Modularization of Large Projects Using Eclipse and OSGi

Marc R. Hoffmann · 16.10.2012 · Eclipse Finance Day Zurich
Let’s Face Reality.
Tight Coupling, Unused Code, Duplications, Unclear Status
Country Wide

2h Forecast for all Trains

Real-Time Operational Data

Level of Detail: Tracks & Signals

SBB CFF FFS

Rail Control System (RCS)
Modularization Objectives

• Multiple projects with different life cycles
• Improved code re-use
• Manage 3rd party dependencies
Module, Model, Build & IDE, 3rd Party Libs, Modularize Code, Module Runtime

OSGi, Eclipse, Maven, Tycho
Module Model
Build & IDE
3rd Party Libs
Modularize Code
Module Runtime

OSGi & Eclipse
ch.sbb.project/
ch.sbb.testproject/
ch.sbb.project.subproject.bundle/
ch.sbb.project.subproject.bundle.test/
ch.sbb.project.subproject.bundle/
ch.sbb.project.subproject.bundle.test/
ch.sbb.project.subproject.bundle/
ch.sbb.project.subproject.bundle.test/
org.apache.commons.logging/
org.slf4j/
Module, Model, Build & IDE, 3rd Party Libs, Modularize Code, Module Runtime

Component Life Cycle

In Production
- ch.sbb.project.a/
- ch.sbb.project.subproject.bundle/
- ch.sbb.project.subproject.bundle.test/

In Development
- ch.sbb.project.b/
- ch.sbb.project.subproject.bundle/
- ch.sbb.project.subproject.bundle.test/
Module, Model, Build & IDE, 3rd Party Libs, Modularize Code, Module Runtime

Single Source

`.product`

`feature.xml`

`MANIFEST.MF`

Maven/Tycho Build

Eclipse IDE
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>ch.sbb.project</groupId>
  <artifactId>ch.sbb.project.bundle</artifactId>
  <description/>
  <packaging>eclipse-plugin</packaging>
  <parent>
    <groupId>ch.sbb.rcs</groupId>
    <artifactId>master.pom</artifactId>
    <version>3.4.0-SNAPSHOT</version>
  </parent>
</project>
Module, Model, Build & IDE, 3rd Party Libs, Modularize Code, Module Runtime

Eclipse IDE

- Unified project layout
- Shared project settings
- Project Sets generated from features
Build Best Practices

- Dedicated build manager
- Automation is the key
- Keep tools up-to-date
- Regularly collect requirements from dev teams
The End
(of your project’s lifecycle)
• What exact version do we use?
• Who depends on this library?
• What dependencies exist between libraries?
• Should we upgrade?
• What licenses do we use?
• Identify each library’s coordinates

• Create a separate bundle for each library incl. source JARs

• Minimize dependencies

The Hard Work
pom.xml

```xml
<artifactId>org.apache.commons.collections</artifactId>
<packaging>eclipse-plugin</packaging>
<bundle.version>3.2.1</bundle.version>

<dependencies>
  <dependency>
    <groupId>commons-collections</groupId>
    <artifactId>commons-collections</artifactId>
    <version>${bundle.version}</version>
  </dependency>
</dependencies>

tycho-packaging-plugin:generate-bundle
Separate API from Implementation
Module, Model, Build & IDE, 3rd Party Libs

Modularize Code, Module Runtime

API

Intern

Module, Run*me, 3rd Party Libs

API

Module, Run*me
Good Reasons

- New features
- Existing implementation needs to be re-used for new modules
- Re-design required for new requirements

When?
Module, Model, Build & IDE, 3rd Party Libs, Modularize Code, Module Runtime

Domain

Technology

Guidelines
OSGi Stack

Module Model
Build & IDE
3rd Party Libs
Modularize Code
Module Runtime

Execution Environment
Modules
Life Cycle
Services
Application Bundles
What do we get?

Modules

• Separate class loaders

Bundle life cycle

• Explicit initialization

Services

• Extensibility
Module, Model, Encapsulation, Dependency Management, IDE Support, Testing, Project Layout, Build, Packaging
RCS Project
500 Bundles
25 Products

Concepts
Tycho
Build
OSGi Runtime

Modularization

2010  2011  2012  2013
Thank You! Questions?

Marc R. Hoffmann  •  hoffmann@mountainminds.com