Chapter 3

CONFIGURATION MANAGEMENT OF SPL

SPL has been around as an approach that can help reuse software components instead of reinventing the wheel. However, advantages of SPL come only through disciplined software engineering principles coupled with good configuration management. Configuration Management (CM) involves the management of configurations of artifacts being used in product derivation in a given SPL. In fact CM keeps track of changes made to products and artifacts. Different versions of components and their features are understood from the CM. However, the CM for SPL is still in its infancy. Stated differently it is not sufficiently matured to be a full-fledged approach for CM of SPL. Since customer requirements are dynamic in nature addressing the change propagation and change control are essential tasks of configuration management. In this chapter a Generic Frame Work is proposed for addressing configuration management aspect and Quality product derivation in product line software systems and a CM model is discussed which supports twenty-three cases of change propagation. These cases throw light into different aspects of change propagation. Often the change propagation includes forward or rearward change propagation. For efficient handling of change propagation, the changes are categorised into major, minor and micro changes. The categories are based on the guidelines required by backward change and forward change propagation. In this chapter also an Automated Change Promulgation (ACP) algorithm is proposed and implemented. The empirical results revealed that there is significant improvement in the performance of CM when the proposed approach is used.

3.1 EVOLUTION OF SPL

In the last few decades, software engineering has undergone many changes. The software is required to be developed faster and there are a number of versions of software coming up in a rapid succession. In this context, configuration management plays a crucial role in the success of any product line software system. Therefore it is indispensable to have a sound configuration management system to have longevity of
software in the real world. The proposed framework in this paper provides an effective means to have such configuration management for SPL. Moreover, the framework has placeholders for optimizing configuration management and product derivation.

The organizations achieved high-quality products with time to market feature besides reduction in the cost of production. This is achieved by using the concept of software product lines[13]. The product line is a well-known phenomenon in the hardware industry which is adapted to software industry too. As the products in software have much similarity the idea of product line which represents a family of products with certain differences and commonality. Commonality and variability are the two terms to be well understood in SPL. The former refers to the features that are in common in many products while the latter indicates that there are certain features which are common to all products. Deriving a new product is based on the identification of the variability points. Thus variability plays a vital role in new product derivation.

SPLs make use of four activities that are indispensable to be successful. They are known as Core Asset Development (CAD), Custom Asset Development (CuAD), Product Development (PD) and Management of all the three (core assets, custom assets, products). The aim of CAD is to develop building blocks for production and such building blocks are essential or mandatory. The CuAD aims at building reusable building blocks that are specific and usage in all products are not mandatory. They are used to customize products and based on the variability. PD aims at producing new products by reusing existing artifacts and building new ones if required. This activity is made productive in SPL by reusing components. Finally, the management is the activity which is very important to coordinate efforts and take care of versions, their identification, change control, budgets, commitment, deliveries and so on. Project management cannot be separated from the SPL development. As SPL goes on over a time period, there should be continuous project management activities in terms of planning, executing, optimizing and improving processes involved. The project manager is the role which is to ensure that the time and budget constraints are met while developing an SPL.
Domain Engineering and Application Engineering

SPL configuration management is bound to accommodate components with similarity and variability. Artifacts based on different components and services are configured as part of SPL, should satisfy business goals and application domain compatibilities. Nevertheless, all products need to have some core assets that can be reused. This phenomenon makes a world of difference between the SPL approach and the traditional approach to software product development. The product domains come under problem space while the solution space constitutes product lines. Some examples are shown below

- Philips Digital TVs is the product line in consumer electronics domain
- Boeing 747 Family is the product line in Avionics domain
- GNU Compiler Suite is the product line in Compilers

In this chapter Automated Academic Regulations (AAR) product line is used for demonstrating configuration management. This SPL is in an education domain wherein it is a product line which is for various academic regulations. As and when the syllabus is to be revised to suit the industry needs there shall be a Board of Studies (BOS) meeting to finalize upon the syllabus for the current academic year. In this process the syllabus is updated in the existing subjects and also new subjects are added to the regulations. In the Choice Based Credit System (CBCS) students are allowed to choose their choice subjects available in the regulations for attaining the credits required for the core subjects and optional subjects. As and when subjects are revised each revision is given a different version number based on the revision. Students have a choice of selecting any version of the subjects either from core or optional. The reusability found in the traditional product development model is confined to new program construction. However, SPL exploits the reusability more. The reusability can be envisaged either horizontally or vertically with variant and adaptive designs. This approach can reduce risk, time and development cost besides improving the quality.

The traditional engineering model is different from that of SPL engineering. The former is characterized by domains, applications, and implementations in the
individual plane. The individual implementations are assets here. In the latter, assets are not individual implementations. Instead, they are components that are reused in the architectural framework of SPL. In a single common product, as explored by Krueger et al. [7] the domain engineering activities are mapped to application engineering phases as shown in Table 3.1.

Table 3.1: Mapping Domain and Application Engineering Phases

<table>
<thead>
<tr>
<th>DOMAIN ENGINEERING PHASE</th>
<th>APPLICATION ENGINEERING PHASE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain analysis</td>
<td>Requirements analysis</td>
</tr>
<tr>
<td>Domain design</td>
<td>Application design</td>
</tr>
<tr>
<td>Domain implementation</td>
<td>Application implementation</td>
</tr>
</tbody>
</table>

3.2 CHANGE PROPAGATION

There is another aspect essential to SPL. It is known as change management which is achieved through configuration management activities besides others. It is responsible for keeping track of changes being made in the real-time production environment. As there are constant changes in requirements and production of artifacts, it is inevitable to have a sophisticated configuration management for SPL [120]. There are many reasons for software to be subjected to change. They include new functionalities, platform changes, corrective changes, bug fixes, platform changes, and changes initiated for quality purposes and so on.

Software Configuration Management (SCM) involves various techniques and disciplines that are related to initiation, evaluation and controlling of changes made to SPL at different levels during the development and also after development based on changes incorporated into the SPL for improving quality and to fulfill changing needs of customers from time to time. In the software production environment, it emphasizes the significance of controlling changes so as to maintain different versions and features associated with artifacts and product in SPL. When compared to conventional software development, SPL faces different issues. In the case of
conventional software development, there is time dimension in the product evolution as explored in [121]. In the case of SPL on the other hand, the evolution is in two dimensions namely space and time. Both products and artifacts have their line of production interestingly. The evolution of artifacts takes place in time dimension while the evolution of the products takes place in the space dimension.

Software Configuration Management (SCM) controls the evolution of software products by using its well-established procedures and functions [122]. The focus of this chapter is on SCM for SPL. There are many challenges faced by software development environments. The challenges are in terms of complexity, changed requirements, code duplication, ambiguity in requirements, interoperability conflicts and so on. Across software development companies, these are common issues. The software industry found a solution to many of the issues in the form of software product line. The development of product lines instead of individual products has led to dramatic changes in the perceptions of quality product development. A family of products concept in the name of SPL promotes productivity, increases reusability and reduces time and effort required for development [121]. It can be considered as a paradigm shift in the way software product development. The novel paradigm bestows many advantages such as building common solutions, reusing components and building customized products [122].

**Product Line Architecture**

Towards the SPL development, an approach known as Product Line Architecture (PLA) came into existence. PLA exploits common functionalities and differences or variation points in a better way. PLA identifies common features required by many products. They are known as core assets. It also identifies certain features which are specific to some products. They are known as custom assets.

The software industry in the real world learned lessons from SPL and adopted industry standards in SPL development and configuration management. The component-based approach is reinforced in SPL. Software components or reusable assets that are well defined with industry standard interfaces promote reusability and avoid repetition of work. It affects budgets and timelines of SPL positively. Advantages of SPL can be realized when PLA is used to map artifacts subjected to
configuration management in the confines of PLA. When new products are derived, it involves the selection of core and custom assets based on the requirements of customers. Thus, derived products meet business goals of the organization besides meeting the genuine needs of customers. Thus the product line has gained significant importance in software development industry.

3.3 STATE-OF-THE-ART

In the case of generic SCM model proposed by Clements and Northrop [96], the configuration management encompasses product instances, core and custom assets. With respect to product instance in SPL, there is a product in use in the wake of configuration management of SPL. Product derivation is explored by Van Gurp [7] by proposing a Coupling Variation Modelling (CVM) tools that work in tandem with subversion. In another development of SCM, Kruger [7] proposed a method that can be used to exploit conventional SCM tools for SPL. The infrastructure pertaining to core assets and product derivation are brought to the confines of SCM. Kruger and Van Duresen [118] and Van Ommering [8] focused on automatic forward propagation. As changes occur in artifacts, products reflect those changes. They manually maintained dependencies among components being used in the SPL. Towards SPL configuration management Molhodo SPL [123] came into existence. It is a prototype that supports configuration management and considers evolution at the configuration level instead of programming level in terms of source code. Here, changes can be propagated from core assets or custom assets to products as well. It may be the other way round from products to different assets in SPL.

In this chapter, a systematic configuration management is adopted and change propagation is explored and the proposed method facilitates product specific changes to the components that are shared. It happens without affecting the preferred components. Support of product specific changes to underlying core and custom assets is very important. However, interference between changes to artifacts, changes to products is needed. In order to achieve these custom assets can have product specific branch that can help incorporate changes. The core and custom assets have a specific branch to support changes to the product.
ACP algorithm is proposed in order to support SCM in SPL. The algorithm contains versioning model for products and artifacts with different variables like x, y and z. These variables represent major, minor and micro changes respectively. Every artifact is assigned a major version, minor version and micro version. Major version indicates a substantial change in the product, minor change indicates the platform change and micro change indicates the bug fixes that have been incorporated after building major version. In the evolution of product line there might be trivial changes. These changes may not be recorded in order to avoid unnecessary usage of valuable resources. In order to support these features guidelines are incorporated into ACP algorithm. The guidelines in the ACP algorithm are related to handling different kinds of changes. Towards this end, it can handle plenty of cases in which change can be propagated. Therefore change propagation is a nontrivial task which is carefully handled using ACP algorithm which invokes subroutines in order to realize different kinds of change to be propagated as per the business rules or guidelines.

3.4 PROPOSED FRAMEWORK FOR CONFIGURATION MANAGEMENT AND PRODUCT DERIVATION OF SPL

SPL is a dynamic phenomenon where changes are to be made frequently. Many kinds of changes are possible. Those changes are to be envisaged and properly implemented in configuration management of SPL. Change management, in general, is a very important aspect that has a set of activities. In the case of SPL, the change propagation and corresponding configuration management are quite different. This section provides information on this in some detail.

For this reason, it is important to have modularity and variation points in order to support future requirements. This is possible to reuse baseline architecture pertaining to SPL. Moreover, product line should support best cases that are economically feasible. It involves attributes pertaining to quality while deriving new products and change management is underway. While implementing change in a system it is essential to ensure consistency, correctness and completeness as explored by McGregor [96]. It is essential to know when changes take place to the product line. Since change management is an integral part of product lines, the know-how on factors that influence change is very important. The factors are described here.
- A new variant might force an existing variation point to be changed and thus the product domain.
- When products are added to a product line, it could influence changes in variation point.
- Sometimes a variation point can be split into two that causes adding new assets or modify existing assets.
- Constant evolution to accommodate new needs of customers can drive changes to the product line.
- Even it is possible that when many variations are involved, it is possible to split a product line into multiple product lines.

Configuration management has been available for traditional products and in fact, its maturity level is adequate. With respect to SPL, the configuration management is still in its infancy and needs further research and development to cope with the demands in the real world. Some approaches in the literature are tailor made and not sufficient to have a general solution. In other words, there is a need for a comprehensive framework that can guide configuration management and quality product derivation. This is the motivation behind the research in this thesis. In this chapter, a generic framework is proposed. It guides SPL configuration and new product derivation in a generic way. The framework is modular and extendable. It has provision for improving its flexibility by using ontology and variability-aware design patterns. An overview of the proposed generic framework is shown in Figure 3.1.

The proposed framework is generic in nature. It does mean that it can be adapted to any domain. It helps in bringing about a systematic approach to SPL configuration management and new product derivation with industry best practices. The framework has provisions for SPL configuration management, product derivation, variability-aware design patterns and ontology for optimizing configurations and product derivation. In this chapter, AAR product line is taken as case study to apply the framework and its underlying algorithms. AAR refers to Automated Academic Regulations which is a software developed for Academia, for smooth information flow and accessibility to data, enabling quick decisions. It is proprietary software developed for an academic Institution. The source of the
software is from Software Solutions Pvt Ltd Hyderabad. However, only the realization of one part of the framework that is configuration management is focused in this chapter. The other parts are product derivation, variability aware design patterns and ontology construction.

Product derivation is needed in order to have an automatic approach to consider different artifacts that meet the requirements of a new product (based on customers' genuine needs). The framework has a placeholder for the derivation of a new product. The functionality of this part includes taking quality parameters and features from the user and produce different products with permutation and combinations of artifacts. Once the permutations are produced, weights associated with artifacts and the critical path analysis is used to derive a high-quality product. The variability aware design patterns are another placeholder in the proposed framework.

Figure 3.1: Overview of the proposed framework
This part focuses on the identification of variability in the product line and identifies a design pattern or set of design patterns in order to have an industry standard approach for solving a recurring problem. The usage of a composite design pattern (combination of two or more existing design patterns) that is variability-aware design patterns are explored for SPL configuration management and product derivation.

Yet another placeholder in the framework is ontology construction. This is pertaining to the usage of ontology to represent feature models. When feature models are represented as ontology, it is easier to navigate among features of different kinds. Moreover, ontology can provide a programmable interface which can automate the reconfiguration of SPL. It also can optimize the product derivation. More details on product derivation, the usage of variability-aware design patterns and domain ontology construction for automatic reconfiguration of SPL and quality product derivation are deferred to the ensuing chapters. Section 3.5 focuses on the first part of the framework that is a systematic approach for configuration management.

3.5 SYSTEMATIC APPROACH FOR CONFIGURATION MANAGEMENT

In the existing system both artifacts and products are involved in the configuration management. Core assets exhibit application product independent and domain specific features that can be reused in different products. Whereas custom assets exhibit product-specific features that are reflected in that product only instead of all products derived in SPL. In fact, a product is the combination of assets of those two kinds. The SPL takes care of producing a new product by combining core and custom assets as required. Thus any product in the product line can share core assets essentially and custom assets as per the needs of customers. It can be concluded here that a product can have two possible parts in that. They are core and custom parts that are made up of core and custom assets as discussed.

An important aspect which is explored in this chapter is configuration Identification and change propagation. Change propagation is the process of propagating changes from one component to another component. Possible change
propagations are as follows.

1. Core assets $\rightarrow$ Products
2. Custom assets $\rightarrow$ Products
3. Products $\rightarrow$ Core assets
4. Products $\rightarrow$ Custom assets

An asset which is reused in many products is also known as a public asset. When changes are propagated from assets to the product, it is known as forward amend promulgation. One of the examples for this is a change in the public asset being used in a product with required changes in the related core and custom assets. On the other hand, backward amend promulgation refers to the change propagation from product to assets. An example for this is the possible propagation of a corrective change in the public asset of a product which triggers changes in assets of different products in order to incorporate such public asset changes.

There are different possibilities with respect to change propagation. The same is presented in Table 3.1. The table shows different notations to represent core and custom assets. The notations are as shown below.

- IA represents core assets
- DA represents custom assets
- P represents product instances
- IA* represents changes made to the core asset IA
- DA’ represents the changes made to the custom asset DA
- P^ represents the changes made to the product P
- P*^ represents the merged results of all changes made to IA, DA, and P
- P*^ represents the merged results of changes of IA and P
- P’^ represents the merged results of changes of DA and P
- P* represents the merged results of changes of IA and DA

As shown in Table 3.2, the cases from 1 to 14 reflect forward promulgation while the cases from 15 to 23 reflect the other kind of propagation known as backward change promulgation.
There are 23 cases presented. Each case has a specific meaning. All cases are described here.

**Possible Cases**

**Case 1**
- The product is sharing the core asset, IA and custom asset, DA.
- Changes have been made to DA, in this case, changes made in the custom assets project are brought to the shared asset in the product.
- An example of this case is a correction made to an asset in the custom asset projects which is useful to the product sharing the asset. Here sharing means using the asset.
- Thus changes are pushed to the product

**Case 2**
- The product is sharing the core asset, IA and custom asset, DA.
- Changes have been made to IA the core asset. In this case, changes made in the core assets are reflected in the product P.
- An example of this case is a correction made to an asset in the core asset which is useful to the product.
- Thus the changes are pushed to the product.

**Case 3**
- The product is sharing the core asset, IA and custom asset, DA.
- Changes have been made to IA and DA in the core and custom assets.
- In this case, the changes made in the core assets and custom assets are brought to the shared asset in the product.
An example of this case is when a correction is made to an asset in the core and custom asset which is appropriate to the product sharing the asset and thus the changes are pushed to the product.

Case 4

- The Product is sharing the asset IA from the core asset and DA from the custom asset.
- Changes have been made to the shared asset P in the product and the asset IA in the core asset.
- In this case, the changes from the asset IA in the core asset are merged with the shared asset P with the product specific changes.
- This case would represent a product’s independent evolution while bringing correction changes from the core.

Table 3.2: Different forms of change propagation

<table>
<thead>
<tr>
<th>CASE NO</th>
<th>BEFORE</th>
<th>AFTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IA D'A P</td>
<td>IA D'A P'</td>
</tr>
<tr>
<td>2</td>
<td>IA* DA P</td>
<td>IA* DA P*</td>
</tr>
<tr>
<td>3</td>
<td>IA* D'A P</td>
<td>IA* D'A P*</td>
</tr>
<tr>
<td>4</td>
<td>IA* DA P^</td>
<td>IA* DA P*^</td>
</tr>
<tr>
<td>5</td>
<td>IA D'A P^</td>
<td>IA D'A P^*</td>
</tr>
<tr>
<td>6</td>
<td>IA* D'A P^</td>
<td>IA* D'A P*^</td>
</tr>
<tr>
<td>7</td>
<td>IA* D'A P^</td>
<td>IA* D'A P*</td>
</tr>
<tr>
<td>8</td>
<td>IA D'A P</td>
<td>IA D'A P</td>
</tr>
<tr>
<td>9</td>
<td>IA* DA P</td>
<td>IA* DA P*</td>
</tr>
</tbody>
</table>
### Case 5

- The product is sharing the asset IA from the core asset and DA from the custom asset.

- Changes have been made to the asset P in the product and the asset DA in the custom asset, in this case, the changes from the asset DA in the custom is merged with P with the product specific changes.

- At this stage, asset P of the product has both sets of changes. This case would represent a product’s independent evolution while bringing correction changes from core assets.

<table>
<thead>
<tr>
<th></th>
<th>IA</th>
<th>DA</th>
<th>$</th>
<th>IA</th>
<th>DA</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>IA</td>
<td>$</td>
<td>P</td>
<td>IA</td>
<td>DA</td>
<td>P</td>
</tr>
<tr>
<td>12</td>
<td>$</td>
<td>DA</td>
<td>P</td>
<td>IA</td>
<td>DA</td>
<td>P</td>
</tr>
<tr>
<td>13</td>
<td>IA</td>
<td>DA</td>
<td>P^</td>
<td>IA</td>
<td>DA</td>
<td>P^</td>
</tr>
<tr>
<td>14</td>
<td>IA</td>
<td>DA</td>
<td>P^</td>
<td>IA</td>
<td>DA</td>
<td>P^</td>
</tr>
<tr>
<td>15</td>
<td>IA</td>
<td>DA'</td>
<td>P^</td>
<td>IA</td>
<td>DA'^</td>
<td>P^</td>
</tr>
<tr>
<td>16</td>
<td>IA*</td>
<td>DA</td>
<td>P^</td>
<td>IA*^</td>
<td>DA</td>
<td>P^</td>
</tr>
<tr>
<td>17</td>
<td>IA*</td>
<td>DA'</td>
<td>P^</td>
<td>IA*^</td>
<td>DA'^</td>
<td>P^</td>
</tr>
<tr>
<td>18</td>
<td>IA*</td>
<td>DA'</td>
<td>P^</td>
<td>IA</td>
<td>DA</td>
<td>P^</td>
</tr>
<tr>
<td>19</td>
<td>IA</td>
<td>DA'</td>
<td>P^</td>
<td>IA</td>
<td>DA</td>
<td>P^</td>
</tr>
<tr>
<td>20</td>
<td>IA*</td>
<td>DA</td>
<td>P^</td>
<td>IA</td>
<td>DA</td>
<td>P^</td>
</tr>
<tr>
<td>21</td>
<td>$</td>
<td>DA</td>
<td>P</td>
<td>IA</td>
<td>DA</td>
<td>P</td>
</tr>
<tr>
<td>22</td>
<td>IA</td>
<td>$</td>
<td>P</td>
<td>IA</td>
<td>DA</td>
<td>P</td>
</tr>
<tr>
<td>23</td>
<td>$</td>
<td>$</td>
<td>P</td>
<td>IA</td>
<td>DA</td>
<td>P</td>
</tr>
</tbody>
</table>
Case 6

- The product is sharing the asset IA from the core asset and DA from the custom asset.
- Changes have been made to the shared asset P in the product, the asset IA in the core asset and the asset DA in the custom asset, in this case, changes from the asset IA in the core and DA in the custom are merged with the asset P with the product specific changes, now, P of the product has all the set of changes.
- This case would represent a product’s independent evolution while bringing correction changes from the core and custom

Case 7

- Changes have been made to assets in the custom assets, core assets and the product.
- The developer wants to replace the modified asset in the product with modified assets in the core and custom assets. After the developer performs this action, the asset in the product will be identical to the one in the core and custom assets.
- In this case, the developer may find the product specific changes which may not be useful and could be subsequently not replaced with the changes made in the core and custom.

Case 8

- Changes have been made to assets in the custom assets and product.
- The developer wants to replace the modified asset in the product with modified assets in the custom assets. After the developer performs this action, the asset in the product will be identical to the one in the custom assets.
In this case, the developer may find the product specific changes which may not be useful and could be subsequently replaced with the changes made in the custom.

**Case 9**

- Changes have been made to the assets of the core assets and product.
- The developer wants to replace the modified asset in the product with modified assets in the core assets.
- After the developer performs this action, the asset in the product will be identical to the one in the core assets.
- In this case, the developer may find the product specific changes might not be useful and could be replaced with the changes made in the core asset.

**Case 10**

- At this stage, an asset from the core assets and custom assets, which had not been shared with the product, is now shared with the product.
- The asset may be needed to the product.

**Note:**

- $ indicates that no asset from the core assets or custom assets had been shared by the product before.
- After change it may be used in the product.

**Case 11**

- At this stage, an asset from the core assets that had not been used in the product is now used in the product. The asset may be needed to the product.

**Case 12**

- At this stage, an asset from the custom assets that had not been with the product is now with the product. The asset may be needed to the product.
Case 13
- The Product is using the asset IA from the core asset and DA from the custom asset.
- Changes have been made to the asset P in the product.
- In this case, the changes in the product asset P are changed with product specific changes. This case might represent a product’s independent evolution.

Case 14
- The Product is using the asset IA from the core asset and DA from the custom asset.
- Changes have been made to the asset P in the product.
- In this case, changes in product asset P are changed with product specific changes and these changes are reflected in core assets and custom assets.

Case 15
- The Product is using the asset IA from the core asset and DA’ from the modified custom asset.
- Changes have been made to the asset P in the product.
- In this case, changes in product asset P are changed with product specific changes and these changes are reflected in core assets and custom assets.

Case 16
- The Product is using the modified asset IA* from the core asset and DA from the custom asset.
- Changes have been made to the asset P in the product.
- In this case, changes in product asset P are changed with product specific changes and these changes are reflected in core assets and custom assets.
Case 17

- The Product is using the modified asset IA* from the core asset and modified DA’ from the custom asset.
- Changes have been made to the asset P in the product.
- In this case, changes in product asset P are changed with product specific changes and these changes are reflected in core assets and custom assets.

Case 18

- The Product is using the modified asset IA* from the core asset and modified DA’ from the custom asset.
- Changes have been made to the asset P in the product.
- In this case, changes in product asset P are changed with product specific changes and these changes are not reflected into existing core assets and custom assets but new core and custom assets are created with the said features.

Case 19

- The Product is using the modified asset IA from the core asset and modified DA’ from the custom asset.
- Changes have been made to the asset P in the product.
- In this case, changes in product asset P are changed with product specific changes and these changes are not reflected into existing core assets and custom assets but new core and custom assets are created with the said features.

Case 20

- The Product is using the modified asset IA* from the core asset and modified DA from the custom asset.
- Changes have been made to the asset P in the product.
• In this case, changes in product asset P are changed with product specific changes and these changes are not reflected into existing core assets and custom assets but new core and custom assets are created with the said features.

Case 21,22,23

• These cases show that when there are changes in the products, these changes are reflected back into the assets let it be core or custom assets as required in the application.
• These follow the guidelines specified for backward change propagation.

Summary of Cases

• The cases from one to fourteen are termed as forward change propagation
• The cases pertaining to backward change propagation are reflected in cases from ten to twenty-three.

Note: In the thesis Propagation and Promulgation are used interchangeably.

Forward Change Promulgation

Core assets and custom assets are created or changed according to the needs of a customer to have a new product in the product line. In such case artifacts with micro and minor changes are propagated to the existing product. In the case of a major change, the existing product is not subjected to those changes, but the new features are stored as a separate asset in the corresponding databases.

Backward Change Promulgation

With respect to configuration management of SPLs the changes to the products are also considered as they are subjected to dynamic changes based on needs of customers or due to quality maintenance activity. When a major change in the product is done, this is not simply updated but implemented and stored as a new artifact which may be either core of custom assets based on the nature of major
change. In the case of minor or micro changes, the same is reflected in the corresponding assets found in the database.

3.5.1 Recommendations for Change Propagation

The proposed approach facilitates the product specific changes made to shared components in the product without interfering with the change that has been made to the referred components of artifacts. The core and custom assets may have product specific branch in order to avoid interference. These aspects were discussed earlier in this chapter. Directly the guidelines required for change propagation in product lines are provided here.

1. In forward change promulgation minor and micro changes are updated in the existing product of the product line.
2. When there are major changes in the core assets or custom assets, they are not updated in the existing product but created as new assets in the corresponding core and custom asset databases respectively, based on ACP algorithm which is presented in section 3.6.
3. In backward change promulgation minor and micro changes are updated to the existing core assets or custom assets database.
4. When there are major changes in the product, these changes are not updated in the corresponding core assets or custom assets database but created as a new asset in the corresponding core and custom assets database.

The versioning attributes aforementioned in this chapter such as x, y, and z are used to represent major, minor and micro changes respectively. These are used in the proposed algorithm named Automated Change Propagation (ACP). According to the convention specified, each asset is named as $A_{x,y,z}$ where the x, y, and z values are incremented when changes are made corresponding to those attributes.

3.6 AUTOMATED CHANGE PROPAGATION (ACP) ALGORITHM

In fact, the proposed change promulgation approach with ACP algorithm has, in turn, many algorithms to accommodate forward change promulgation, core asset
configuration, the creation of new core assets, changing core assets, configuration of custom assets, creating new custom assets, and backward promulgation.

As shown in the algorithms there are different cases to be handled. The ACP approach is capable of accommodating the dynamics of change propagation with both forward and backward promulgation.

//Algorithm for Forward change promulgation/

Create Assets()

1. {
2.   CoreAssets()
3.   {
4.     While(Completion of product)
5.       {
6.         If(require Core Assets in Database)
7.           Then
8.             Copy them into product
9.           Else
10.          NCA = createCoreAssetsConfig()
11.         Copy NCA into new product version and Core Assets Database
12.       }
13.   }
14.  CustomAssets()
15.  {
16.    While(Completion of product)
17.    {
18.      If(required configuration Custom Assets are in Database)
19.        Then
20.         Copy them into product
21.        Else
22.         NCA = createCustomAssetsConfig()
23.       Copy NCA into new product version and Core Assets Database
24.    }
25.  }
26.}
// Core Assets/
1. Create CoreAssetsConfig()
2. {
3. If (required core assets does not exist in the database)
4. Then
5. NCA = newCoreAssets()
6. Return NCA
7. Else
8. MCA = modifyCoreAssets()
9. Return MCA
10. }

// Creating new core Asset //
1. newCoreAssets()
2. {
3. Read CoreAsset with 3 attributes named as X,Y,Z;
4. Store in CoreAssets database;
5. Return CoreAsset;
6. }

//Modifying Core Assets//
1. ModifyCoreAsset()
2. {
3. Read existing coreasset with 3 attributes X,Y,Z
4. If (Changes are there in X)
5. {
6. Then Update Core Assets configuration and product with major version
7. Exit;
8. }
9. If (Changes are there in Y)
10. Then return MCA(Modified Core Asset) with minor changes
11. If (Changes are there in Z)
12. Then return MCA(Modified Core Asset) with macro changes
13. }

//Create Custom Assets/

1. Create CustomAssetsConfig()
2. {
3. If (required custom assets does not exist in the database)
4. Then
5. NCA = newCustomAssets()
6. Return NCA
7. Else
8. MCA = modifyCustomAssets()
9. Return MCA
10. }

// Create new Custom Assets/

1. newCustomAssets()
2. {
3. Read CustomAsset with 3 attributes named as X,Y,Z;
4. Store in CustomAssets database;
5. Return CustomAsset;
6. }
//Modify Custom Assets//
1. ModifyCustomAsset()
2. {
3. Read existing customasset with 3 attributes X,Y,Z
4. If (Changes are there in X)
5. {
6. Then Update Custom Assets configuration and product with major version
7. Exit;
8. }
9. If (Changes are there in Y)
10. Then return MCA(Modified Custom Asset) with minor changes
11. If (Changes are there in Z)
12. Then return MCA(Modified Custom Asset) with macro changes
13. }

//Algorithm for Backward promulgation//
1. Maintenance ()
2. {
3. Read product version with attributes namely X,Y,Z
4. If(updates are required in X)
5. {
6. Then
7. If(changes are required in coreAsset)
8. {
9. Then
10. Create CoreAsset();
11. Else
12. Create CustomAsset();
13. }
14. Update product;
15. }
16. If(updates are required in Y & Z)
17. Then
18. Update product, core asset and custom asset;
19. }

3.6.1 Walkthrough of Product Line Example AAR

The following product line example demonstrates the proposed approach which includes change propagation and product line evolution. In Figure 3.2, the core asset development and configuration is visualized. It is evident that the changes made to assets are considered as major, minor and micro with $A_{x,y,z}$ notation. Accordingly the core assets A, B and C are represented as $A_{1,1,1}$, $B_{1,1,1}$ and $C_{1,1,1}$. When A is subjected to major change it is identified as $A_{2,1,1}$. In the same fashion, if there is a minor change in the asset B, it is represented as $B_{1,2,1}$. When there is a micro change in the asset C, it is represented as $C_{1,1,2}$.

![Figure 3.2: Version trees of core assets](image)

As shown in Figure 3.2, the change made to assets A, B and C are appropriately shown with different versions. The versioning discussed a little earlier in this section provide information on the assets and their level of changes. The $x,y,z$ notation reflects changes made to the asset with distinctly reflecting the kind of changes. Thus values of $x$, $y$, and $z$ play a vital role in CM of SPL.
As shown in Figure 3.3, custom assets represented as I, J, K with x, y, z versioning values are subjected to changes in the process of configuration management based on the corrective or adaptive changes made to the assets keeping the needs of customers in mind. The changes made to x, y, z attributes used in the versioning reflects the kind of change such as major or minor or micro.

As shown in Figure 3.4, the forward and backward propagation of changes is
illustrated. It has notations such as PR and PQ representing products, IA representing core assets and DA representing custom assets. A, B, and C are considered to be core assets while the I, J, and K are considered to be custom assets. A and B are the core assets and custom asset J are used in the product PR. Similarly, B and C core assets and I and k custom assets are part of the product PQ. From the version trees representing both the products shown in the Figure 3.4, the following information is inferred.

- The product PR is at version PR\textsubscript{1.1.1}, the core assets is at IA\textsubscript{1.1.1}, custom assets DA\textsubscript{1.1.1} and PQ is at version PQ\textsubscript{1.1.1} shows the products in the product line named PR and PQ with their independent evolutions.

- Product PR is formed by the combination of two core assets named A\textsubscript{1.1.1} and B\textsubscript{1.1.1} along with a custom asset J\textsubscript{1.1.1}. Because of a minor change in the core asset B\textsubscript{1.1.1} it is changed to B\textsubscript{1.2.1} this change is reflected in the project PR\textsubscript{1.2.1}.

- After following the change propagation guidelines mentioned in section 3.5.1, the forward change promulgation and backward change promulgation is depicted in Figure 3.4.

- In the evolution of custom assets asset K\textsubscript{1.1.1} has been modified to K\textsubscript{1.1.2} to reflect the micro change. This being a micro change is brought into the product as well.

- Since this asset is being used in the product PQ\textsubscript{1.1.1} the micro change has been updated in the products PQ\textsubscript{1.1.1}.

- Similarly, there has been a major change in the core asset C\textsubscript{1.1.1} which is depicted as C\textsubscript{2.1.1} which is stored as an asset in the core asset database. Adding a new version of the asset is separately shown the backward propagation.
3.7 RESULTS AND DISCUSSION

Automated Academic Regulations (AAR) is the product line used to demonstrate the usefulness of the proposed approach. In this SPL, a new product can be derived by choosing core and custom artifacts. Once a product is derived, the changes to the assets and products are supported with forward and backward change propagation approaches explored in this chapter. As discussed earlier the two cases are reiterated here before presenting results.

Keeping these change management dynamics in perspective, the ACP algorithm works accordingly. The algorithm can help in maintaining assets and products in SPL to have many advantages like quality, consistency and time to market products.

- When the core assets or custom assets are modified or created according to the requirements of a new product in the product line, then the artifacts core assets, custom assets with minor and micro change are promulgated to the existing product which is named as forward promulgation.

- If there is a major change from the existing functionality then the existing product is not updated with the changes. In the configuration management of product lines the products also can be changed or modified dynamically as per user requirements under maintenance activity.

- When there is a major change in the product of a product line then this change is not updated but stored as a new asset in corresponding core asset or custom asset database this is named as backward promulgation.

- When there is a minor or micro change i.e., either the change is done to fix the bug or any fixing other nonfunctional requirement it is updated in core asset or custom asset database.
As shown in Table 3.3, the results of performance comparison between existing Molhodo SPL algorithm[123] and proposed algorithm is presented. Here PR, PQ and PN are the products in product line automated academic regulations.

<table>
<thead>
<tr>
<th>PRODUCT LINE</th>
<th>TIME TAKEN IN MILLISECONDS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EXISTING</td>
</tr>
<tr>
<td>PR</td>
<td>20</td>
</tr>
<tr>
<td>PQ</td>
<td>25</td>
</tr>
<tr>
<td>PN</td>
<td>15</td>
</tr>
</tbody>
</table>

Where PR represents the product with core and custom subjects selected by a student opting for a course. Similarly, PQ and PN are the products opted by two different set of students. The approach is generic and can be applied to any curriculum. For testing, M.Tech Software Engineering core subjects are considered as core assets and elective subjects are considered as custom assets and the results obtained are as shown.

As shown in Figure 3.5, it is evident that three products such as PR, PQ and PN are in the product line of the case study SPL AAR. The time taken to derive products is observed and plotted in the graph. The performance of proposed ACP algorithm and an existing algorithm is compared. The existing algorithm takes twenty milliseconds to derive PR while the ACP takes fourteen milliseconds. In the same fashion there is significant performance improvement in the case of other products as well. It is evident that the results of ACP algorithm are compared with an existing algorithm. The results revealed that the time taken for different products in the product line is less with ACP algorithm when compared with that of an existing algorithm.
Figure 3.5: Graph showing the time taken to build a product in the product line

SUMMARY

With respect to configuration management in SPL, multiple evolving baselines for different assets are considered instead of many individual product baselines. This chapter covered the proposed approach that caters to the configuration management of SPL which consists of core assets, custom assets besides products that have been derived. The products, share core and custom assets as required. It is required to maintain changes made to individual assets and products from time to time. As the customer needs change constantly, it is essential to have the configuration of assets and products so as to have accurate decision making. This chapter threw light into change propagation with both forward and backward promulgation. The ACP algorithm supports different change dynamics. Since custom assets are also under configuration management it supports 23 cases of change propagation. It also shows significant performance improvement when compared with the method proposed by Chen Thao [123]. The subsequent chapter realizes the second part of the framework that is quality product derivation.