Case Study

Semantic Versioning A Large Existing Codebase

EclipseCon 2014

Raymond Augé <raymond.auge@liferay.com (mailto:raymond.auge@liferay.com)>

@rotty3000 | #eclipsecon #semver

Outline

- The Goal
- The Problem
- Semantic Versioning
- BND
- The conclusion
- The Project
- The solution
- Tools
- Our Enhancements

The Goal

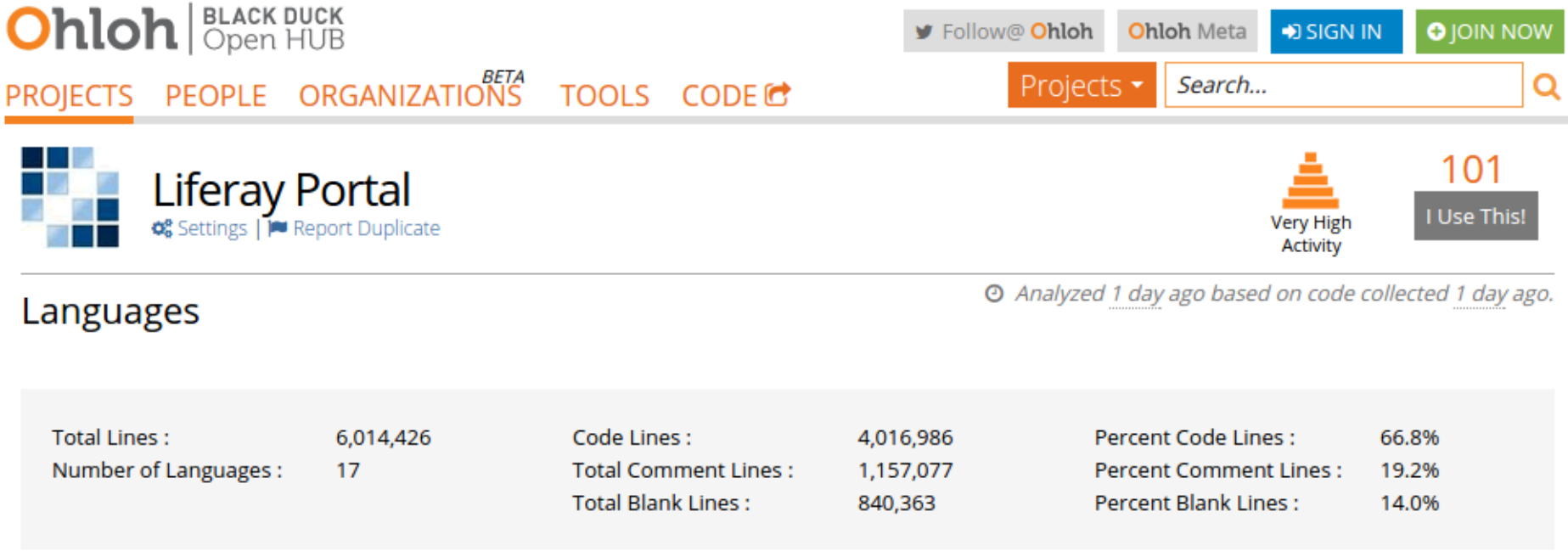
Improve Code Quality !

Increase Developer Joy !

The Project

Liferay Portal project is a large open source application

A large community of users and contributors



⚠ The problem

Product versioning scheme

- **Minor** Releases (6.1.0 ➔ 6.2.0)
  - **Binary patch** security hot-fixes
  - Collective **Fix-Packs**
  - No incremental feature updates
  - API Breaking changes (**hopefully not!**)

⚠ The problem

Product versioning scheme

- **Major** Releases (6.2.0 → 7.0.0)
  - More than **1 year** between releases
  - Monolithic upgrades
  - API Breaking changes (**guaranteed!** Best effort to know which!)
- Tight coupling rampant across features

The solution

To a large degree resolving these issues begins with **API Management.**

The solution

Controlled evolution demonstrates **API Reliability**

Ability to resolve implementation issues while maintaining compatibility demonstrates **API Stability**

Though there are no measures of **API Productivity**, it's clear that good APIs improve developer productivity

The solution

Reliability, **Stability**, **Productivity** make happy Developers / Users / Customers!

Semantic Versioning

➔ Achieve **programmatic detection** of API changes

➔ Provide developers with tools to ease the learning curve

Artifact granularity is not sufficient when size exceed some inordinate number of APIs.

Semantic Versioning

Using the **OSGi Alliance Semantic Versioning** white paper definition

http://www.osgi.org/wiki/uploads/Links/SemanticVersioning.pdf (http://www.osgi.org/wiki/uploads/Links/SemanticVersioning.pdf)

The unit of granularity is **package**.

➔ However, implementing Semantic Versioning at the scale of this project is a significant amount of work!

🔧 Tools

➔ The key to **Semantic Versioning** is tooling.

Humans are biased, error prone

Semantic Versioning is boring and tedious

Let machines do the work

There aren’t many well known tools.

★ **Fortunately**, the most well known one is **FANTASTIC** (and can provide even more useful information than it currently lets on.)

Tools

With proper tooling it's simple to adopt Semantic Versioning **incrementally**!

🪄 BND

> "bnd is the Swiss army knife of OSGi, it is used for creating and working with OSGi bundles. Its primary goal is take the pain out of developing bundles."
>
> *http://www.aqute.biz/Bnd*
>
> — Peter Kriens

➔ http://www.aqute.biz/Bnd/Bnd (http://www.aqute.biz/Bnd/Bnd) (the library)
➔ http://bndtools.org/ (http://bndtools.org/) (a complete OSGi Suite for Eclipse)

BND - Baseline

> "Baselining compares the public API of a bundle with the public API of another bundle."
>
> *http://www.aqute.biz/Bnd/Versioning*
>
> — Peter Kriens

BND - Baseline

Invocation

```
java -jar biz.aQute.bnd-latest.jar baseline -d ./biz.aQute.bnd-latest.jar
/other/bnd.jar
```

Output

```
================================================================
  biz.aQute.bnd 2.3.0.20140315-151701-2.2.0.20131017-210830
================================================================
  Package                                                 Delta      New        Old
Suggest    If Prov.
  aQute.bnd.build                                         MINOR      2.3.0      2.2.0
ok         -
  aQute.bnd.header                                        MINOR      1.3.0      1.2.0
ok         -
  aQute.bnd.osgi                                          MINOR      2.2.0      2.1.3
ok         -
  aQute.bnd.service.classparser                           ADDED      1.0.0      -
ok         -
  aQute.bnd.service.extension                             ADDED      1.0.0      -
ok         -
  aQute.bnd.service.phases                                ADDED      1.0.0      -
ok         -
  aQute.bnd.service.repository                            MINOR      1.3.0      1.1.0
ok         -
  aQute.bnd.service.url                                   MINOR      1.2.0      1.1.0
ok         -
  aQute.bnd.version                                       MINOR      1.1.0      1.0.0
ok         -
```

Easily recognize degree of version change

Detects all types of change

🪄 BND

➔ The project started with no versioning, and with developers ignorant about Semantic Versioning.

BND’s existing information does not provide enough detail for less experienced developers to understand how things get broken.

🪄 BND

Internally BND performs exceptionally detailed API analytics, not exposed in it’s default output

The project needed that information ❗

➔ None of our code is OSGi aware. BND is designed to operate with OSGi bundles.

🪄 BND

Can BND still be used when the code isn't OSGi ready **?**

➔ **Yes!** Even the most basic BND configuration is useful

```
Bundle-SymbolicName: ${bundle.name}
Bundle-Version: ${bundle.version}
Export-Package: *
Import-Package: *
```

BND - Creating Jars

Consider this simple directory structure

```
/bnd.bnd
/bnd.jar
/src/main/java/com/test/Fee.java
/src/main/java/com/test/IFoo.java
```

Fee.java

```
----
package com.test;
public class Fee {
  public void doFee() {}
  public void doFee2(IFoo foo) {}
}
----
```

IFoo.java

```
----
package com.test;
public interface IFoo {
  public void doFoo();
}
----
```

```
bnd.bnd
----
Bundle-SymbolicName: a
Bundle-Version: 1.0.0
Export-Package: *
Import-Package: *
Include-Resource: build/classes
-output: build/libs/${Bundle-SymbolicName}.jar
----
```

🪄 BND - Creating Jars

Compile

```
mkdir -p build/classes
javac -d build/classes $(find . -name "*.java")
rsync -aq --exclude '*.java' src/main/java/* build/classes/
```

Jar with BND

```
mkdir -p build/libs
java -jar bnd.jar bnd -p bnd.bnd
```

BND - Creating Jars

Result

```
build/libs/a.jar!MANIFEST.MF
----
Manifest-Version: 1.0
Bnd-LastModified: 1395117308444
Bundle-ManifestVersion: 2
Bundle-Name: a
Bundle-SymbolicName: a
Bundle-Version: 1.0.0
Created-By: 1.7.0_51 (Oracle Corporation)
Export-Package: com.test;version="1.0.0"
Include-Resource: build/classes
Require-Capability: osgi.ee;filter:="(&(osgi.ee=JavaSE)(version=1.7))"
Tool: Bnd-2.3.0.20140315-151701
----
```

🪄 BND - Baseline

Baseline is the operation of comparing one jar to a previous version of the same jar in order to analyze for API changes.

The base case

```
java -jar bnd.jar baseline build/libs/a.jar repo/a-latest.jar
============================================================
  a 1.0.0-1.0.0
============================================================
```

BND - Baseline

```
IFoo.java
----
package com.test;
public interface IFoo {
  public void doFoo();
  public void doFoo2();
}
----
```

Added a new method

Produces

```
java -jar bnd.jar baseline -d build/libs/a.jar repo/a-latest.jar
================================================================
* a 1.0.0-1.0.0 suggests 2.0.0
================================================================
  Package                                                 Delta      New        Old
Suggest    If Prov.
* com.test                                                MAJOR      1.0.0      1.0.0
2.0.0      1.0.0
```

Adding a method to an interface is a `MAJOR` change

BND - Baseline

```
IFoo.java
----
package com.test;
@aQute.bnd.annotation.ProviderType
public interface IFoo {
  public void doFoo();
  public void doFoo2();
}
----
```

Added the `@ProviderType` annotation

Produces

```
java -jar bnd.jar baseline -d build/libs/a.jar repo/a-latest.jar
================================================================
* a 1.0.0-1.0.0 suggests 1.1.0
================================================================
  Package                                                 Delta      New        Old
Suggest    If Prov.
* com.test                                                MINOR      1.0.0      1.0.0
1.1.0      -
```

The change is now `MINOR` and suggested version reflects this

BND - Baseline

Baseline indicates that the package's version still needs to be properly assigned.

To assign a proper version create a text file called `packageinfo` in the package directory.

```
packageinfo
----
version 1.1.0
----
```

Rebuild and baseline

```
java -jar bnd.jar baseline -d build/libs/a.jar repo/a-latest.jar
================================================================
* a 1.0.0-1.0.0 suggests 1.1.0
================================================================
  Package                                                 Delta      New        Old
Suggest    If Prov.
  com.test                                                MINOR      1.1.0      1.0.0
ok         -
```

Note the package state is no longer *dirty*.

Baseline now suggests that the library version be increased to `1.1.0` but refrain from doing so until the lib is ready to release.

⊞ Our Enhancements

Our team wrapped BND operations for use in ant and gradle.

This allowed us deeper access to the extensive information BND has available.

Different reporting levels allow developers to choose what most suits their needs.

Baseline reporting is automatically enabled for all builds and uses a remote repository for zero configuration setup.

Optionally, persisted reports can be reviewed later or used by things like CI to fail builds, etc.

⊞ Our Enhancements

Case #1

```
  PACKAGE_NAME                                                DELTA      CUR_VER    BASE_VER
REC_VER    WARNINGS
= ================================================== ========== ========== ==========
========== ==========
* com.liferay.portal.kernel.monitoring.statistics    MAJOR      6.2.0      6.2.0
7.0.0      VERSION INCREASE REQUIRED
  >   class      com.liferay.portal.kernel.monitoring.statistics.DataSampleThreadLocal
    -   implements java.lang.Cloneable
    >   method     clone()
      +   access     protected
    +   method     initialize()
      +   access     static
```

Interface removed

Method signature changed from public to protected

Static method added

⊞ Our Enhancements

Case #2

```
  PACKAGE_NAME                                                DELTA      CUR_VER    BASE_VER
REC_VER    WARNINGS
= ================================================== ========== ========== ==========
========== ==========
* com.liferay.portal.kernel.template                    MAJOR      6.3.0      6.2.0
7.0.0      VERSION INCREASE REQUIRED
  <   class      com.liferay.portal.kernel.template.BaseTemplateHandler
    +   method     getTemplatesHelpContent(java.lang.String)
      +   return     java.lang.String
    +   annotated  aQute.bnd.annotation.ProviderType
  >   class      com.liferay.portal.kernel.template.TemplateHandlerRegistryUtil
    -   method     <init>()
[snip]
```

Abstract class

Annotated as `@ProviderType`

Deletion of a method is always `MAJOR`

Case #2 (cont')

```
  PACKAGE_NAME                                                DELTA      CUR_VER    BASE_VER
REC_VER    WARNINGS
= ================================================== ========== ========== ==========
========== ==========
* com.liferay.portal.kernel.template                  MAJOR      6.3.0      6.2.0
7.0.0      VERSION INCREASE REQUIRED
[snip]
  <   interface  com.liferay.portal.kernel.template.TemplateHandler
    +   method     getTemplatesHelpContent(java.lang.String)
      +   access     abstract
      +   return     java.lang.String
    +   annotated  aQute.bnd.annotation.ProviderType
  -   interface  com.liferay.portal.kernel.template.TemplateHandlerRegistry
[snip]
  -   version    6.2.0
  +   version    6.3.0
```

Interface is modified

Method is added

Because it's `@ProviderType` change is `MINOR`

Version had previously been increased, but now it's `MAJOR`

! The conclusion

➔ Did we achieve our goals?

Improve Code Quality !

More quickly identify problem areas

- *Catch all* packages suffer too much change (too many classes affected): how should classes be slit up logically
- Over embellished bug fixes: never mix bug fixes and API changes
- Bad design decisions are more obvious

! The conclusion

➔ Did we achieve our goals?

Improve Code Quality !

Packages which don't change over time are either very stable or unused

➔ **stable**: *Isolate and congratulate the maintainer*

➔ **unused**: *Delete without prejudice*

Packages which do change frequently are possible problem areas, or need to be isolated into individual modules

! The conclusion

➔ Did we achieve our goals?

Increase Developer Joy !

Developers are more accountable (oddly this makes other developers happy)

Ability to produce **100% accurate reports of API change** across the entire product: means they have a reliable source of information

Just like automated tests, automated API change detection makes developers feel more confident

Increased enthusiasm evident among our developers

⏻ Thank You!