Technical Debt in Eclipse Development

Eclipse Con France 2018

13 June 2018
Table of content

1 - Technical debt in your RCP application 5
Technical debt in your RCP application

Who?
Olivier Prouvost:
- OPCoach (www.opcoach.com)
- Eclipse expert since 2004
- Provides training and consulting on Eclipse technologies
- France / Europe ... or even further
- Committer on e4 tools and platform UI
- olivier@opcoach.com
- Twitter: @OPCoach_Eclipse

Why this talk?
During several consulting mission
I have seen so many debt symptoms!

Audience
This talk is for:
- Eclipse 3.X developers that want to reduce their debt.
  - Running with an old 3.X target platform
  - Running with a 4.X target platform but still using 3.X mechanisms
- Eclipse 4.X developers that want to have more information about architecture and tooling.

Technical debt?
In one word:
- This is all the tasks ignored or delayed today that will cost later.

Why is the technical debt settling?
- Because it has a visible cost and an unvisible return of invest.
General symptoms

➢ lack of tests (usually none !!)
➢ reinventing the wheel ... but with new bugs!
➢ no continuous integration
➢ bad outsourcing (to reduce the costs but actually it costs more)
➢ no source code analysis
➢ people not skilled
➢ no real management (lack of measures, controls and procedures)

Eclipse development symptoms

➢ deprecated usages still used
➢ bad usage of extensions points
➢ internal packages used
➢ still using jdk 7
➢ bad architecture (ui and core mixed, too many plugins / not enough plugins)
➢ developers know Java... all right they can do RCP development !

And the three main symptoms explained in this talk :

➢ bad target platform (still a 3.X !).
➢ developing using 3.X concepts
➢ lack of UI Tests

Reasons to change

The disadvantages of Eclipse 3.x :

➢ Eclipse 3.0 code delivered 14 years ago (2004) and Juno (3.8 - 2012)
➢ Many UI extension points and extensions all over the framework
➢ Strong dependency to the framework (inheritance, dependencies ...)
➢ API are not uniform
➢ Many singletons and no OSGi services
➢ No Injection
➢ No separation between content and appearance (no renderer)
➢ Difficulty to style application

Advantages of Eclipse 4 runtime

➢ Application model is UI agnostic (SWT, Java FX...) thanks to POJOs
➢ Injection reduces the amount of code and simplifies dramatically testing
➢ Eclipse 4 event notifier is very concise and easy to use with injection
➢ The UI customization is easier thanks to :
  ➢ CSS
  ➢ renderers that can be overridden

Bugs fixed

Since the 3.8.2 version a lot of bugs have been fixed !
Just on platform.ui, platform.runtime and SWT:

<table>
<thead>
<tr>
<th>Version</th>
<th>Start</th>
<th>End</th>
<th>platform.ui</th>
<th>platform.runtime</th>
<th>SWT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Juno</td>
<td>30/06/12</td>
<td>30/06/13</td>
<td>393</td>
<td>26</td>
<td>324</td>
</tr>
<tr>
<td>Kepler</td>
<td>30/06/13</td>
<td>30/06/14</td>
<td>698</td>
<td>43</td>
<td>194</td>
</tr>
<tr>
<td>Luna</td>
<td>30/06/14</td>
<td>30/06/15</td>
<td>644</td>
<td>45</td>
<td>259</td>
</tr>
<tr>
<td>Mars</td>
<td>30/06/15</td>
<td>30/06/16</td>
<td>675</td>
<td>86</td>
<td>332</td>
</tr>
<tr>
<td>Neon</td>
<td>30/06/16</td>
<td>30/06/17</td>
<td>545</td>
<td>48</td>
<td>453</td>
</tr>
<tr>
<td>Oxygen</td>
<td>30/06/17</td>
<td>30/06/18</td>
<td>415</td>
<td>38</td>
<td>428</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td><strong>3370</strong></td>
<td><strong>286</strong></td>
<td><strong>1990</strong></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>5646</strong></td>
</tr>
</tbody>
</table>

**The application model**

- It describes the UI hierarchy of the application and other components
- Its API is an EMF API

*Image 1 Application Model

**Fragments, processors and migration**
Fragments and processors are the key concept for your development:

**Warning:** using the compatibility mode the legacy model is neither not editable nor visible

### E4 Tooling

There are 2 major tools to use
- the E4 spies
- the fragment model editor

#### The E4 model spy
- Once you are running using a 4.X runtime
- Whatever your development (3.X or 4.X way)
  - install the E4 spies
  - pick them up from the Eclipse Market Place
- You can look at the live model using the model Spy (Alt Shift F9)
The application fragment Editor

The fragment editor is used to edit some parts of the model.
It is a major tool for your E4 development.
Reducing the debt

The three points to fixed:
- bad target platform (still a 3.X !).
- developing using 3.X concepts
- lack of UI Tests

Minimize your debt : manage your TP properly

This is a major issue
- Never use less than the N-1 version
- Use the right tooling to edit and manage it
- Use the same TP for your development and for the build!

Target platform : specific tooling

First of all, you must install the famous target platform editor written by Mikaël Barbero :
- see : https://github.com/mbarbero/fr.obeo.releng.targetplatform

1 - https://github.com/mbarbero/fr.obeo.releng.targetplatform
Edit a .tpd file that will generate the .target file:

Target Platform Editor

**Target platform**: organize your project

To manage it correctly, you must:

- create a specific project named `com.xxx.yyy.tp`
- create the associated tpd file named: `com.xxx.yyy.tp.tpd`
- let the tooling generate the .target file in the same directory
- add a pom file with `eclipse-target-definition` packaging
- refer to the target project in your parent pom
- refer to the target file in each IDE.
Example of a minimal tpd file for a 3.X running application (using compatibility):

```plaintext
com.xxx.yyy.tpd
1
2  target "Minimal RCP 3 target Platform" with source requirements
3
4  location "http://download.eclipse.org/releases/photon" {
5    org.eclipse.rcp.feature.group
6    org.eclipse.ui.workbench
7    org.eclipse.jdt.junit4.runtime
8    org.eclipse.equinox.executable.feature.group
9  }
10
11 // Delta pack for other platforms
12  location "http://download.eclipse.org/eclipse/updates/4.8" {
13    org.eclipse.equinox.sdk.feature.group
14  }
```

### 3.X and 4.X runtime compatibility

If you haven’t
- used internal packages
- modified the presentation engine

Your 3.X application should run with the 4.X runtime.

**Advantages:**
- it will be easier to debug (model spy and other spies)
- your application should start quicker
- you can start to develop new plugins using model fragments
- you will be ready to update the target platform each year

### Using the E4 concepts

- The model fragment is a way to declare the UI elements of your application
- It will bind POJO objects to the model
- POJO’s tests are really easier to write (no dependencies on framework)

### Defining your views

- If your view is not in a perspective and can be opened later
- Use a model fragment with : xpath:/ and the descriptors feature
- Define your view using a PartDescriptor
- Add your menus and/or specific handlers
To see you view in the menu Window->Show view, add specific tags:

- Defining your perspective
  - May be you have a plugin that defines a perspective
  - It will directly contain the parts and their definition.
  - Use a snippet in the model fragment using xpath:/ and snippets feature
  - Add the perspective inside and describe it
  - You can mix either pure E4 views or legacy E3 views using:
    - `class` URI:
    - `ID`: ID of the view in E3
  - To display the perspective, use the `EPartService.switchPerspective`
Defining your handlers

To define additional handlers on common commands like copy:

- Define the command:
  - Either import the command in the fragment using its E3's ID
  - Or define it in the fragment using `xpath:` with the `commands` feature

- Bind your local handler to this command
  - Add the handler in the fragment:
    - `with xpath:` and the `handlers` feature
    - in the Part’s handlers category
    - in the Perspective’s handlers category

- The handler is a POJO class that can be easily tested
Defining your menu contribution

- Sometimes it is not possible to add a command in a Part or in a menu because:
  - it is not available in the fragment
  - it is contributed by another plugin
  - it comes from the legacy model

- Use the `menuContribution` or the `toolbarContribution` features
You can easily get this sample:

- `git clone git://git.eclipse.org/gitroot/e4/org.eclipse.e4.tools.git`
- `open: bundles/org.eclipse.e4.tools.spy/fragmente4.xmi`

**Testing your E4 POJOs**

E4 POJOs need injection and a proper initialized context. OPCoach has developed a specific E4 test case that initializes this context.

The test case provides several convenient methods like:
- creating a Part using the context
- creating a Handler using the context
- setting the current selection
- checking if a text field in a POJO contains the right value
- checking if a tree is expanded with the correct nodes

**Status of the project**

- This is still an experimental project but it works well
- It is available on:
  - [https://github.com/opcoach/E4Tester](https://github.com/opcoach/E4Tester)
- It is delivered under EPL license with this URL:
The E4 TestCase

- The `E4TestCase` abstract class must be extended to define your test.
- It provides these basic methods:

```java
44 // Add the new E4 test platform
45 @location "https://dl.bintray.com/opcoach/E4Tester/" {
46   com.opcoach.e4tester.feature.feature.group
47 }
```

**Defining your E4 POJO Tests**

Define your test in a plugin fragment
Then extends the `E4Testcase`, create your part(s) and check the POJO's fields
Don't forget to evaluate this talk!
Evaluate the Sessions

Sign in and vote at eclipsecon.org

-1 0 +1