each publicly available product is a configuration that is versioned and can be recreated at any time. Their experience shows that many organizations do not employ comprehensive approaches for dealing with what constitutes a release. Important documentation such as the entire test framework for the release is often overlooked because of the cumbersome nature of how such assets are used. Their technique uses references to versions of these assets rather than copying the assets.

In [19] present an approach for both resource versioning in SPL, and also how to propagate resource changes into related resources. They propose a version control technique for SPL features. They classify resource versions with feature logical version, feature container version and artifact version, and then apply this approach to diverse SPL evolution scenarios. However, they only discuss version relations between

features and the source code without considering mandatory artifacts such as domain and design models. This shortcoming prevents the delivery of product specific artifacts to customers.

Kroon in his master's thesis [18] proposes a layered model for CM of SPLs and creates a prototype CM tool based on the proposed model. One of the major drawbacks of this model is that there is no separation between Core and Variable Assets. In fact, there is no difference between the two Asset categories and this is the fundamental weakness of this model.

In [6] identifies the considerations for applying product-line development and finds out how to develop a usage model of configuration and build management in SPL. Also, he proposes the Daily Builds and release process. He states that the following features can be supported by any build automation tool: (i) the latest label of each asset, (ii) the latest composite baseline of a product, and (iii) the specific label/change list of each asset.

Thao in his doctoral thesis [20] proposes a layered model for CM of product lines. In his model he does not consider Krueger's discussion concerning CM and different grain types.

Since the main focus of this paper is on four main activities of CM, namely, ChM, VM, RM and BM, for SPL, in Table 2 below we compare the previous works in the field of CM for SPL from the viewpoint of whether they propose support for ChM, VM, RM and BM models for SPL. As can be seen, none of them propose all four models for SPL.

Table 1
Summary of activities in the field of CM in SPL.

No	*Author*	*Activities Carried out*	*Challenges and Shortcomings*
1	Clemente	Provides the General Model of CM in SPL	Lack of difference between Corrective and Effective Changes
2	Van Ommering	Describes the CM process in Philips Product Line based on Koala	Not a general solution. Based on Component-based Koala solution
3	Krueger	Improves the General Model of CM in SPL. Divides the CM problem into nine sub-problems and provides a solution for each one	Does not cover all Asset types that can be used in the product line. Does not offer any Process Model for ChM in SPL
4	Deursen	Describes the Feature-oriented CM in DOCGEN Product Line	Dependencies between components and products must be maintained manually because it cannot be achieved automatically.
5	Ajila	Modeling SPLS evolution	Does not provide the details of the proposed model of ChM
6	Taborda	Proposes a Release Matrix, a generalization of a software release plan, as a technique for managing software product lines	The presented technique is only a concept that requires further exploration and detailing
7	Staples	Describes the four problems involved in applying the traditional CM methods to a SPL and provides a number of solutions	It is only a suggestion
8	Yu et al.	Provides the two sectional models (namely, production domain and product domain) to improve the previously proposed model	Does not provide any VM model in SPL
9	Van Gurp	Combines Subversion (the traditional CM tool) with variability Concept and provides a SPL CM tool	Does not support diverse resources in developing SPL and its products.
10	Mohan and Ramesh	Suggests some practices on ChM in SPL	Does not provide any Process Model for ChM

(*contd...*)

(*Table 1 contd...*)

No	*Author*	*Activities Carried out*	*Challenges and Shortcomings*
11	Kurman	Presents a Release strategy for SPL	The presented strategy does not cover all the problems in CM
12	McGregor et al.	Presents an approach for CM in an SPL	Does not propose any CM model
13	Mitschke et al.	Presents an approach to resource versioning in SPL	Did not consider mandatory artifacts such as domain and design models
14	Rahmanet al.	Discusses various CM activities in practice	Does not propose any CM model
15	Kroon	Development of a prototype of a CM tool based on the proposed Layered model	No separation between Core Assets and Variable Assets
16	SEI	Describes the CM as a necessary part in SPL Framework	Does not propose any CM model
17	Kim	Develops a usage model of configuration and build management in SPL	Does not describe any details about the proposed model
18	Thao	Design and implementation of MaLhado (a SPL CM system)	No Process Model for ChM and VM
19	Ko et al.	Proposes a versioning scheme in evolutionary SPL	No Process Model for ChM. No separation between Core Assets and Variable Assets

Table 2
Comparison of previous works, in the field of CM for SPL, proposing support for ChM, VM, RM and BM models.

No	*Author*	*Proposes ChM Model*	*Proposes VM Model*	*Proposes RM Model*	*Proposes BM Model*
1	Clemente	✓	–	–	–
2	Van Ommering	✓	–	–	–
3	Krueger	✓	–	–	–
4	Deursen	✓	–	–	–
5	Ajila	–	–	–	–
6	Taborda	–	–	✓	–
7	Staples	–	–	–	–
8	Yu and Ramaswamy	✓	–	–	–
9	Van Gurp	✓	–	–	–
10	Mohan and Ramesh	✓	–	–	–
11	Kurman	–	✓	✓	–
12	McGregor et al.	–	✓	✓	–
13	Mitschke et al.	–	✓	–	–
14	Rahman et al.	–	–	–	–
15	Kroon	–	–	–	–
16	SEI	–	–	–	–
17	Kim	–	–	–	✓
18	Thao	–	–	–	–
19	Ko et al.	–	✓	–	–

4. PROPOSED METHOD

A successful CM model requires a well-defined and institutionalized set of policies and standards that clearly define:

(1) The set of artifacts (configuration items) under the jurisdiction of CM

(2) How the artifacts are named

(3) How an artifact under CM is allowed to change

(4) How different versions of an artifact under CM are made available and under what conditions each one is allowed to be used

(5) How products are prepared for external release and how we can keep track of the product versions that have been released to different customers

(6) How we can assemble product components, data, and libraries, and then compile and link these to create a working product

These policies and standards must be documented in a CM plan that informs everyone in the organization about how CM is carried out. In this section, we answer the above questions regarding policies and standards. In section 4.1, we complete our previous VM model [24] to cover items 1 and 2, and in section 4.2 we review our previous ChM model [24] to answer items 3 and 4. Finally, in section 4.3 we propose RM and BM models to cover item 5 and 6. Also, we show how we can use the proposed CM model for evolutionary-based SPL system development and maintenance.

4.1. Proposed VM Model for SPL

As stated in the previous section, a CM model must establish and document some policies for a set of artifacts (configuration items) under CM and describe how the artifacts are named. In this section, we propose a VM resource model for SPL and specify how we can name the elements of SPL in any level of aggregation.

In our previously proposed CM model we consider the following SPL resources: Feature Model, Feature, Product, Core Asset, Variable Asset, Release and Customer. Table 3 shows a version identifier, enclosed in parenthesis, for each SPL resource in the VM model[20].

Also in SPL, the Feature Model is positioned at the center of the Resource VM model. The Feature Model consists of several Features and each one is implemented by one or more Core/Variable Assets and also each Asset could be related to more than one Feature. Depending on the Feature selection, each product is generated from the Features (and accordingly from the related Assets) of SPL and also a Feature could be selected by more than one product. Furthermore, each product may have more than one version that is released to customers as a product release.

Table 3
Version Identifier for SPL Resources.

Resource	*Version Identifier*
Feature Model(FM)	(FM Name, FM Version)
Feature(F)	(F Name, F Version)
Product(P)	(P Name, P Version)
Core Asset(CA)	(CA Name, CA Version)
Variable Asset(VA)	(VA Name, VA Version)
Release (R)	(Customer-Name, R Number)

Figure 1 shows the relations between the SPL resources, and Table 4 presents the identifiers for those relations. For example, (F_A, 1.0) | (CA1, 1.1) indicates that version 1.1 of Core Asset CA1 is related to version 1.0 of Feature F_A, and hence, any product that selects Feature (F_A, 1.0) must also use (CA1, 1.1).

In this VM model, two types of version change rules are introduced: Minor Version Change and Major Version Change. As mentioned earlier, the version number of every element has the *Major version. Minor version* format, for example, version 1.1. The Minor version number is increased after every Corrective Change and this rule is named "Minor Version Change". Similarly, the Major version number is increased after every Effective Change in SPL elements and Products and this rule is called "Major Version Change".

In SPL a Feature resides at the center of all artifacts including source code. Depending on the Feature selection, the related artifacts should be extracted from SPL in order to instantiate a product. According to Table 4, the version identifier of a Feature of SPL is composed of *(FM Name, FM Version)* | *(F Name, F Version)*, where the first part is the product line Feature Model version, and the second part denotes the Feature version. This version identifier is automatically updated when one of the SPL Assets is changed according to the following rules.

Rule 1. The version of Feature Model is identical to the version of Root Feature.

Rule 2. When the artifacts of a Feature are modified, the Feature version is updated.

Rule 3. When a Feature version is updated; the parent's Feature version is updated accordingly.

Rule 4. When the product line version is updated, the product line versions of all Features are updated accordingly.

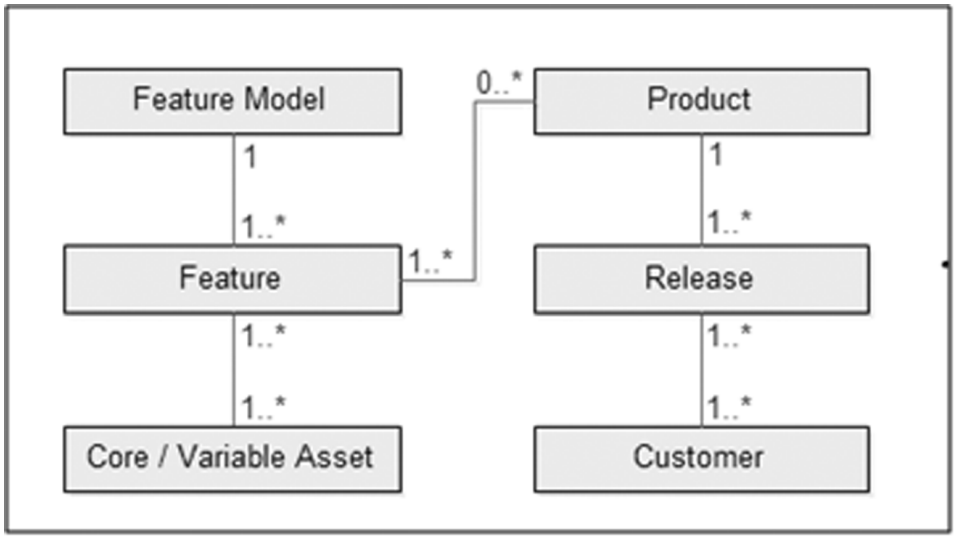

Figure 1: Relations between SPL Resources.

Table 4
Relation Identifier for SPL Resources.

Relation	*Version Identifier*
Feature Model (FM) & Feature (F)	(FM Name, FM Version) | (F Name, F Version)
Product (P) & Feature (F)	(P Name, P Version) | (F Name, F Version)
Feature Model (FM) & Product (P)	(FM Name, FM Version) | (P Name, P Version)
Feature (F) & Core Asset (CA)	(F Name, F Version) | (CA Name, CA Version)
Feature (F) & Variable Asset (VA)	(F Name, F Version) | (VA Name, VA Version)
Product (P) & Release (R)	(P Name, P Version) | (Customer-Name, R Number)

4.2. Proposed ChM Model for SPLs

As stated in [1], the SPL Framework has two main phases: Domain Engineering and Application Engineering. The Domain Engineering phase is concerned with building the product line system, whereas the Application Engineering phase focuses on building the products according to the Features selected from FM. In both phases there is a loop running from the last box to the first box to show the evolutionary nature of this framework. This loop represents an evolutionary phase which handles diverse evolutionary scenarios including the SPL evolution (the loop in the Domain Engineering phase) and product Evolution (the loop in the Application Engineering phase).

In our previously proposed ChM model [9], depending on the type of change, different change propagation methods can be used. In the following section, we discuss the change propagation methods for Assets and present some examples to show how VM and ChM models can be used to handle the Corrective and Effective Changes in the two evolutionary phases mentioned above. Table 5 shows the change propagation paths for various changes to artifacts in SPL.

Table 5
Change propagation paths for various changes to artifacts in SPL [9].

Type of Change	*Evolution in Domain Engineering Phaseof SPL Framework*		*Evolution in Application Engineering Phaseof SPL Framework*	
	Core asset	*Variable asset*	*Core asset*	*Variable asset*
Corrective	Propagation to all products	Propagation to products that support related Feature	Propagation to Domain Engineering phase and also to all products	Propagation to Domain Engineering phase and products that support related Feature
Effective	Available in new products and new version of existing products	Available in new products and new version of existing products	Propagation to Domain Engineering phase and available in new products	Report to Domain Engineering section

4.2.1. ChM Model for SPL Evolution in Domain Engineering Phase

Any change to FM results in modifying SPL. It also requires the system version to be increased and this, in turn, signals that a system update has occurred. The FM can be changed in two ways. First, planned modification can take place to add new Features or update existing ones. Second, the SPL system is updated to correct faults and such updates are not planned. This frequently occurs when the system has critical defects such as performance issues. In the following we describe different scenarios for applying changes in the Domain Engineering phase.

(i) *Corrective Changes to Core Assets:* The Corrective Changes that are made to Core Assets in SPL should also be applied to all products to fix any detected faults in them. After applying a Corrective Change, the minor version of Core Assets, Features, the Feature Model and products are updated. Furthermore, the fault in Core Assets of all products must be fixed.

(ii) *Effective Changes to Core Assets:* If Effective Changes are made to Core Assets, SPL should be updated immediately. Also, the major version number of the Feature Model, Feature and Core Asset must be increased to indicate that a major change has occurred. The Effective Changes to Core Assets do not need to propagate to existing products as they are only relevant to new products and new versions of existing products.

(iii) *Corrective Changes to Variable Assets:* The Corrective Changes made to Variable Assets should also be reflected in the products that use the related Feature of those Assets because it is important

to fix any fault detected in a product. Also the minor version number of Variable Assets, Features, the Feature Model and products must be updated to indicate that a minor change has taken place. Furthermore, the detected faults must be fixed in all products that use the changed Variable Asset.

(iv) *Effective Changes to Variable Assets:* The Effective Changes applied to Variable Assets are relevant only to new products and new versions of existing products; they are not reflected in the existing products. In this case the version number of Variable Asset, Feature and the Feature Model is increased as a major version change.

Figure 2 shows an example of an Effective Change to a Core Asset in the Domain Engineering phase. Initially, the product (P_A, 1.0) is instantiated from (F_A, 1.0) of the Feature Model (FM_A, 1.0) (see 1 in Figure 2). Then, the planned updates (for instance, updating Feature (F_A, 1.0) by applying Effective Changes to Core Asset (CA, 1.0)) are carried out in the FM. This causes the FM_A to go through a major version change and increases its version number from 1.0 into 2.0 (see 2). Note that the Effective Changes to the Core Asset (CA, 1.0)) do not cause the existing product P_A to be updated. However, whenever a new product (for example, Product P_B) is to be created, it must be instantiated from the updated FM, (FM_A, 2.0) (see 3).

Figure 3 illustrates an example of a Corrective Change to a Core Asset in Domain Engineering phase. Suppose that one Corrective Change occurs in Core Asset (CA, 1.0). As a result, the Asset goes through a minor version change and its version changes from (CA, 1.0) to (CA, 1.1). Also the version identities of the related Feature and the Feature Model are changed to (F_A, 1.1) and (FM_A, 1.1), respectively. Since the change occurs in the Domain Engineering phase, all products that use the Changed Asset (CA) should be regenerated. Therefore, the change is propagated to Product P_A, resulting in a minor version change. This causes the version to be changed from (P_A, 1.0) to (P_A, 1.1). Note that in case of an update to correct a fault, there is no difference between Core and Variable Assets.

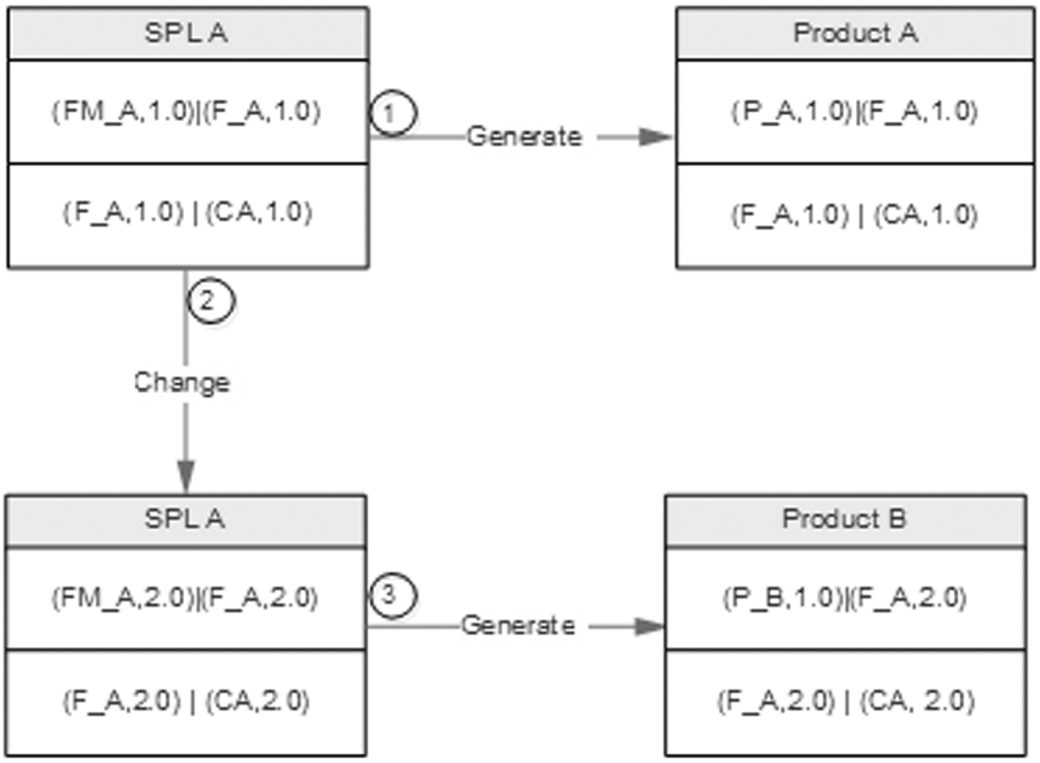

Figure 2: Evolution of SPL (Effective Changes) [24].

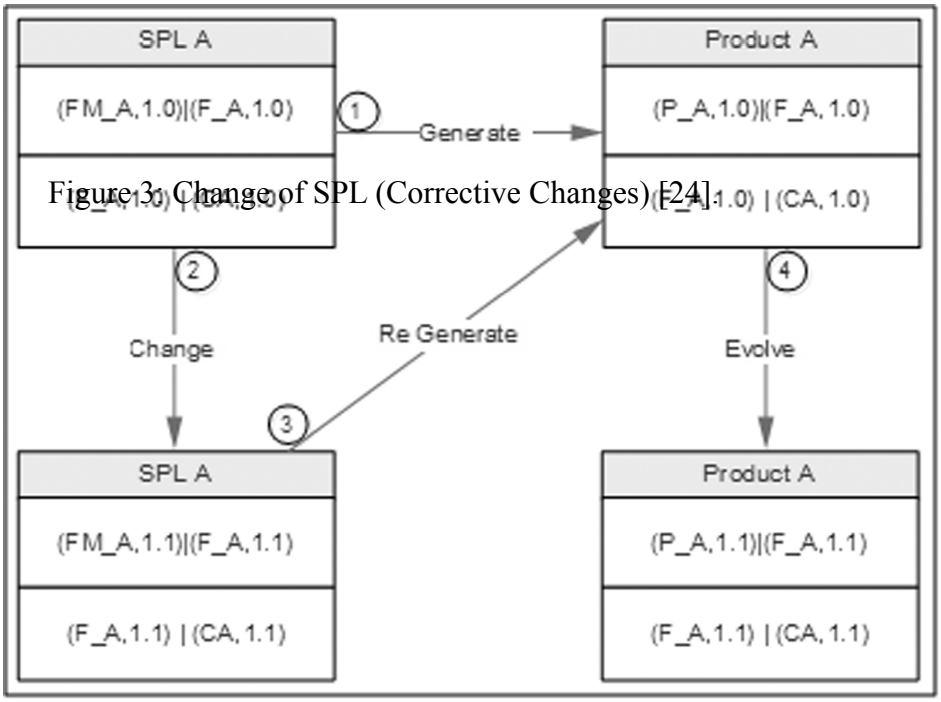

Figure 3: Change of SPL (Corrective Changes) [24].

4.2.2. ChM Model for SPL Evolution in Application Engineering Phase

Product evolution typically occurs as a result of emergency updates or localized changes to the product. In this section, we explain different scenarios for change propagation in the Application Engineering phase[9].

(i) *Corrective Changes to Core Asset of a product:* If a core Asset of a product is updated via Corrective Changes, the product will go through a minor version change and its version number will be increased. These changes should also be reflected in SPL. This means that SPL, Feature and the related Asset will have a minor version change. This is because it is important to fix the fault in all products that use the changed Core Asset. Also, the version number of all regenerated products must be updated.

(ii) *Effective Changes to Core Asset of a product:* If Effective Changes are made to the core part of a product, they should not be reflected to other products but must be reported to SPL. The affected product will go through a major version change and its version number is increased. Furthermore, the updated version of the Core Asset must be added to SPL and the version numbers of the Core Asset, Feature and the Feature Model are increased. This makes it possible for the updated Core Asset to be used in new products.

(iii) *Corrective Changes to Variable Asset of a product:* If Corrective Changes are made to the Variable Asset of a product they must be reflected in SPL and other products that use the related Feature of the changed Asset. In this case, the affected products and the Variable Asset go through a minor version change. Note that the Corrective Change must be applied to the Variable Asset in SPL, and as a result, that Asset will be used in new products.

(iv) *Effective Changes to Variable Asset of a product:* If Effective Changes are made to the Variable Asset of a product they are not required to be reflected to other products and SPL. In this case, the product and Variable Asset go through a major version change and their version numbers are increased accordingly.

Figure 4 illustrates an example of applying a Corrective Change to a Core Asset of a product in the Application Engineering phase. Initially, products P_A and P_B are instantiated from the Product Line (FM_A,

1.0) (see 1 and 2 in Figure 4). When a fault occurs and then fixed in product P_A, the version of product P_A is not changed (see 3). Then, the update specific to the product is reported to the SPL CCB (see 4). Also, the fault must be fixed in the Core Asset in SPL (see 5), and the derived products should be updated accordingly (see 6). Step 6 is crucial for making the derived products consistent with the latest version of SPL. Without this step, the updated product remains conformant with the previous version of SPL not the latest one.

Figure 5 illustrates another example of an Effective Change to a Variable Asset in the Application Engineering phase. Suppose that an Effective Change occurs to Variable Asset (VA, 1.0). The Asset goes through a major version change and its version label (VA, 1.0) changes to (VA, 2.0). Since the change has occurred in the Application Engineering phase and it is an Effective Change, it is not required to propagate this change to other products and SPL.

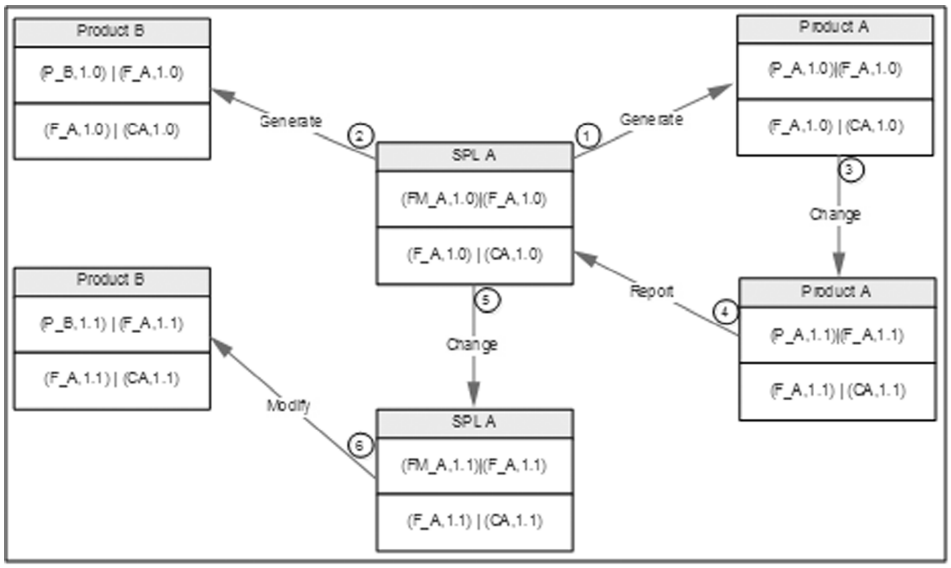

Figure 4: Change of Product (Corrective Changes to Core Asset) [9].

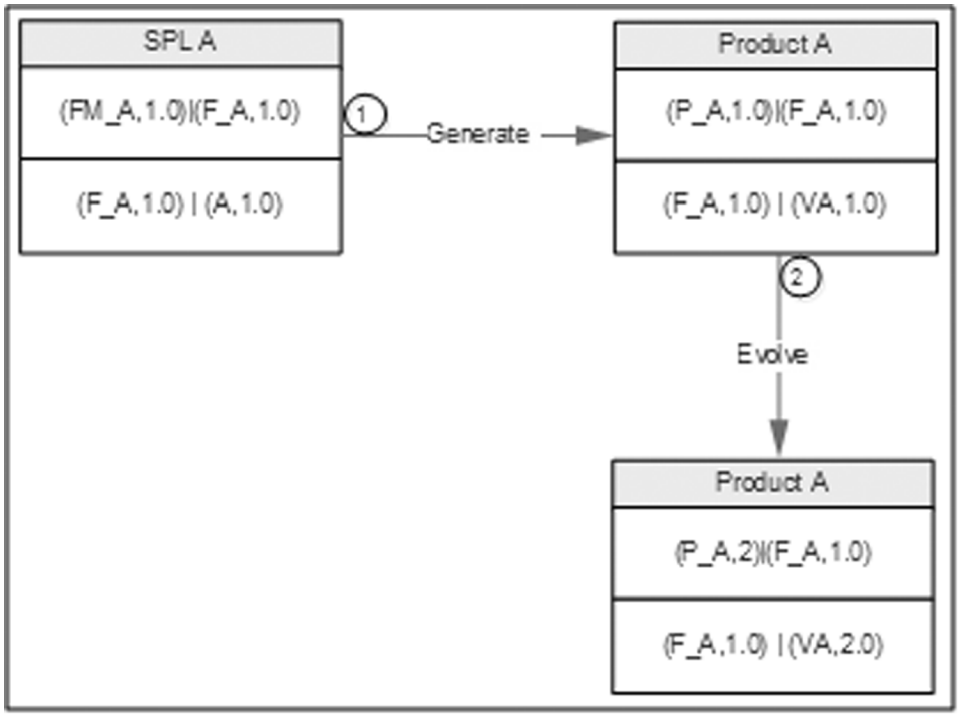

Figure 5: Evolution of Product (Effective Change to Variable Asset) [9].

4.3. Proposed RM and BM Model for SPLs

RM (Release Management) is concerned with building the final customer solution. BM (Build Management) refers to creation of a version of product, for the purpose of integration or testing. In SPLs, RM is often involved in the release of different products to customers and is therefore a complex process. In the event of a problem, it may be necessary to reproduce exactly the same software that has previously been delivered to a particular customer. Each publicly available product is a configuration that is versioned and can be recreated at any time regardless of whether there are formal annual releases or almost continuous releases including nightly builds. RM and BM in SPL are faced with four challenges, the first three are related to RM and the last one is related to BM:

(1) When must a new version of a product be released to customers?

(2) How could we maintain the relationship between releases and product versions?

(3) How could we maintain the relationship between releases and customers?

(4) How could we reproduce exactly the product that has previously been delivered to a particular customer?

Preparing and distributing a product release is an expensive process, particularly for SPLs. As well as the technical work involved in creating a release distribution, advertising and publicity material have to be prepared and marketing strategies put in place to convince customers to buy the new release of the system. Careful thought must be given to release timing. If releases are too frequent or require hardware upgrades, customers may not move to the new release, especially if they have to pay for it. If system releases are too infrequent, market share may be lost as customers move to alternative systems.

The various technical and organizational factors that you should take into account when deciding on when to release a new version of a system are listed in [1]. These include: technical quality of the system, platform changes, Lehman's fifth law, competition, marketing requirements and customer change proposals. But in this paper the challenge is on the last technical factor: customer change proposals. In the case of customized systems, customers may have requested and paid for a specific set of system change proposals, and they expect a system release as soon as these have been implemented. So for SPLs, we can identify two types of release namely major releases, which deliver significant new functionality, and minor releases, which repair bugs and fix customer problems that have been reported. Therefore we propose that after each minor version change of a product we must have a minor release, and after each major version change of a product a minor release must be delivered to the customers.

When planning the installation of new product releases, we cannot assume that customers will always install the new product releases. Some customers may be happy with an existing release of the product and may not consider changing to a new release worthwhile. Therefore, it cannot be assumed that previous releases of a product are already installed by the customer when a new release is delivered to the customer. To illustrate this problem, consider the following scenario:

(1) Release 1 of a product P_A is given to customer C_A and put into use (P_A, 1.0) | (C_A, 1).

(2) Another customer C_B that uses P_B requests an effective change that leads to a major change on a variable asset which causes product P_A to have a major version change as well. These changes lead to publishing major releases of P_A and P_B, {(P_A, 2.0)| (C_A, 2.0) and (P_B, 2.0)| (C_B, 2.0)}, respectively. However, some customers may not need the facilities of release 2 so will not install it and will continue using release 1.

(3) Release 3 requires the changed variable assets provided in release 2

The software distributor cannot assume that the variable assets required for release 3 have already been installed in all sites. Some sites may go directly from release 1 to release 3, skipping release 2. Some sites may have modified the variable assets associated with release 2 to reflect local circumstances. Therefore, the variable assets must also be distributed and installed with release 3 of the product.

Other challenges of RM in SPL include how we could maintain the relationship between (i) releases and product versions and (ii) releases and customers. According to table 4, the version identifier of a product release is composed of (P Name, P Version) | (Customer-Name, R Number), where the first part is the product version, and the second part denotes the customer name and release version. In this way, we could easily find out the release number of the latest version of a product delivered to a specific customer.

Let us consider an example Figure 6. In this example we illustrate a situation where a Corrective Change is applied to a Core Asset of a product in the Application Engineering phase. Here our intention is to show how we can maintain the relationship between release and product versions and also between releases and customers named in Figure 6. Initially, two products P_A and P_B are instantiated from the product line (FM_A, 1.0) (see 1 and 2 in Figure 6). These are released to four customers C_A, C_B, C_C and C_D (see 3 and 4 in Figure 6). When a fault is detected and then fixed in product P_A, the version of product P_A is changed with a minor version change (see 5). Subsequently, the update specific to the product is reported to the SPL CCB (see 7), and as a result, a minor release must be delivered to customers (see 6). Following this, the fault must be fixed in the Core Asset in SPL (see 8), and finally, the derived products should be updated accordingly (see 9) and released to customers with a minor release (see 10). Step 9 is crucial for making the derived products consistent with the latest version of SPL. Without this step, the updated product remains conformant with the previous version of SPL not the latest one.

The last challenge is how we could reproduce exactly the product that has previously been delivered to a particular customer. The reproduction process is performed by BM and involves checking out asset versions from the repository managed by the VM system. The configuration description used to identify a baseline is also used by BM. We suppose that VM uses the concepts of a public repository and a private workspace. Developers check out assets from the public repository into their private workspace and may change these as they wish in their private workspace. When their changes are complete, the assets are checked in to the repository.

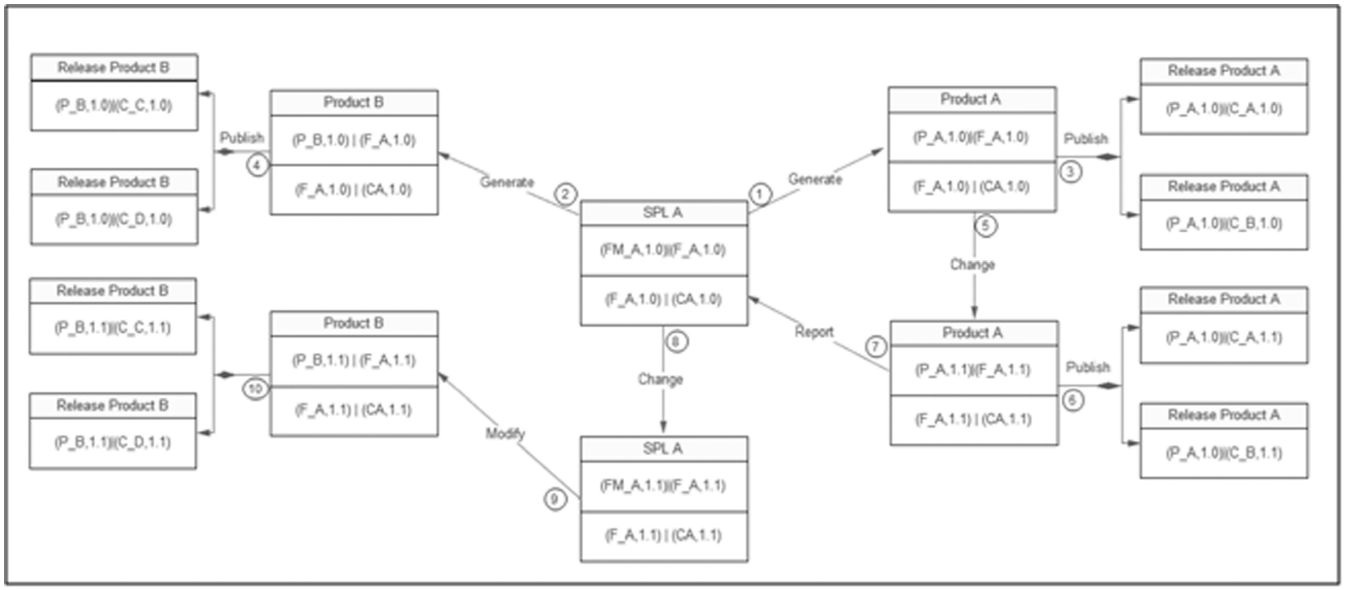

Figure 6: Minor release change in SPL[9], [21].

A baseline is the definition of a specific product. The baseline therefore specifies the asset versions that are included in the product. The mainline is a sequence of product versions developed from an original baseline. Baselines may be specified using a configuration language, which can be used to specify what assets are included in a specific version of a particular product. It is possible to explicitly refer to a specific asset version, e.g. (CA_A, 1.0), or simply to specify the asset identifier, e.g. (CA_A). The latter case means that the most recent version of the component should be used in the baseline. Baselines are important because it is often required to re-create a specific version of a complete product. For example, a product line may be instantiated so that there are individual product versions for different customers. You may have to re-create the version delivered to a specific customer if, for example, that customer reports bugs in their version of the product that have to be fixed.

Suppose that we have a repository containing different asset versions. If it is needed to deliver a specific release of a product to a customer, we can find out from VM the product version number related to this release number. Subsequently, VM could specify the core and variable assets, with their version numbers that are used in that product. This allows us to re-create a specific version of a complete product.

For example, suppose that in Figure 6 a new customer C_G requests the release 1.1 of product P_B. In this case, BM would find out which version of product P_B has been used in release 1.1 by inspecting the relation (P_B, 1.0) | (C_C, 1.1). Now BM knows that version 1.0 of P_B is used in release 1.1. At this point, in order to re-create the required product for C_G, BM requests VM for the core and variable assets and their versions that are used in version 1.0 of P_B. In this example, because of relations (P_B, 1.0)| (F_A,1.0) and (F_A, 1.0)| (CA, 1.0), VM informs BM that version 1.0 of the core asset CA is used in version 1.0 of P_B, and in this way BM could re-create product P_B and deliver it to customer C_G.

5. CASE STUDY: MOBILE PRODUCT LINE

In this section, we describe the use of the proposed CM model, suggested in the previous section, in an SPL case study concerned with a Mobile Phone Product Line (MPPL) [14] employed in Mobile Phone companies. We use the Feature diagram and PL implementation (an architecture diagram) corresponding to those proposed in [15]. Figure 7 presents the Feature Model of the MPPL.

As mentioned in section 4, a set of Features can define the scope of SPL. Different products may be created from different subsets of these Features. These subsets are often chosen to meet specific business

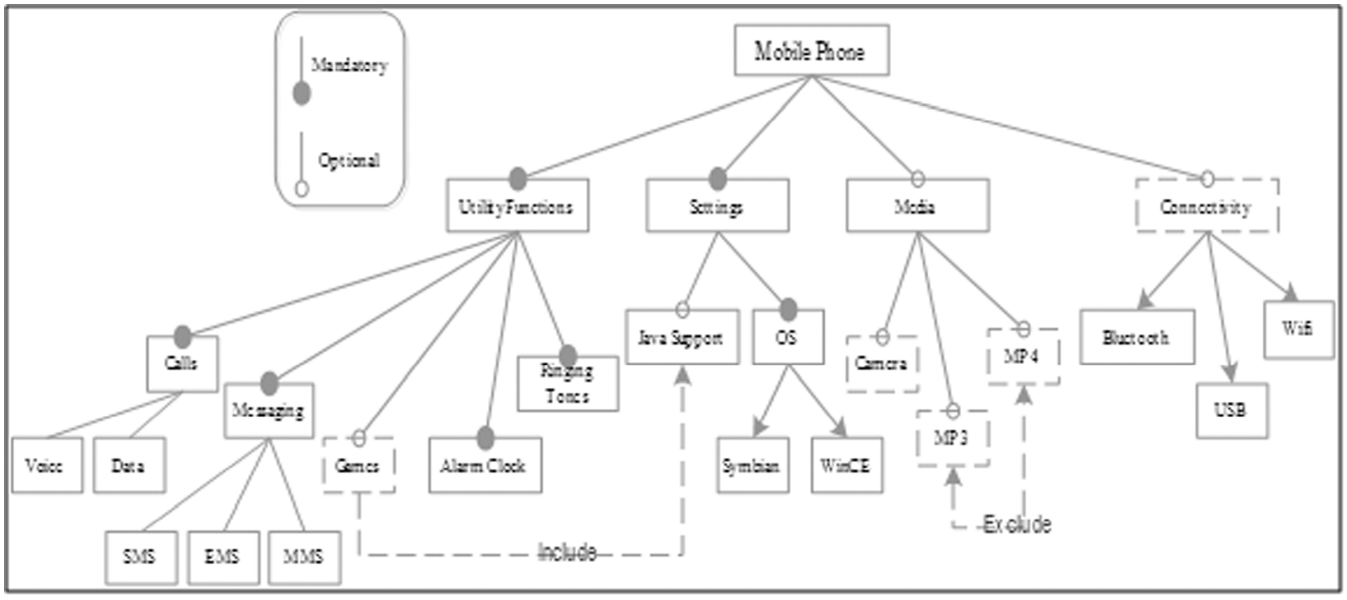

Figure 7: Feature Model of MPPL [9], [21].

requirements and form the PL specification of SPL. On the other hand, these Features need several Assets for their implementation. A specific implementation consists of a set of Assets. All the possible implementations (or, variants) in SPL constitute the product line implementation. Table 6 shows the implemented relation for MPPL Features and Assets. For example, to provide the Voice Feature for a product we must use {{keypad, Mice, Speaker}} Assets in it. According to Figure 7, we can specify the type of each Asset to be Core or Variable. The Assets related to a Mandatory Feature are Core Assets and those related to Optional Features are Variable Assets. In the following we illustrate our proposed models via two examples.

Example 1: According to Table 7, products P1 and P2 are created by selecting different subsets of Features from the Feature Model of MPPL. Messaging is a mandatory Feature that all products must support. When the change request 1 (see Table 8) is received from the users of product P1, Corrective Changes are applied to {SMSApps} Core Asset, and the version number of product P1 is increased (as a minor version change) but these changes must be applied to the SPL as well. So the version number of Core Asset related

Table 6
Implemented Relations for MPPL Features and Assets [14], [20].

Feature	*Assets*	*Feature*	*Assets*
Utility Functions	{{UtilInterface, UtilAdapter}}	JavaSupports	{{MIDP2.0}, {MIDP3.0}}
Calls	{{Transmitter, Receiver}}	OS	{{SymbianOSSetup}, {WinCEOSSetup}}
Voice	{{Keypad, Mice, Speaker}}	Symbian	{{SymbianOSSetup}}
Data	{{Keypad, GPRSApps}}	WinCE	{{WinCEOSSetup}}
Messaging	{{MessageInterface,MessageAdapter}}	Media	{{MediaInterface,MediaAdapter}}
SMS	{{SMSApps}}	Camera	{{VGA}, {MP1.3}, {MP2.0}, {MP3.2}, {MP5.0}}
EMS	{{EMSApps}}	MP3	{{MP3Apps}}
MMS	{{MMSApps}}	MP4	{{MP4Apps}}
Games	{{GamesInterface,GamesAdapter}}	Connectivity	{{ConnectivityInterface, Connectivity Adapter}}
AlarmClock	{{AlarmApps}}	Bluetooth	{{BluetoothHwd}}
RingingTones	{{RingTonesApps}}	USB	{{USBHwd}}
Settings	{{SettingInterface, SettingAdapter}}	Wifi	{{WifiHwd}}

Table 7
Sample Products of MPPL[9], [21].

Product	Features	Core and Variable Assets
Product 1	Utility Functions (Calls (Voice)), Messaging (SMS), Alarm Clock, Ringing Tones) Settings (OS (Win CE)) Media (Camera) Connectivity (Wifi, Bluetooth)	{{UtilInterface, UtilAdapter}},{{Transmitter, Receiver}},{{Keypad, Mice, Speaker}},{{Message Interface, Message Adapter}}, {{SMS Apps}}, {{MMS Apps}}, {{Alarm Apps}}, {{Ring Tones Apps}}{{Setting Interface, Setting Adapter}}, {{Win CEOS Setup}}{{Media Interface, Media Adapter}}, {{VGA}, {MP1.3}, {MP2.0}, {MP3.2}, {MP5.0}}{{Connectivity Interface, Connectivity Adapter}}, {{WifiHwd}, {BluetoothHwd }}
Product 2	Utility Functions (Calls (Voice), Messaging (SMS, MMS), Alarm Clock, Ringing Tones) Settings (OS (WinCE)) Media (Camera) Connectivity (Bluetooth)	{{UtilInterface, UtilAdapter}},{{Transmitter, Receiver}},{{Keypad, Mice,Speaker}},{{Message Interface, Message Adapter}},{{SMS Apps}}, {{Alarm Apps}}, {{Ring Tones Apps}}{{Setting Interface, Setting Adapter}}, {{Win CEOS Setup}}{{MediaInterface, Media Adapter}}, {{VGA}, {MP1.3}, {MP2.0}, {MP3.2}, {MP5.0}}{{Connectivity Interface, Connectivity Adapter}},{{Bluetooth Hwd}}

to the Feature and Feature Model will be increased as well. Since the updated Asset is a Core Asset, the changes must be propagated to all affected products which in this case is only product P2. Also, the version number of P2 is increased as a minor version change. The evolution path for this example is shown in Figure 8 where the numbers in brackets denote the path for applying the changes to MPPL.

(1) Two products P1 and P2 are created from the MPPL SPL (the selected Features in each one are shown in Table 7).

(2) One Feature of product P1 is updated due to a change to the related Asset.

(3) The changed Asset is a Core Asset, so the changes made to it must be reflected in SPL as well.

(4) The changes to the SPL are Corrective, so they must be applied to Product P2 as well. This causes P2 to be re created from the MPPL SPL.

Example 2: According to Table 7, products P1 and P2 are created by selecting different subsets of Features from the Feature Model of MPPL. The Camera Feature is an optional Feature so it is not mandatory for a product to support it. When the change request 2 (see Table 9) is received from the users of P2,

Table 8
Change Request1 to Product P1[9] .

Problem Report	
ID	CA_001
Product ID	P1
Problem	When an SMS is received at the same time as a phone call, the alarm will not play
Suggested Solution	Apply Corrective Change to SMS Feature (SMSApps Core Asset)

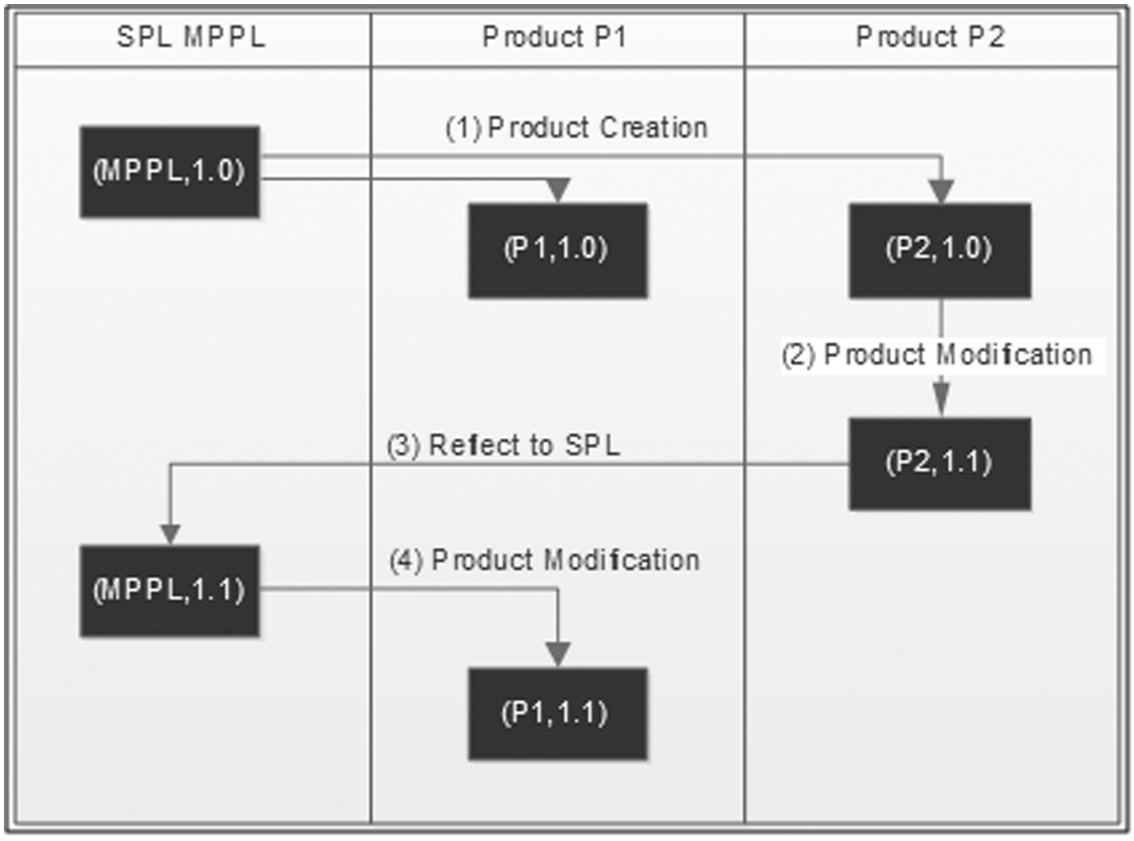

Figure 8 : Product Change (Corrective Change to Core Asset) in MPPL [9], [21].

Table 9
Change Request 2 for Product P2[9], [21].

Problem Report	
ID	CA_002
Product ID	P2
Problem	Camera Picture Quality is not ok.
Suggested Solution	Apply Effective Change to Camera Feature (MP5.0 Variable Asset)

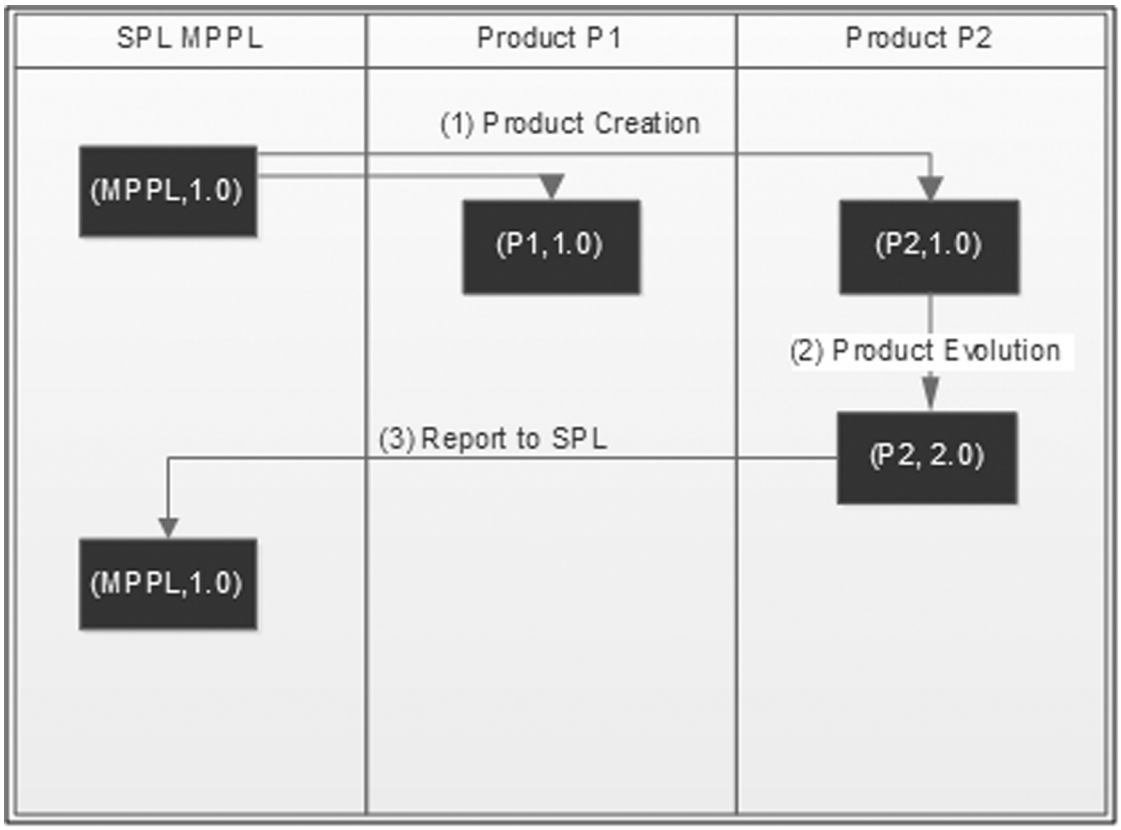

Figure 9: Product Evolution (Effective Change to Variable Asset) in MPPL.

Effective Changes are applied to the Variable Asset {MP5.0}. Consequently, the product goes through a major version change and its version number is increased. Furthermore, the applied changes must be reported to SPL and the version numbers of Variable Asset, related Feature and the Feature Model will be increased. Note that the updated Asset is a Variable Asset and the change is an Effective Change, so although product P1 contains the Camera Feature, the changes should not be applied to P1. Therefore, the version number of P1 remains unchanged. The evolution path for this example is shown in Figure 9.

(1) The products P1 and P2 are created from the MPPL SPL (the selected Features of each product are shown in Table 7)

(2) One Feature of P2 is updated by changing a related Asset.

(3) The change is an Effective Change to a Variable Asset, so it must be reported to SPL, but it is not required to propagate it to other products such as P1.

Example 3: According to Table 7, products P1 and P2 are created by selecting different subsets of features from the Feature Model of MPPL. We have shown a small part of relations between MPPL resources in Table 10. Suppose that customer C_B requests to receivethe release of product P1 again. In this case,

Table 10
A small part of relations between MPPL resources.

Relations between Feature Model and Feature	*Relations between Feature Model and Product*	*Relations between Feature and Assets*	*Relations between Products and Features*	*Relations between Products and Releases and Customers*
(MPPL, 1.0) | (Voice, 1.0)	(MPPL, 1.0) | (P1, 1.0)	(Voice, 1 .0)| (Keypad, 1.0)	(P1, 1.0) | (Voice, 1.0)	(P1, 1.0) | (C_A, 1.0)
(MPPL, 1.0) | (SMS, 1.0)	(MPPL, 1.1) | (P1, 1.1)	(Voice, 1.0)| (Speaker, 1.0)	(P1, 1.1) | (Voice, 1.1)	(P1, 1.0) | (C_B, 1.0)
(MPPL, 1.0) | (Wifi, 1.0)	(MPPL, 1.0) | (P2, 1.0)	(Voice, 1.1)| (Keypad, 1.0)	(P1, 1.0) | (SMS, 1.0)	(P1, 1.1) | (C_A, 1.1)
(MPPL, 1.0) | (Bluetooth, 1.0)	(MPPL, 1.1) | (P2, 1.1)	(Voice, 1.1)| (Speaker, 1.1)	(P1, 1.1) | (SMS, 1.1)	(P2, 1.0) | (C_C, 1.0)
(MPPL, 1.1) | (Voice, 1.1)		(SMS, 1.0)| (SMSApps, 1.0)	(P1, 1.0) | (Wifi, 1.0)	(P2. 1.0) | (C_D, 1.0)
(MPPL, 1.1) | (SMS, 1.1)		(SMS, 1.1)| (SMSApps, 1.1)	(P1, 1.1) | (Wifi, 1.0)	(P2, 1.1) | (C_C, 1.1)
(MPPL, 1.1) | (Wifi, 1.0)		(Wifi, 1.0)| (WifiHwd, 1.0)	(P1, 1.0) | (Bluetooth, 1.0)	(P1, 1.1) | (C_E, 1.1)
(MPPL, 1.1) | (Bluetooth, 1.0)		(Bluetooth, 1.0)| (BluetoothHwd, 1.0)	(P1, 1.1) | (Bluetooth, 1.0)	
(MPPL, 1.0) | (Camera, 1.0)		(Camera, 1.0)| (VGA, 1.0)	(P2, 1.0) | (Voice , 1.0)	
(MPPL, 1.1) | (Camera, 1.0)		(Camera, 1.0)| (MP1.3, 1.0)	(P2, 1.1) | (Voice, 1.1)	
		(Camera, 1.0)| (MP2.0, 1.0)	(p2, 1.0) | (SMS, 1.0)	
		(Camera, 1.0)| (MP3.3, 1.0)	(P2, 1.1) | (SMS, 1.1)	
		(Camera, 1.0)| (MP5.0, 1.0)	(p2, 1.0) | (Camera, 1.0)	
			(p2, 1.0) | (Camera, 1.0)	

BM could simply refer to the resource relation table in VM (Table 10) and find that C_B had release 1.0 of P1 in the past. BM could also distinguish between the core and variable assets used in P1 from the selected relation in Table 10 (shown with darker color), and build and deliver a copy of release 1.0 of P1 to C_B.

6. CONCLUSION

In this paper we completed our previous ChM and VM models and also proposed two new models for RM and BM to cover four main activities of CM for SPLs in a comprehensive CM model. The proposed models help to manage artifacts involved in the evolution of SPL and its products. Also, we illustrated the application of our proposed models to the MPPL SPL as a case study. The proposed models have a number of advantages as follows:

(1) They can handle different types of elements that can exist in an SPL such as Core/Variable Assets, Products, Features, Feature Models and Releases.

(2) The proposed ChM model covers two types of change, Corrective and Effective, and use different change propagation strategies to deal with them. For example, if Corrective changes are made to a product, it is important that these changes are reflected back to the Core or Variable Assets in SPL. This is because these

changes may be associated with a residual fault in Core or Variable Assets. Therefore, it is required to modify the corresponding Assets and update all related products. However, if Effective Changes are made to a product, it is not necessary to modify the corresponding Assets in SPL and update the related products because the Effective Changes made to one product may be not appropriate for others.

(3) The proposed models are provided for evolutionary-based SPL system development and maintenance in two phases of SPL Framework: Domain Engineering and Application Engineering. The models can deal with the distinct change propagation schemes used in each phase.

(4) The proposed VM model support VM for all SPL elements in diverse software evolution scenarios and maintain the relation between different elements of SPL. This reduces the effort required for resource management.

(5) The proposed VM model introduces two types of version change: Minor Version Change and Major Version Change. The former is used in version change after every Corrective Change; the latter is employed in version change after every Effective Change to SPL elements and products.

(6) The proposed RM model introduces two types of release: Minor Release and Major Release. The former is used in release change after every minor version change; the latter is employed in release change after every major version Change in SPL products.

(7) The proposed RM model simultaneously supports the relationship between products and releases and also the relationship between releases and customers.

(8) The proposed BM model could simplify the process of re-creating a version of a product for a specific existing customer or a new customer.

In the future we intend to consider other subsidiary activities of CM, namely, parallel development, distributed engineering, workspace management, configuration status accounting, configuration audits and process management in SPL, using the same approach as developed for the proposed ChM, VM, RM and BM models in this paper.

REFERENCES

[1] I. Sommerville, *Software Engineering*. Pearson, 2011.

[2] A. E. Kelly, "Design-Based Research in Engineering Education," in *Cambridge Handbook of Engineering Education Research*, A. Johri and B. M. Olds, Eds. Cambridge University Press, 2014, pp. 497–518.

[3] R. L. Nord, *Software Product Lines: Third International Conference, SPLC 2004, Boston, MA, USA, August 30-September 2, 2004, Proceedings*. Springer Berlin Heidelberg, 2004.

[4] K. Pohl, G. Böckle, and F. J. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer Berlin Heidelberg, 2005.

[5] I. Crnkovic, U. Asklund, and A. P. Dahlqvist, *Implementing and Integrating Product Data Management and Software Configuration Management*. Artech House, 2003.

[6] J. D. Campbell, A. K. S. Jardine, and J. McGlynn, *Asset Management Excellence: Optimizing Equipment Life-Cycle Decisions, Second Edition*. CRC Press, 2010.

[7] H. Mouratidis, *Software Engineering for Secure Systems: Industrial and Research Perspectives: Industrial and Research Perspectives*. Information Science Reference, 2010.

[8] B. B. Chua, "Rework Requirement Changes in Software Maintenance," *Software Engineering Advances (ICSEA), 2010 Fifth International Conference on*. pp. 252–258, 2010.

[9] *Software Product Lines: Practices and Patterns*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

[10] R. Ommering, "Software Configuration Management: ICSE Workshops SCM 2001 and SCM 2003, Toronto, Canada, May 14—15, 2001 and Portland, OR, USA, May 9—10, 2003. Selected Papers," B. Westfechtel and A. Hoek, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 16–23.

[11] K. L. S. Soujanya and A. A. Rao, "A Systematic Approach for Configuration Management in Software Product Lines," vol. I, 2015.

[12] A. Deursen, M. Jonge, and T. Kuipers, "Software Product Lines: Second International Conference, SPLC 2 San Diego, CA, USA, August 19—22, 2002 Proceedings," G. J. Chastek, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 217–234.

[13] S. A. Ajila and A. B. Kaba, "Using traceability mechanisms to support software product line evolution," *Information Reuse and Integration, 2004. IRI 2004. Proceedings of the 2004 IEEE International Conference on* . pp. 157–162, 2004.

[14] S. Ajila, "Change Management: Modelling Software Product Lines Evolution," *Proceeding 6 th World Multi-conference Syst. Cybern. Informatics*, vol. 7, pp. 492–497, 2002.

[15] L. J. M. Taborda, "The Release Matrix for Component-Based Software," *Matrix*, 2004.

[16] C. Elsner, D. Lohmann, and W. Schröder-Preikschat, "Product derivation for solution-driven product line engineering," *Proc. First Int. Work. Featur. Softw. Dev. (FOSD '09)*, p. 35, 2009.

[17] S. V. Shrivastava and H. Date, "Distributed Agile Software Development/ :," *J. Comput. Sci. Eng.*, vol. 1, no. 1, pp. 10–17, 2010.

[18] E. Kroon, "Layered configuration management for software product lines," p. 95, 2009.

[19] D. Romero, S. Urli, C. Quinton, M. Blay-Fornarino, P. Collet, L. Duchien, and S. Mosser, "SPLEMMA: A Generic Framework for Controlled-evolution of Software Product Lines," in *Proceedings of the 17th International Software Product Line Conference Co-located Workshops*, 2013, pp. 59–66.

[20] C. Thao, "A configuration management system for software product lines," no. August, 2012.