Deploying Successful Enterprise Tools

Eclipse Day at GooglePlex 2009

Joep Rottinghuis
Productivity Tools Architect, eBay Inc.
August 27, 2009

Abstract

For a tool to be successful in an enterprise, it takes much more than writing a great Eclipse plug-in and tossing it out there.

Most tool developers focus on the early phases of PDLC such as UI design, coding functionality and performance. These are necessary, but not sufficient.

This presentation will cover various other aspects that drive and sustain adoption:

- testing
- deployment mechanisms
- usage tracking
- maintenance
- documentation
- support and user education

Joep will discuss the challenges and solutions found at eBay to make Eclipse based tools successful in a large, fast-paced organization.

Agenda

1. testing
2. deployment mechanisms
3. usage tracking
4. maintenance
5. documentation
6. support and user education

Testing

- **UI testing is challenging**
  - Avoid chaining use cases even with UI testing tool.
- **De-couple your UI from the Model/Controller**
  - Beware of unexpected dependencies on Eclipse bundles
- **Use your tool for your own development**
- **Use beta-testers**
  - Hard to control

Deployment Mechanisms

1. **Manual**
   - Fixed issues re-appear
2. **Eclipse Update Site**
   - Older mechanism
3. **P2 Repository**
   - Good for latest & greatest Eclipse plug-ins
   - Need to control & notify user when to check for new version
4. **JNLP (RCP) App**
   - Good for latest & greatest
   - Great for “non-Eclipse” users
5. **Per workspace**

Deployment Mechanisms

4. JNLP (RCP) App pitfalls:

- Signing of all jars (including RCP jars) is required
- Pack200 re-arranges jar contents which invalidates jarsigner hash: repack, sign, pack
- Browser JRE plug-in version mismatch
- 32-bit SWT libraries on Windows
- IE Browser cache full
- Java WS download pop-up in headless mode on Unix
  - `javaws -import -silent <url>`
  - `javaws -Xnosplash <url>`
- Compatibility key

- Classloader implementation of OSGi bundle visibility bites JDK logging
- Eclipse <3.0 plug-ins in RCP

Deployment Mechanisms

1. Manual
2. Update site
3. P2 Repository
4. JNLP (RCP) App

5. **Per workspace**
   - Plug-in coupled with code: Code generation, tools use and create libraries.
   - Requirement to roll forward and backwards
   - Requirement to have varying versions in different workspaces
   - Current solution: auto-config. Next-gen: workspace materialization

Per Workspace plug-ins

Pros:

1. Plug-in code can keep pace with workspace version
2. When debugging code, the plug-in matches the source code
3. API’s of underlying libraries can evolve (even break backwards compatibility)
4. Tool (Plug-in) can be used to determine which plug-ins need to be installed

Per Workspace plugins

Cons:

1. OSGi bundle (bundle symbolic name) can be part of either Eclipse installation or workspace, but not both.
2. File->Switch workspace does not work anymore
3. Requires modification of Eclipse startup
4. Eclipse 3.4 / 3.5 P2 compatibility ok per documentation (being tested)
5. Linux / Mac compatibility TBD

Eclipse startup ~~hack~~ hook

Logical steps in use case:

a) Start Eclipse as usual
b) Have your own "Config" plug-in
c) Identify required per-workspace features
d) Identify versions of features/plug-ins/jars needed
e) Source the feature binaries
f) Prepare install site in current workspace
g) Restart Eclipse using additional startup parameters

Eclipse startup ~~hack~~ hook 2

To get per-workspace plug-ins requires the following changes to Eclipse:

- **Modify how Eclipse starts**
  - Eclipse 3.2.x replace startup.jar
  - Eclipse 3.3.x replace org.eclipse.equinox.launcher
- **Modify main class**
  - Rename `run` method to `basicRun`
  - Add a new `run` method to add your own startup parameters passed to `Eclipse.exe`
- **Add runtime option -configuration**

Eclipse startup ~~hack~~ hook 3

```
org.eclipse.equinox.launcher.Main

public class Main {
...
public int run(String[] args) {
int result = 0;
try {  // Manipulate input arguments to add
       // workspace specific settings
       args = EbayStartupHook.beforeMain(args);
       if (args == EbayStartupHook.QUIT) {
              return 0;
       }
       // EBAY STARTUP HOOK END
       basicRun(args);
       ...
```

Eclipse startup ~~hack~~ hook 4

```
com.ebay.tools.startup.EbayStartupHook

public class EbayStartupHook{
...
public static String[] beforeMain(String[] input) {
...
// Given arguments -data {workspace},
// add -configuration {workspace}/ebay-local-configuration.
...
```

Eclipse startup ~~hack~~ hook 5

- **Configuration (-configuration) {osgi.configuration.area} [@none, @noDefault, @user.home, @user.dir, filepath, url]**
  - Configuration locations contain files which identify and manage the (sub)set of an install to run. As such, there may be many configurations per install. Installs may come with a default configuration area but typical startup scenarios involve the runtime attempting to find a more writable configuration location.
- **Instance (-data) {osgi.instance.area} [@none, @noDefault, @user.home, @user.dir, filepath, url]**
  - Instance locations contain user-defined data artifacts. For example, the Resources plug-in uses the instance area as the workspace location and thus the default home for projects. Other plugins are free to write whatever files they like in this location.

http://help.eclipse.org/ganymede/index.jsp?topic=/org.eclipse.platform.doc.isv/reference/misc/runtime-options.html

http://help.eclipse.org/galileo/index.jsp?topic=/org.eclipse.platform.doc.isv/reference/misc/runtime-options.html

Usage tracking

Expect the unexpected; anecdotes from the “the real world”:

- **Weather balloons strapped to lawn-chair**
  - Solution: fine & disclaimers
- **Storing coins in an ashtray**
  - Solution: alternate storage and supervision
- **…**

Usage tracking

Unexpected usage "in the software world":

- **Web Service query meant to retrieve daily changes called in a 1 second polling loop.**
  - Solution: creating a call-back mechanism

- **Automated build used to create & check-in thousands of jars for a release used by developer on daily basis causing capacity issues**
  - Solution: authorization/authentication & user education

- **Infrequent use of little known features; re-creation of same functionality in multiple tools**
  - Solution: tracking, user education, dedicated tools team

Usage tracking

Knowing who is using what/when/how is important!

- **With software tracking is relatively easily**
- **Light-weight low-overhead trumps ultimate accuracy**

- **Combine usage, performance, coarse grained logging**
  - Central: Log Events (errors, exceptions, usage), Log Transaction, Heartbeat
  - Local: verbose outputs too expensive to send over the network or not interesting until a specific bug is getting researched with the user.

- **Track centrally if you can, locally where you must**

Usage tracking

Combined approach allows you to answer many important questions:

- **Who uses what functionality, how often and when?**
- **Why should management spend resources on tooling?**
- **Where do you need to more user training / education?**
- **What is the performance?**
- **What errors occur in the users' environment?**
- **Which versions of the plug-ins / tools are used?**
- **When a user does file a bug, what exactly happened?**

Maintenance

- **Maintenance is required to adapt to changing environment**
  - Stale tools get abandoned or waste users' time
- **Release effort depends on deployment options chosen**
- **Separate production bug fixes from development stream**
- **Usage tracking / central logging lets you make informed decisions about bug fixes and new feature development**
  - Don't just give users what they ask for, figure out what they really need

Documentation

- **Proper documentation is first line of defense in battle to keep your users informed of functionality**

- **Keep it up to date**
  - Much outdated documentation can be worse than less documentation which is actually accurate

- **Use the various mechanisms to provide documentation:**
  - Micro-help / mouse-over bubbles
  - Context sensitive help
  - Eclipse Help shipped & and standalone on web-site

- **Do not forget release notes**

Support and user education

Support and user education is critical. Use every option available.

Self-help:

- **FAQ**
- **Forum**
- **Wiki - beware of stale information**
- **(Recorded) training brown-bag training session**
- **(Recorded) how-to’s**

Support and user education

- **See every interaction with user as an opportunity**
  - Follow up
- **Personal / boots on the ground approach**
- **Responsiveness to “squeaky wheel”**
- **Turnaround time of bugs and (small) enhancements**
  - Perception is reality
- **Public tracking of new feature requests**

Conclusion

If you want your tools to be successful in the enterprise, it takes much more effort than coding a few cool Eclipse plug-ins.

This requires attention to many details of various other aspects that drive adoption as discussed in this presentation.

Questions

Deploying Successful Enterprise Tools

Joep Rottinghuis, eBay