Configuration and Build Management of Product Line Development

Steve Kim (Sungchul Kim)
Principal Engineer
Samsung SDS
(sungchul3.kim@samsung.com)
AGENDA

Product-line development
- Definition
- Considerations

Usage models
- Depot structure
- Branch Strategy
- Baseline Strategy
- Integration with defect tracking tools & process
- Build and Release
GOALS

Primary
• Identify the considerations on applying product-line development
• Find out how to develop a usage model

Secondary
• Show how to implement product-line development with Perforce
Ch1. Product line development
Introduction

Enhance the efficiency of SW development when multiple products are to be developed simultaneously

• Higher productivity
• Higher quality
• Faster time to market
• Lower labor needs

Many methods and practices are introduced

• S/W Reuse
• Component-based development
• Product line engineering (Product family engineering)
Variation management is a key element to distinguish the other development process

- Reusable S/W Architecture
- Separated teams and responsibilities
- The governance enforcing S/W reused

The usage model of configuration and build management on product line development will be introduced

- Code structure
- Branch strategy
- Label strategy
- Change management
- Build and Release management
Definition of Product-line development

- A set of related products are produced through the combination of reused core assets together with product specific custom assets

[Adapted from “General configuration management and asset evolution model for software product line”, Softwareproductlines.com]
<table>
<thead>
<tr>
<th>Terms</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>The basic unit for configuration management</td>
</tr>
<tr>
<td>Asset</td>
<td>A collection of components. An asset may contain one or more components</td>
</tr>
<tr>
<td>Core Asset</td>
<td>Contains a set of domain specific but application independent components that can be adapted and reused in various related products.</td>
</tr>
<tr>
<td>Custom asset</td>
<td>Contains a set of application specific components</td>
</tr>
<tr>
<td>Product</td>
<td>A collection of core assets and custom assets. Products share the same or similar core assets.</td>
</tr>
<tr>
<td>Product instance</td>
<td>After a new product is produced, it may also need to be configuration managed. The product under configuration management is called product instance.</td>
</tr>
</tbody>
</table>
Variation management for Software Product-line

- Variation in time and space
- Divided into nine smaller issues and suggest the solution for each issue

[Nine Sub-problems of Variation management, Charles W. Krueger]
Ch2. Usage model of configuration and build management
Considerations

Considerations when you implement the usage model

- The code structure of repository to manage both a common assets and variant assets
- Branch strategy for enforcing the development process
- Baseline strategy for tagging each asset and a whole product
- Integration with defect tracking tools
- Daily Build and Release process
Depot Structure can be considered by the followings:

- SW architecture
- The structure of the development team (Single vs Multiple teams)
- The access control policy

- Each depot holds one asset
- One depot holds all assets
- One depot holds multiple assets by the characteristics of assets
Each asset is coded and then released by a team. All code is managed by a project.

- A management unit to control all activities related to an asset
- Using naming rules, you can distinguish a directory from a branch ("PRJ_", "[ ]")
- Grouping related projects with directory can be possible

Depot holds several projects in which the code for assets are managed.

Directory can be used to group the related projects - Year, Chipsets, Oversea labs.
Depot Structure

Additional data is also needed to maintain a Project.

• Project data including Name, Description, Depot path
• Branches and the hierarchy of branches which belong to a project
• The administrators of each project. They are in charge of assigning new developers and making a baseline for releasing an asset
• The policy which controls integrating and locking the branch
• Sometimes, the protection table of each project can also be managed

These can be stored in a depot or other storage such as a database
Sophisticated Code-line strategy is required
• The size of team is getting bigger
• The quality requirement of assets is getting higher

The quality requirement of assets
• Core Assets is higher than Custom assets
Sparse branching is a good solution to avoid integrating the files which are not changed in the developer’s branch.

1. Only Module 1 is branched.
2. Integrate Dev-One to Mainline.
3. Rebase the changes into Dev-One.

[ View spec using Sparse branching ] - Refer to KB #890

```
//Core_Assets/PRJ_Core_Asset/Mainline/... //wk/Core_Asset1/...
+//Core_Assets/PRJ_Core_Asset/Dev_One/ComponentA/Module1/... //wk/Core_Asset1/Module1/...
```
Branch Strategy

Activities related with integrate should be controlled

• Submitting : ( P4 Trigger )
  - Linked with at least one job ( Activity based change management )

• Integrating : ( P4 Broker )
  - The status of jobs linked with changelists is verified
  - Rebase the integrated codes from the target branch before integrating
  - The integrating flow is controlled ( refer to the below tables )

### 1. Inner project

<table>
<thead>
<tr>
<th>From/To</th>
<th>DEV</th>
<th>Sub-INT</th>
<th>INT</th>
<th>REL</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEV</td>
<td>△</td>
<td>O</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Sub-INT</td>
<td>O</td>
<td>X</td>
<td>O</td>
<td>X</td>
</tr>
<tr>
<td>INT</td>
<td>X</td>
<td>O</td>
<td>-</td>
<td>O</td>
</tr>
<tr>
<td>REL</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
</tbody>
</table>

### 2. Between the projects

<table>
<thead>
<tr>
<th>From/To</th>
<th>DEV</th>
<th>Sub-INT</th>
<th>INT</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEV</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Sub-INT</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>INT</td>
<td>X</td>
<td>△</td>
<td>△</td>
</tr>
</tbody>
</table>

O : Allowed, X : Disallowed, △ : Configured depends on each project
Label Strategy

Two types of Baseline will be used
- Baseline: Attached to each asset when the code of the asset is released

  The use of naming rules to indicate the maturity level of code is very useful
  ➞ _INT, _REL

- Composite Baseline: A set of Baselines to reproduce all files which compose a product
  - Keeps the labels of assets composing a product
  - Easily synchronizes all files of a product with a client workspace for developers who don’t know the combination of baselines and assets
  - Can also include composite baselines recursively
Label Strategy

Composite Baseline of Product A

Composite Baseline of Asset Group A

Baseline of Asset C1

Baseline of Asset C2

Baseline of Asset A1

Baseline of Asset A2

Baseline of Asset P1

<table>
<thead>
<tr>
<th>Number</th>
<th>Project Name</th>
<th>Stream Name</th>
<th>Depot Path</th>
<th>Label Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Core Asset C1</td>
<td>INT</td>
<td>//Core Asset_C/Project_C1/INT</td>
<td>C1_0420_Release</td>
</tr>
<tr>
<td>2</td>
<td>Core Asset C2</td>
<td>INT</td>
<td>//Core Asset_C/Project_C2/INT</td>
<td>C2_0419_Release</td>
</tr>
<tr>
<td>3</td>
<td>Custom Asset P1</td>
<td>Sub-INT</td>
<td>//CustomAsset_P/Project_P1/Sub-INT</td>
<td>P1_0420_Release</td>
</tr>
<tr>
<td>4</td>
<td>Core Asset A1</td>
<td>INT</td>
<td>//Core Asset_A/Project_A1/INT</td>
<td>A1_0419_Release</td>
</tr>
<tr>
<td>5</td>
<td>Core Asset A2</td>
<td>INT</td>
<td>//Core Asset_A/Project_A2/INT</td>
<td>A2_0419_Release</td>
</tr>
</tbody>
</table>
Integrating with the defect tracking tools

Change based modification should be enforced

- All work items such as Change Requests and Defect can be synchronized through Jobs in Perforce
- Synchronizing can be considered as one-way or two-way
  - One-way: Defect tracking tool can only change the job status
  - Two-way: Both Perforce and the Defect tracking tool can change the job status
- The policy to verify the job status and whether it can be promoted
  - P4 broker can be a candidate to implement those policies
On each asset (project), it’s own access control mechanism is also required

- Set access control on each code-line by groups
- Freeze some code-lines during integrating and building. But, exceptional users can be allowed

<table>
<thead>
<tr>
<th>No</th>
<th>Access Level</th>
<th>User/Group</th>
<th>Name</th>
<th>Host</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>write</td>
<td>group</td>
<td>Default group</td>
<td>*</td>
<td>//Depot/[Core_Prj]/...</td>
</tr>
<tr>
<td>2</td>
<td>read</td>
<td>group</td>
<td>Additional Group</td>
<td>*</td>
<td>//Depot/[Core_Prj]/Mainline/CompA/...</td>
</tr>
</tbody>
</table>
Integrating with the defect tracking tools

Perforce
- Job
- Changelist
  - File

Defect Tracking Tool
- DEFECT (CR,...)
- Release Note

Bidirectional Synchronizing
- p4Change

One-way Synchronizing
- p4Change

Sync.Gateway

<Perforce>

<ClearQuest>
Build and Release strategy

More complex than conventional development
• The difficulty is the combination of the right versions of the right assets
  - Every asset is changed continually
  - Each team has different rules when managing stable code
• Core assets should be released frequently and these also should be testified on the all products
  - Building and testing on all products requires time and effort
• Miscommunication among the development teams
  - Major changes such as interface changes lack notification
  - The time when applying the change of asset can be disordered
Build and Release strategy

The followings should be clearly defined:

• All products should be built with the same rules how to combine the assets

• Daily builds on all products should be run, and the results of daily builds also are shared with all developers by indicating which assets were failed

• Periodically, the causes of the build error are analyzed and improved
Daily Builds and release process can be depicted as the below:

1. Daily Build
   - Daily Build (All models)
   - Build Result: Success or Fail
   - Fix the error
   - Run builds

2. Release Core Assets
   - Runs Tests
   - Release the Core assets with Baseline

3. Release Products
   - Get the released Core Assets
   - Build & Test
   - Release

Steps:
- Run the CI builds
- Run the daily builds
- Identify and fix the daily builds

Build management System

Core Assets Builders

Product’s Builders
Build and Release strategy

The following features can be supported by build automation tool:

- **Synchronize**
  - the latest label of each asset
  - the latest composite baseline of a product
  - the specific label/changelist of each asset
- **Make a baseline**
  - each asset
  - a composite baseline of a product
- **Integrate codes into the upper branch and make a baseline after the build was completed successfully**
  - Builds ➔ Unit Test ➔ Integrate Codes ➔ Make a baseline
  - Automatically generates the release notes of each asset with P4 jobs which gathered from the defect tracking tools