Configuration and build management of Product line development with Perforce
## Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Description</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010-03-15</td>
<td>Draft</td>
<td>Initial draft</td>
<td>Steve Kim</td>
</tr>
<tr>
<td>2010-04-08</td>
<td>Draft</td>
<td>Edited for grammar</td>
<td>Michael Alessio</td>
</tr>
<tr>
<td>2010-05-17</td>
<td>Final</td>
<td>For submission</td>
<td>Steve Kim</td>
</tr>
<tr>
<td>2010-05-18</td>
<td>Final</td>
<td>Edited for grammar</td>
<td>Michael Alessio</td>
</tr>
<tr>
<td>2010-05-19</td>
<td>Final</td>
<td>Edited for changing Figure 3 and 5</td>
<td>Steve Kim</td>
</tr>
</tbody>
</table>
# Table of Contents

1. Introduction 4  
2. Product-line development 4  
   1.1 Definition 4  
   1.2 Considerations 5  
   1.3 Depot structure 5  
   1.4 Branch Strategy 6  
   1.5 Baseline Strategy 7  
   1.6 Integration with the defect tracking tools 8  
   1.7 Build and release strategy 8
1. Introduction

Product-line development engineering (or Platform based development engineering) has been adopted in organizations that want to enhance their efficiency by simultaneously developing and releasing multiple embedded products.

Many organizations struggle with the complexity inherent in developing and enhancing a product family. This complexity can be managed both by maintaining a strong focus on reusable software architecture and by controlling the governance of the development process.

Variation management in product line development is the key element in our development process. Software development and maintenance are dynamic processes where software constantly evolves. So, configuration management is the control of the evolution of the system.

Concentrating on configuration management and build management, this articles explains how configuration and build management can be implemented in product-line development environments based on the experience of a customer who has developed multiple products simultaneously.

2. Product-line development

1.1 Definition

Product line development is a well-known approach in the field of software engineering. In a software product line, a set of related products are produced through the combination of reused core assets, together with product specific custom assets[1]. Due to the characteristics of product line development, it has management issues, such as change control and evolution management.

Component : A component is the basic unit for configuration management. For example, a single file could be a component. A set of files that unite to perform a function or form an inheritance tree is also called a component.

Asset : An asset is a collection of components. An asset may contain one or more components.

Core asset : A core asset contains a set of domain specific but application independent components that can be adapted and reused in various related products. The concept of a core asset is one of the most important in the product line development methodology.

Custom asset : A custom asset contains a set of application specific components. A custom asset is not designed for reuse, but...
produced for a specific application.

Product : A product is a collection of core assets and custom assets. Products can share the same or similar core assets.

Product instance : After a new product is produced, it may also need to be configuration managed. The product under configuration management is called a product instance.

1.2 Considerations

Product line development requires much effort to construct and maintain an architecture which can be continually reused by all devices. Many approaches to product line development are well defined, but, most organizations have failed to achieve it. Based on the experience of our customers, the maintenance of a reusable architecture and practices require much more in effort and cost than a single implementation.

Concentrating on configuration and build management, the considerations of developing and maintaining both core assets and custom assets are:

1) The code structure of repository to manage both a common assets and variant assets
2) Branch strategy for ruling the development process
3) Baseline strategy for each asset and whole products
4) Integration with defect tracking tools
5) Daily build and release process

1.3 Depot structure

The structure of the repository should take into consideration the software architecture, the structure of the development team, access control, and the characteristics of the product.

We can devise the several alternatives on how to organize assets into Perforce depots:

- Each depot holds one asset
- One depot holds all assets
- One depot holds multiple assets by the characteristics of assets. For example, all related core assets share a depot. This is preferred because of convenience.

As a rule of thumb, when the count of assets reaches over 10 items, you’d better to put related assets on a single depot. Too many depots usually take the time of users to navigate them.

We can categorize the types of depots into three types: core assets, custom assets, and advanced assets, which are immature assets under development for pilot products.

To manage assets, additional information should be also managed by the tools or the system. This includes the following:

1) The members who participated in developing each asset. Only, they can modify the codes of the asset.
2) The administrators of each asset. They are in charge of assigning new developers into the group, making a baseline for
releasing an asset.

3) The policy which controls integrating and locking the branch.

Each asset is controlled by a development team. There is a different level of control depending upon the type of asset. I introduced the concept of a “project” for manage an asset. Project is represented in Perforce as a directory or a branch, with a naming rule to differentiate between a Project and a directory.

Unfortunately, Perforce does not have the concept of a project built in, so a system was built to provide it. The following features are supported by this system:

- Adding and viewing registered members and administrators of the project
- Creating the branch and view the hierarchy of all branches which belong to a project
- The ability of a project administrator to set project permissions without the assistance of the administrator of Perforce administrator.

1.4 Branch Strategy

The decision on which branching strategy can be applied depends on several conditions; such as the maturity of the developers, the size of the team, and the quality requirement of the assets. When the size of team and the quality requirements are high, more branches are needed.

Core assets require a higher level of quality assurance activities such as a code review and testing. So, these changes should be separated by branch and should be controlled by policy. Because of this, more developers might be assigned to a core asset development team than a custom asset team.

In case of a custom asset, code is customized for a specific product. The quality requirement is less important and the size of the team can be smaller than that of a core team.

A project of developing assets has several branches in which the code is evolved and controlled by the policy. The branches under the project are logically organized as a hierarchy, which can guide the flow of integration among the branches. Please refer to Figure 3.

![Diagram of branch strategy](#)

In Figure 3, the activities of change management for developing a core asset are as follows:

1) Each developer owns a development branch in which he changes the codes to accomplish the change requirements.
2) When he completes his work, he integrates all changes to the upper branch.
3) On each change requirement, a code reviewer makes a review on all changes gathered from the lower branches by referring to changelists which are linked with Perforce jobs.

©Samsung SDS, 2011
4) After completing the code review, the passed changelists are integrated into the upper branch.
5) All changes from lower branches are built with other assets. After successfully building, SQA activities are performed.
6) After SQA activities, Release baseline is tagged on all successful code.

There might be an argument on maintaining the separate branches based upon code maturity. There are pros and cons; notwithstanding the complexity and performance, separated branches can allow us to release a safer asset to the developers who will combine them into their own assets.

When we devise branch strategies, the policies of configuration management that govern the activities of a project should also be defined. these policies usually control activities such as submitting, integrating, and labeling.

**Submitting**:

When changelists are submitted, they should be linked with at least one job. This insures that the integration between Perforce and the defect tracking tool are correctly are synchronized with each other. This can be implemented through a Perforce trigger. The trigger might be affected on the developer’s branch in which all changes were made.

**Integrating**:

The status of a job determines if it is allowed to integrate into the upper branch. The allowable statuses of jobs are different based upon the branch. The flow control for integration should prevent the developers making a mistake when they integrate their code. In the Figure 4, the sample policy of the integration flow among branches is shown. Most policies related integrating are implemented with scripts on Perforce broker.

<table>
<thead>
<tr>
<th>1. Inner project</th>
<th>2. Between the projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>from/to</td>
<td>from/to</td>
</tr>
<tr>
<td>DEV</td>
<td>DEV</td>
</tr>
<tr>
<td>FET</td>
<td>FET</td>
</tr>
<tr>
<td>INT</td>
<td>INT</td>
</tr>
<tr>
<td>REL</td>
<td>INT(PRA)</td>
</tr>
<tr>
<td>△</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>△</td>
</tr>
<tr>
<td>X</td>
<td>△</td>
</tr>
<tr>
<td>X</td>
<td>△</td>
</tr>
</tbody>
</table>

![Figure 4] The matrix of integrating flow among branches

**Labeling**:

The naming rules of label are verified with a trigger on label creation and modification, including allowing only the creator of label to modify the contents of label.

1.5 Baseline Strategy

Two types of baselines might be needed for supporting product line development. One type of baseline is attached to each asset. It can be attached on any branch of asset. The main purpose of this type of baseline is to get the list of all files with version for synchronizing into the workspace.

The other type of baseline is used to reproduce a set of all files which compose a product. In our environment, we need to use a label, to aggregate all assets, because the assets are dispersed. This introduces a new concept of attaching a baseline to a product. We call this a “Composite baseline”. When releasing a product, an integration is performed to make a composite baseline by aggregating baselines of the assets which compose a product.
Composite baselines should provide the following functionality in product line development.

1) To keep the labels which were already tagged on assets composing a product.
2) To synchronize all assets with a client workspace of developers by using a composite baseline.
3) To view all composite baselines belonging to each branch of a project.

A baseline is used for releasing a set of files with version on each public asset. The product developers who develop a custom asset use a released baseline to synchronize all assets into their client workspace in order to check whether his changes are compatible with the other assets, which are already released.

Baselines can also be used for showing the maturity of the code. Naming rules of baselines are very useful to indicate which maturity level of codes are acquired.

1.6 Integration with the defect tracking tools

Organizations over mid-size (developers > 50) already are usually fully equipped with defect tracking tools. Most change requests, such as new requirements, enhancement requirements, and defects, are managed by the defect tracking tool. All change requests which are related with modifying files should be integrated through Jobs between the defect tracking tool and Perforce. To avoid data inconsistency, one-way synchronization is recommended. This means that all contents of a change request can be modified only in the defect tracking tool.

When you link a job with changelists in P4, this relationship can be sent to the defect tracking tool. This feature gives the developer a way of finding out which changelists and files were modified for which change requests. With this relationship, we can generate release notes automatically when assets are released by collecting all jobs which have contributed to the current release of the asset.

1.7 Build and release strategy

In product line development, build and release management also are deemed to be more complex than conventional development. This complexity is caused by the difficulty to assemble the correct set of the right versions of the assets. The newly released core asset should be built and testified by all products with all the other assets. In real environment, most of the core assets are released frequently. This process made the developers feel that it was too difficult to combine a set of all assets without errors.

To avoid a build and release inefficiency caused by miscommunication among the project teams who develop assets, there should be a well-defined and detailed build and release process. The process should be specified as follows:

1) Which kinds of builds will be run and by whom
2) The rules for synchronizing source code when the builds of all products start
3) When assets will be released and which quality assurance activity will be done

There are several types of builds, such as Integration builds, daily builds, and CI (Continuous Integration) builds. In each project that develops assets, the integration builds and CI builds may be done during the business hours to verify whether the changes will be compatible with the other assets. And then, at night, the daily builds of all products will be started by the automation build tools with the same rules for synchronizing the source code.

At the morning of next day, the developers check the results of the daily builds of all products which combined both core assets and custom assets. If there are no build errors, then quality assurance work will be performed. And then, core assets and custom assets are released with the release baselines in turn.
Related to Perforce in build and release management. The following steps to synchronize the source codes in build automation tool should be provided:

1) Synchronizing the latest label of each asset
2) Synchronizing the latest composite baseline of a product
3) Synchronizing the specific label/changelist of each asset
4) Make a baseline of each asset
5) Make a composite baseline of a product
6) Automatically generates the release notes of each asset with P4 jobs which gathered from the defect tracking tools/systems

References