Escaping the Works-On-My-Machine badge
Continuous Integration with PDE Build and Git

Matthias Kempka
EclipseSource

mkempka@eclipsesource.com
About EclipseSource

• Eclipse development and consulting in Germany and around the world
• Committers and Leaderships in over 10 Eclipse projects, among them Equinox, RAP, EPP
The Works-On-My-Machine certificate

- Compile application code (optionally check out latest changes from other developers)
- Launch application
- Execute one simple code path of your code changes (Unless you are certain your code works)
- Check in changes
Release Engineering with Eclipse

- OSGi and p2 have special requirements
- Build tools with steep learning curve
- Release engineering is not a normal programming task
## Choice of the Build system

<table>
<thead>
<tr>
<th>Make simple things simple</th>
<th>PDE Build/Ant</th>
<th>Tycho/Maven</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make advanced stuff possible</td>
<td>X</td>
<td>(X)</td>
</tr>
</tbody>
</table>
A working example with PDE Build and Git

• An example product: Hyperbola
• Build for a RCP product and Test Automation
• System independent, mostly self-contained
• Versioning System: Git
A working example with PDE Build and Git

• Fetch `git://github.com/mkempka/hyperbola-pde-build-demo.git`
• Adjust values in `o.e.h.releng/custom.properties`
• Execute ant file `o.e.h.releng/build.xml`
• Preconditions:
  - Eclipse BaseBuilder $\geq$ Indigo
  - Git must be on your $\$PATH$
Scope of this talk

• Walk through some common challenges of projects in the real life
• Present common patterns to address these challenges
• Point to the implementation in the example
Problem: Code in SCM does not compile

- Wrong source code level
- Forgot to check in a class
- Developed against different target
- But: Where should it compile?
Solution: Have a Continuous Integration Build

• Defines the current project sanity
• CI is an Agile best practice
• Feedback from an independent server
  ◦ Best run with a CI server, i.e. Jenkins, Bamboo, ...
• Ideally executed soon after a commit
Problem: Setting up a new workspace

• Which projects do I need?
• Where should they be on the disk?
• What is my target?
• Where does the target come from?
Solution

• Workspace Provisioning with **Yoxos**
• Target Definition
Managing a Target

• Use “Add Content” from “Software Site” and add a p2 repository

• A target definition against p2 repository at Eclipse will work
  - most of the time
  - potentially very slow

• Use the p2 mirror application to manage your own mirror
  - See http://wiki.eclipse.org/Equinox_p2_Repository_Mirroring
Target content

• For a RCP application use these features:
   Eclipse RCP
   Eclipse RCP Plug-in Developer Resources
   Eclipse Platform Launchers
   ... Your own

• To enable the Test framework
   Eclipse Test Framework
   PDE JUnit Runner Support - Add-on
Build target

• PDE Build does not understand Target Definitions yet
  * See Bug 266311

• In customTargets.xml, step preProcessRepos, use

  `<p2.mirror source="${baseRepository}"`
  
  `destination="file:${repoBaseLocation}/mirrored">
  
  `<iu id="org.eclipse.equinox.executable.feature.group"
     version="3.5.0.v20110530-7P7NFUFFLWUL76mart"/>
  
  ...

  `</p2.mirror>`

• Tip: id and version tag can be copied&pasted from the target definition source
Problem: A Software Update is required

- A customer needs a hotfix for a bug
  - The Master branch has already moved beyond the release state
- Or you just have to update your software
  - How big is your update?
Solution: Release from tags

• Separate between CI and release builds
• Release builds are tested and eventually released
• Release builds contain tagged bundles
• Tags are managed in map files
A map file

• A map file contains a line per artifact like this:

```text
plugin@o.e.h.core=GIT,tag=v20100110,repo=REPO,path=o.e.h.core
```

• During the build, the fetch factory will fetch the tag v20100110 tag from repository REPO
Map files and CI builds

• To build the latest version of a specific branch, start the build with

  -DfetchTag=master

• For getting useful qualifiers, also use

  -DforceContextQualifier=$NOW
Releasing from map files

• Edit map files by hand only to
  - Add a plugin/fragment/feature
  - Remove a plugin/fragment/feature
• Automation for managing the tags
  - Eclipse Releng tool for CVS/SVN
     - Scripts for Git, see Bug 328745
Problem: Builds take longer and longer

- Compiling more often than required
- Unnecessary compression and extraction
- Test execution takes a long time
Compile only once

- Compile an ‘Everything’ feature up front into a p2 repository
- Use that repository for the next steps
  - Bundle up products
  - Install test feature
Separated steps in the example

- The example has a main build.xml
- It configures and kicks off 2 consecutive PDE Builds and the test run
  - Compilation
  - Product Assembly
  - Run Tests with Eclipse Testing Framework
Categorize and separate test execution

- Separate fast from long running tests
  - Execute them i.e. at CI Build and Nightly builds
- **PatternTesting** library can help with test annotations
  - @IntegrationTest("online access needed")
  - @SmokeTest
  - @RunTestOn(osName = "Windows")
  - @SkipTestOn(osName = "Windows")
  - @Broken(till = "31-Jul-2011")
Problem: Poor definition of executed tests

- Developers rarely execute all tests in the workspace
- Build has some unknown method of test identification
Solution: Define a mandatory test suite

• The same test suite must be able to run from the workspace and the automated builds

• Use a test suite that can pick up tests automatically, i.e. Bundle Test Collector
BundleTestCollector

• A bundle o.e.h.alltests contains
  ♦ the class BundleTestCollector (c&p’d)
  ♦ the Test Suite(s)

```java
public class AllTests {

    public static Test suite() {
        TestSuite result = new TestSuite("All Hyperbola Tests");
        BundleTestCollector collector = new BundleTestCollector();
        collector.collectTests(result, "o.e.h", "o.e.h", "*_Test");
        return result;
    }
}
```
Problem: Unknown Test Quality

- Writing tests is optional in some teams
- Tests with a high abstraction execute only some code in distant classes
Solution: Check your code coverage

• EclEmma does a good job in the workspace
• Emma also has an integration in Hudson/Jenkins
• Use Emma in the build automation
Setting up Emma with PDE build

- Set up test environment
- Trigger offline instrumentation
- Execute tests
- Run Emma report
Setting up the Test Environment

• Unzip your product

• Install test feature with the **p2 director**
  • Do not drop files into dropins/ folder, logging output is very unreliable there

• See `o.e.h.releng/03-runtests/build.xml` for an example
  • This is the place for additional setup hooks, i.e. server/database setup
Trigger offline instrumentation

<!-- this requires emma.jar and emma_ant.jar and emma_ant.properties in your ant classpath -->
<taskdef resource="emma_ant.properties" />

<emma enabled="${emma.enabled}">
  <instr mode="overwrite"
    metadatafile="${coverageOutputDir}/ecp.em"
    filter="my.namespace.*">
    <instrpath>
      <fileset dir="${plugins-dir}"
        include name="my.namespace*.jar"/>
    </fileset>
  </instrpath>
</instr>
</emma>
Running the Tests

• Instrumented code requires Emma classes
  ✷ They are not in the bundle dependencies
• Put `emma-<version>.jar` in `${JRE_HOME}/lib/ext`
• Launch the test run with
  `-Dosgi.parentClassloader=ext`
Run Emma report

```xml
<emma enabled="${emma.enabled}">
  <report>
    <infiles dir="${coverageOutputDir}" includes="*.em,*.ec"/>
    <xml outfile="${coverageOutputDir}/coverage.xml">
      columns="class, name, method, block"/
    </xml>
    <html outfile="${coverageOutputDir}/coverage.html"/>
  </report>
</emma>
```
Configure Hudson/Jenkins

- Install Emma Plug-in
- Configure project

✓ Record Emma coverage report
Folders or files containing Emma XML reports

builds/tests/ecp/results/testreports/coverage*.xml

- Enjoy coverage reports
Other features in the example

• Start PDE from an ant build file
  • No need to specify the version of
    org.eclipse.equinox.launcher_<version>.jar

• Build feature branches before committing
Conclusion

• Setting up and maintaining a build has many facettes
• The example has many features already integrated and for similar builds only a handful properties need to be adjusted
• More about this at http://eclipsesource.com/blogs
Image Sources


• Light Bulb and Warning Icons via Creative Commons Attribution 3.0 Unported by [http://shlyapnikova.deviantart.com](http://shlyapnikova.deviantart.com)