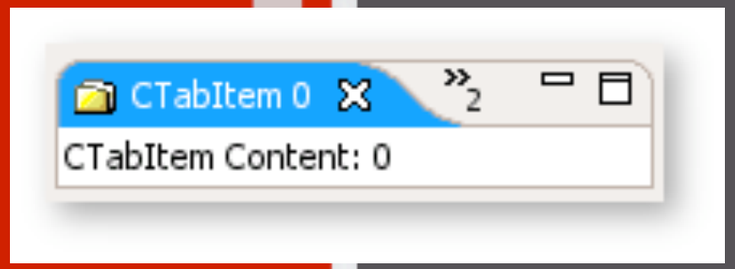
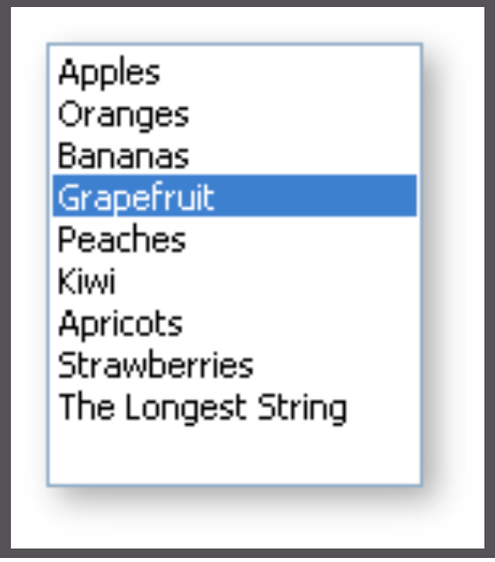
EclipseCon 2009

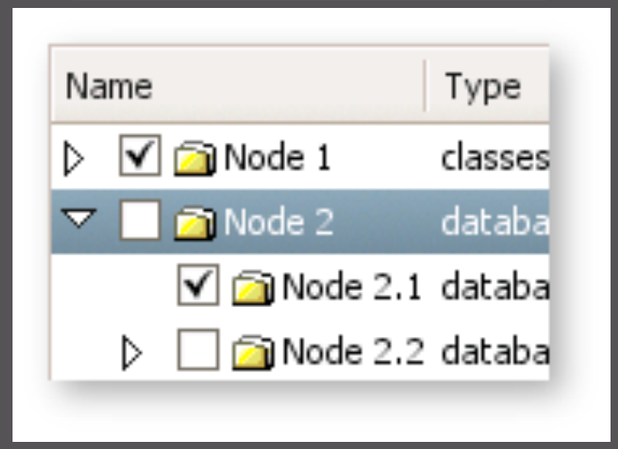
UI Test Automation with SWTBot

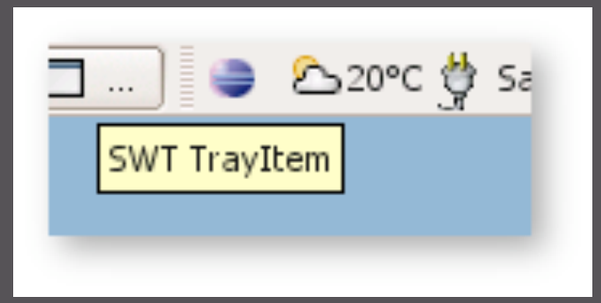
Ketan Padegaonkar, Code Monkey

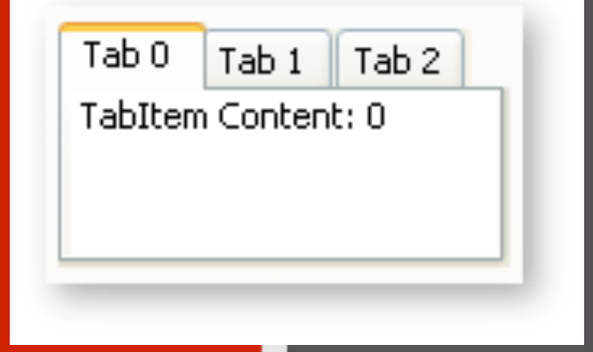
ThoughtWorks Studios.

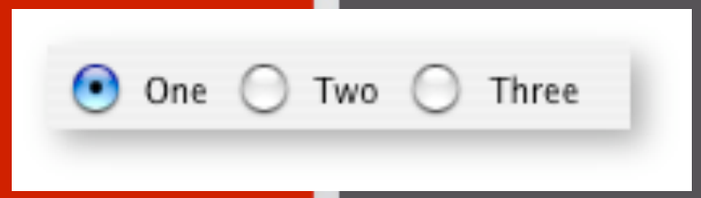


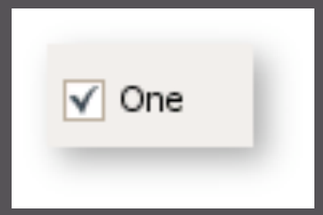


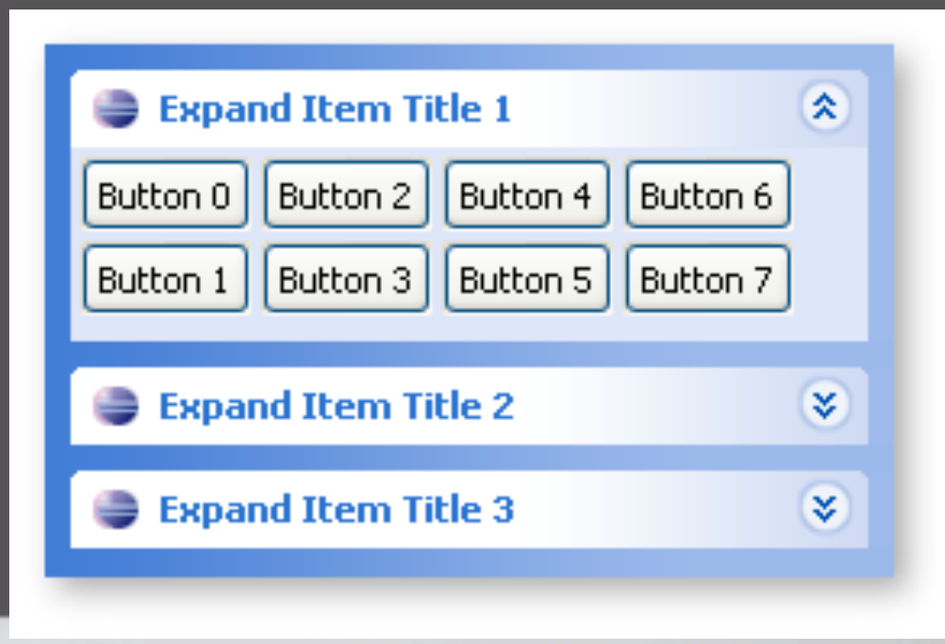


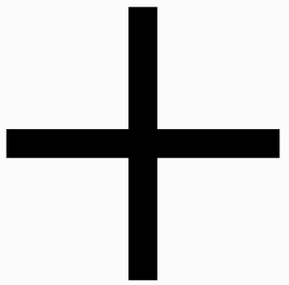




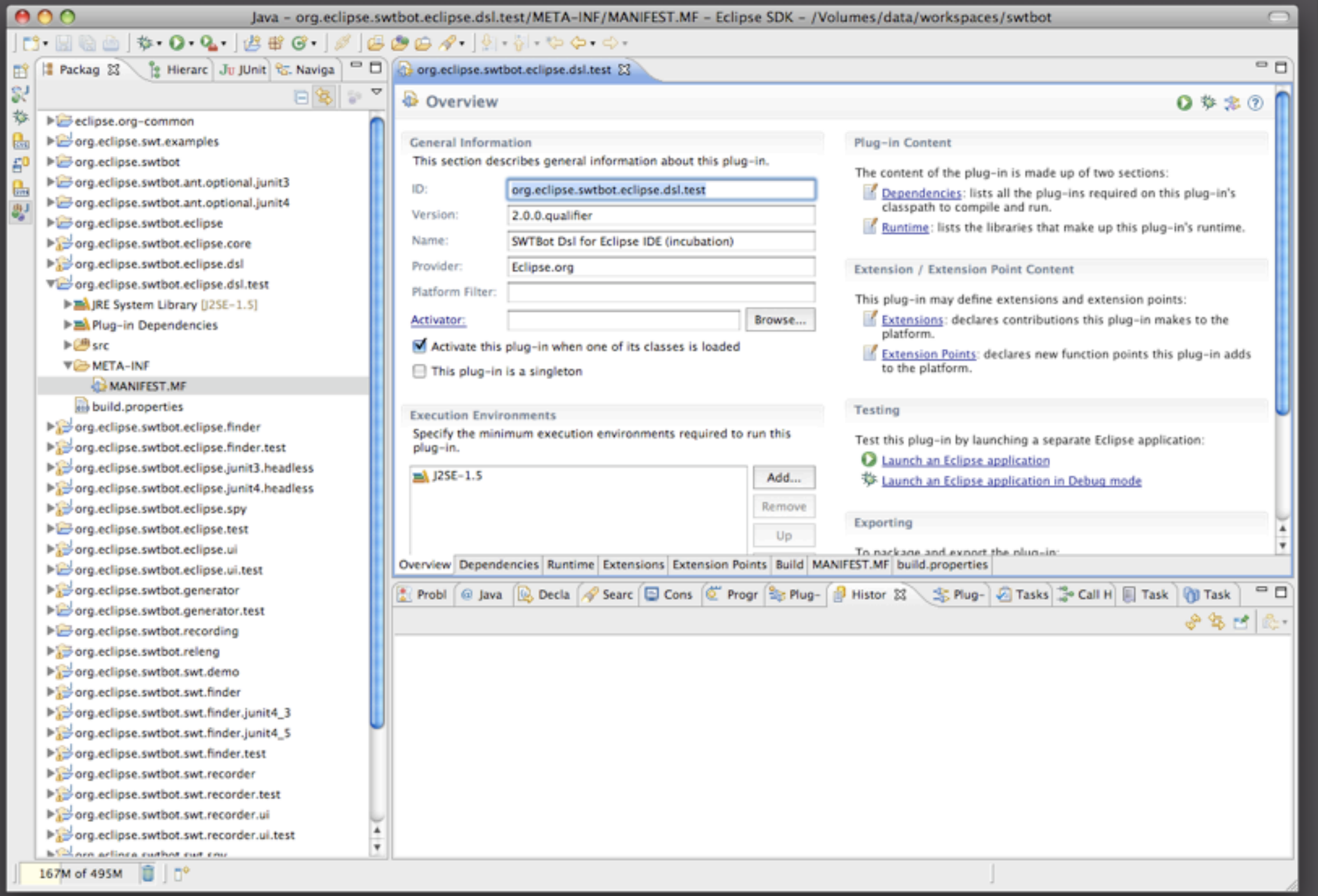




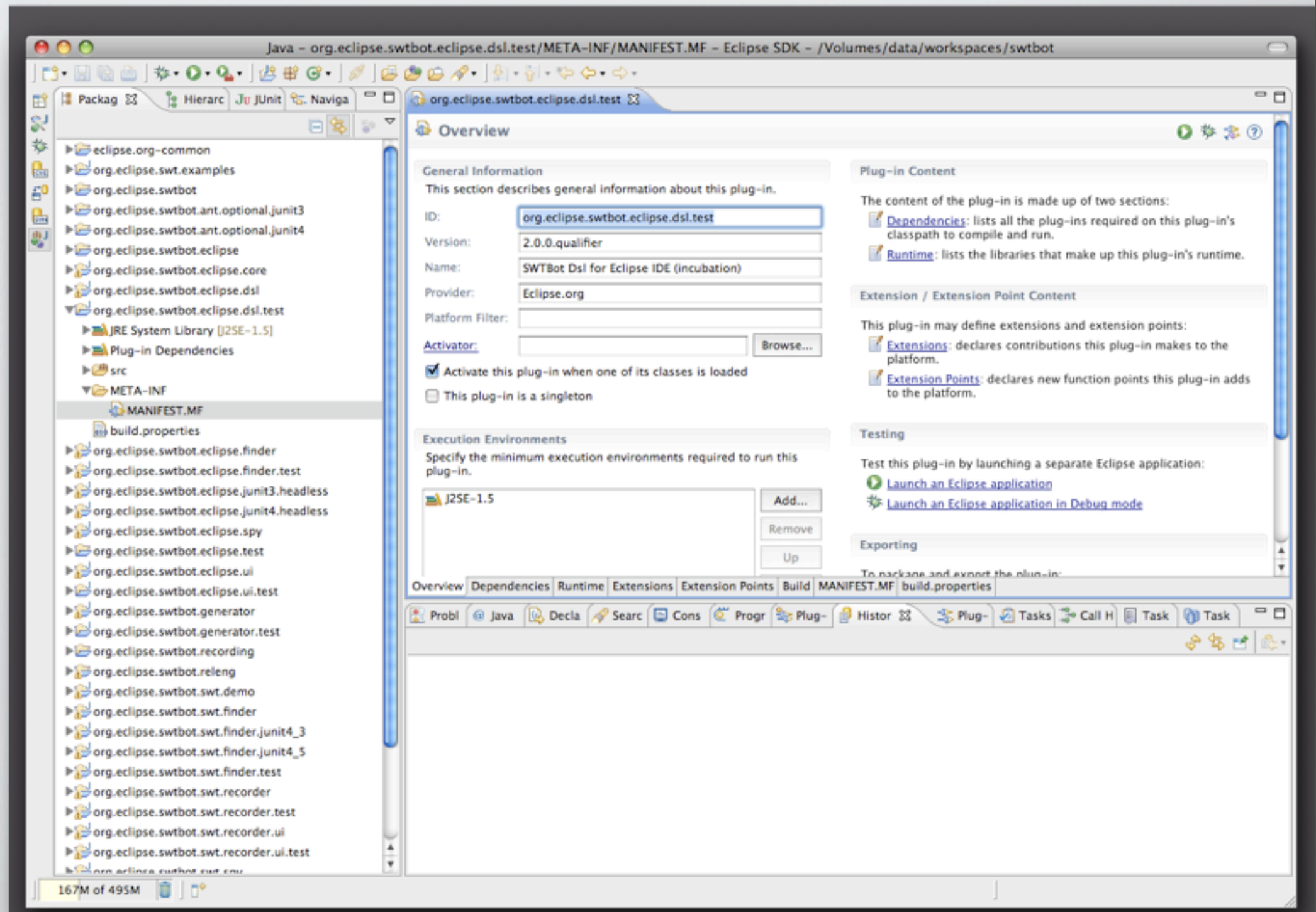


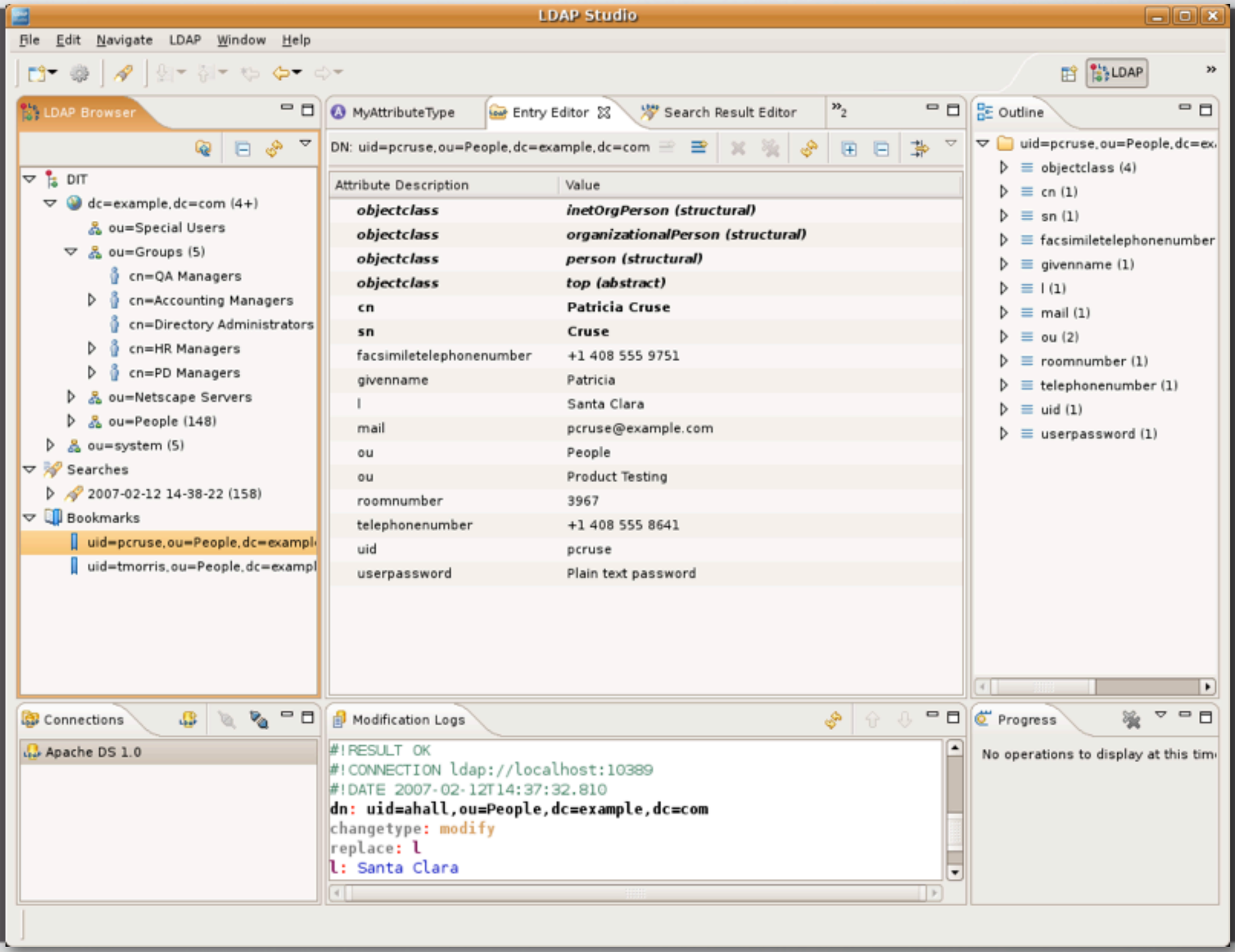


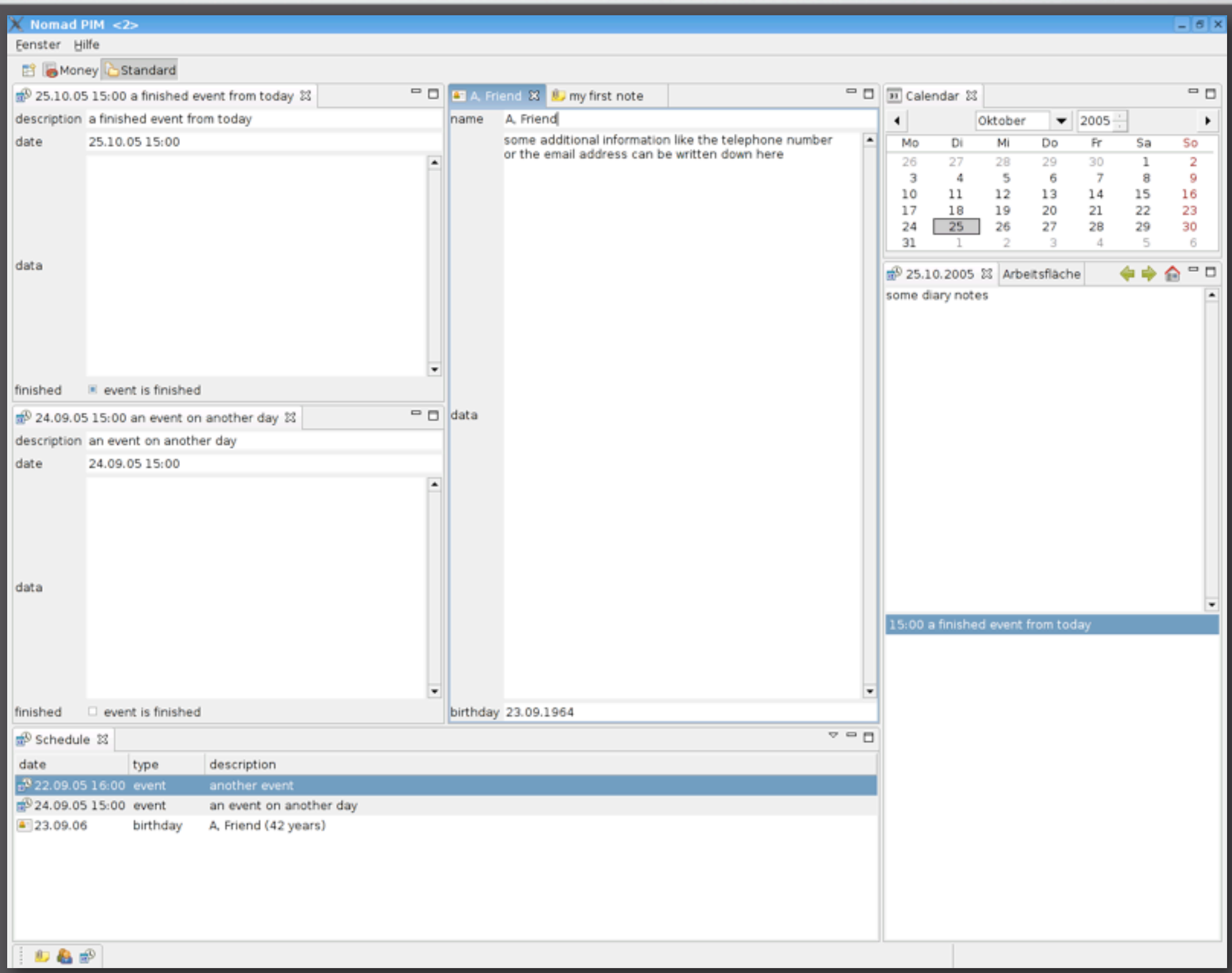
EclipseCon 2009

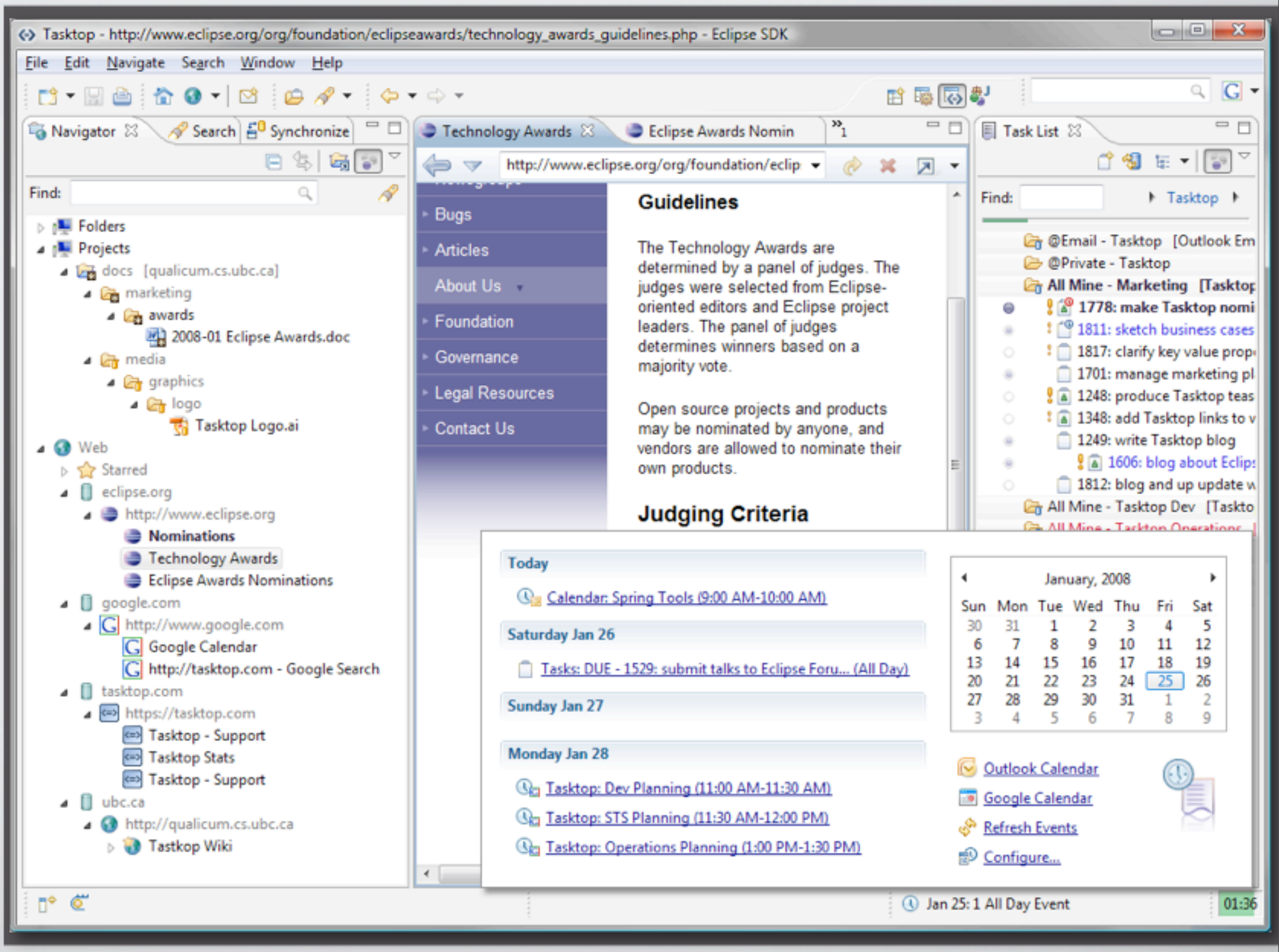


EclipseCon 2009









Challenges

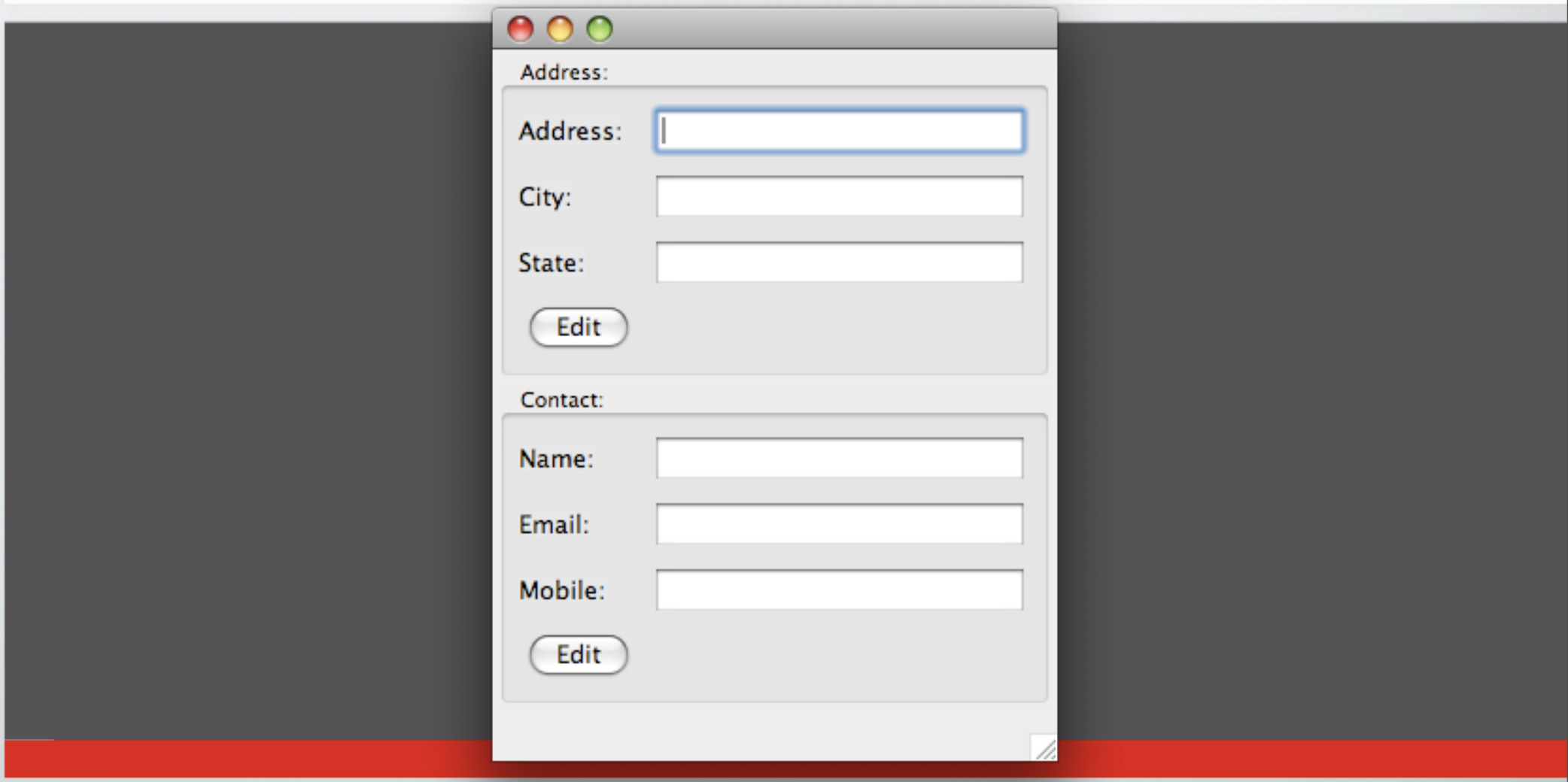
Challenges

Identifying Controls

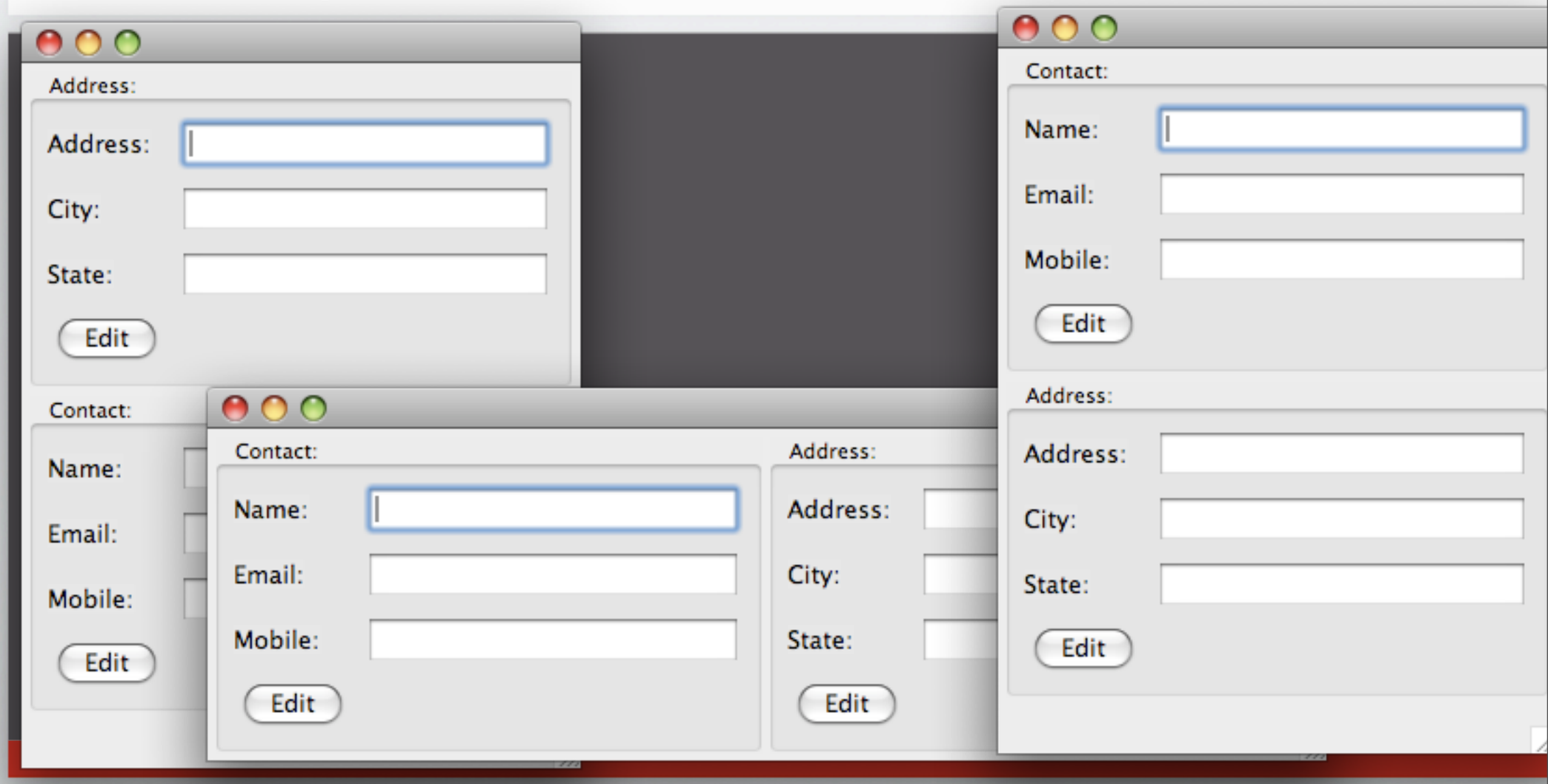
Challenges

Similar looking controls

Similar looking controls



Moving controls



Challenges

Sending “events” to controls

Challenges

Manage SWT Threading

Challenges

Tests to be non-blocking

Challenges

Run in a separate thread

Challenges

Run in a separate thread

Still manage synchronization between threads

Challenges

Multi threaded applications, background jobs

Challenges

Multi threaded applications, background jobs

“Non-deterministic” in amount of time required

Challenges

Internationalization (i18n) and Localization (L10n)

Challenges

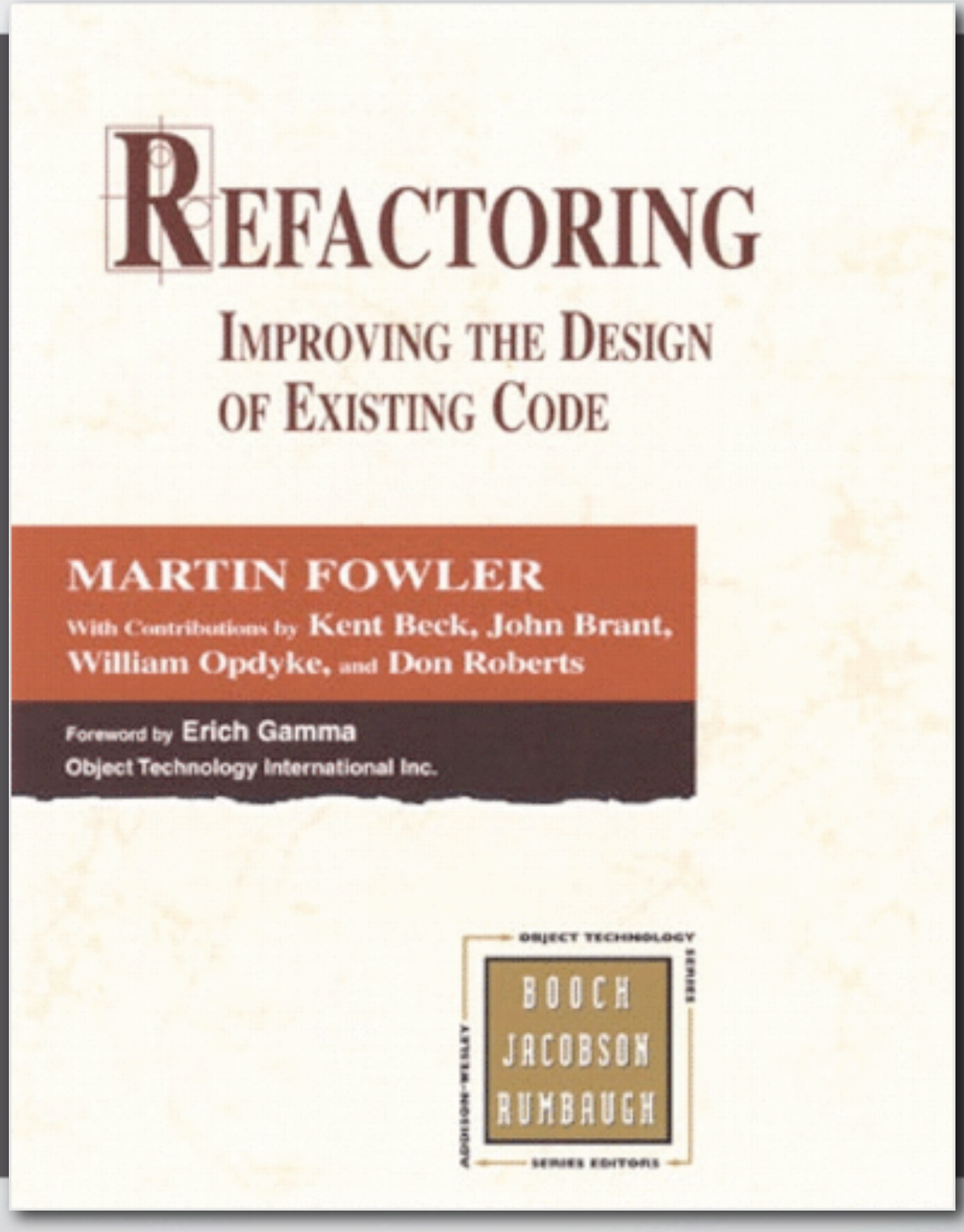
Readability

Challenges

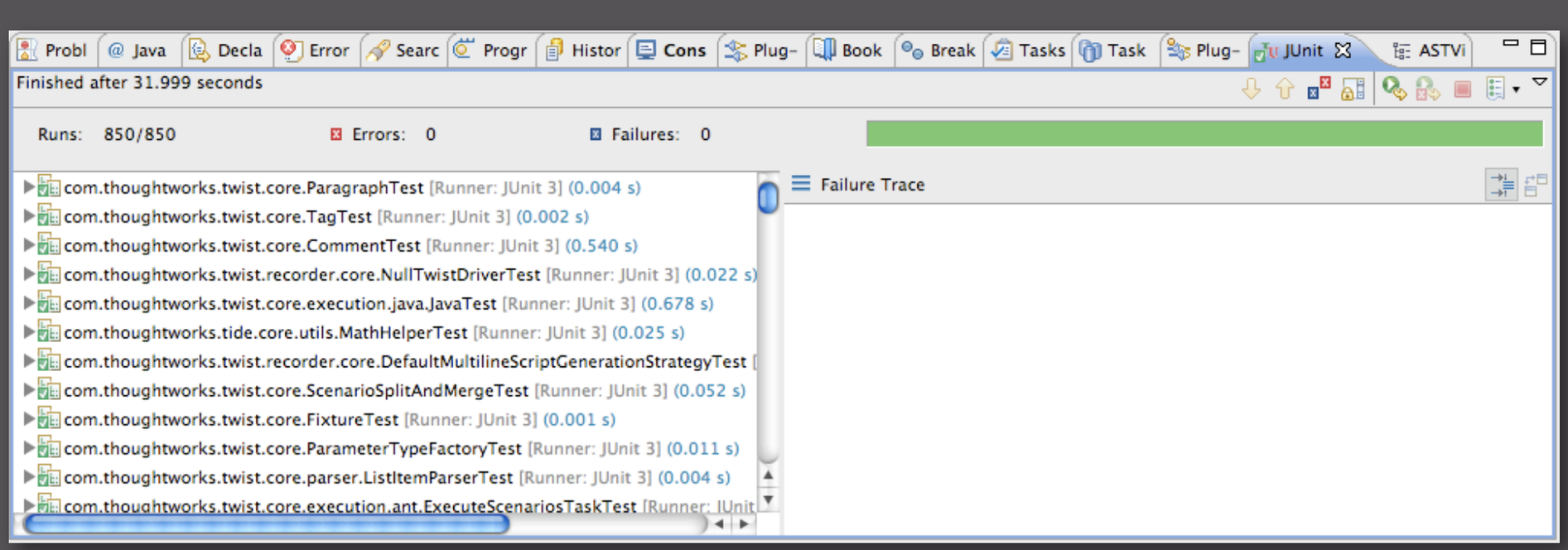
- Identifying controls
- Similar looking controls
- Moving controls
- Sending “events to controls”
- Manage SWT Threading
- Tests to be non-blocking
- Run in a separate thread, and manage synchronization
- Multi threaded applications, background jobs
- Internationalization (i18n) and Localization (L10n)
- Readability

Testing ?

Unit Testing

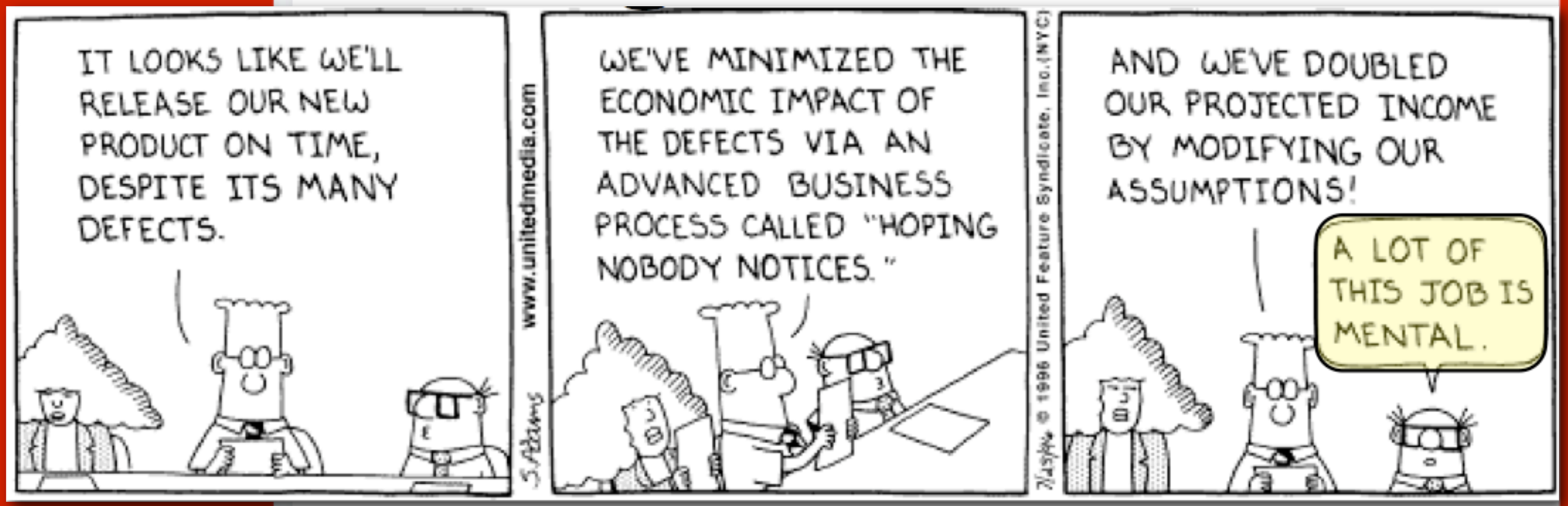


"red green refactor"



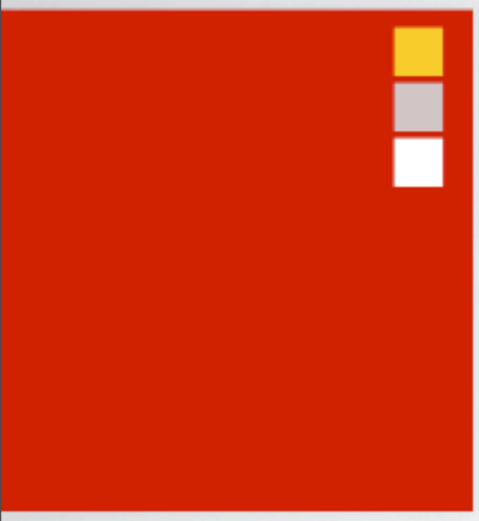
Functional Tests

Writing SWT Tests



Understand SWT

Understand Threading



Quality Analysts/Testers

Quality Analysts/Testers

SWTBot

Agenda

- setting up the environment
- basic SWTBot API
  - custom assertions
  - analyze failures
- how does it work?
  - handling background jobs and long running operations
  - thread safety
  - improve performance
- FluentAPI for common eclipse operations (DSL-ish)

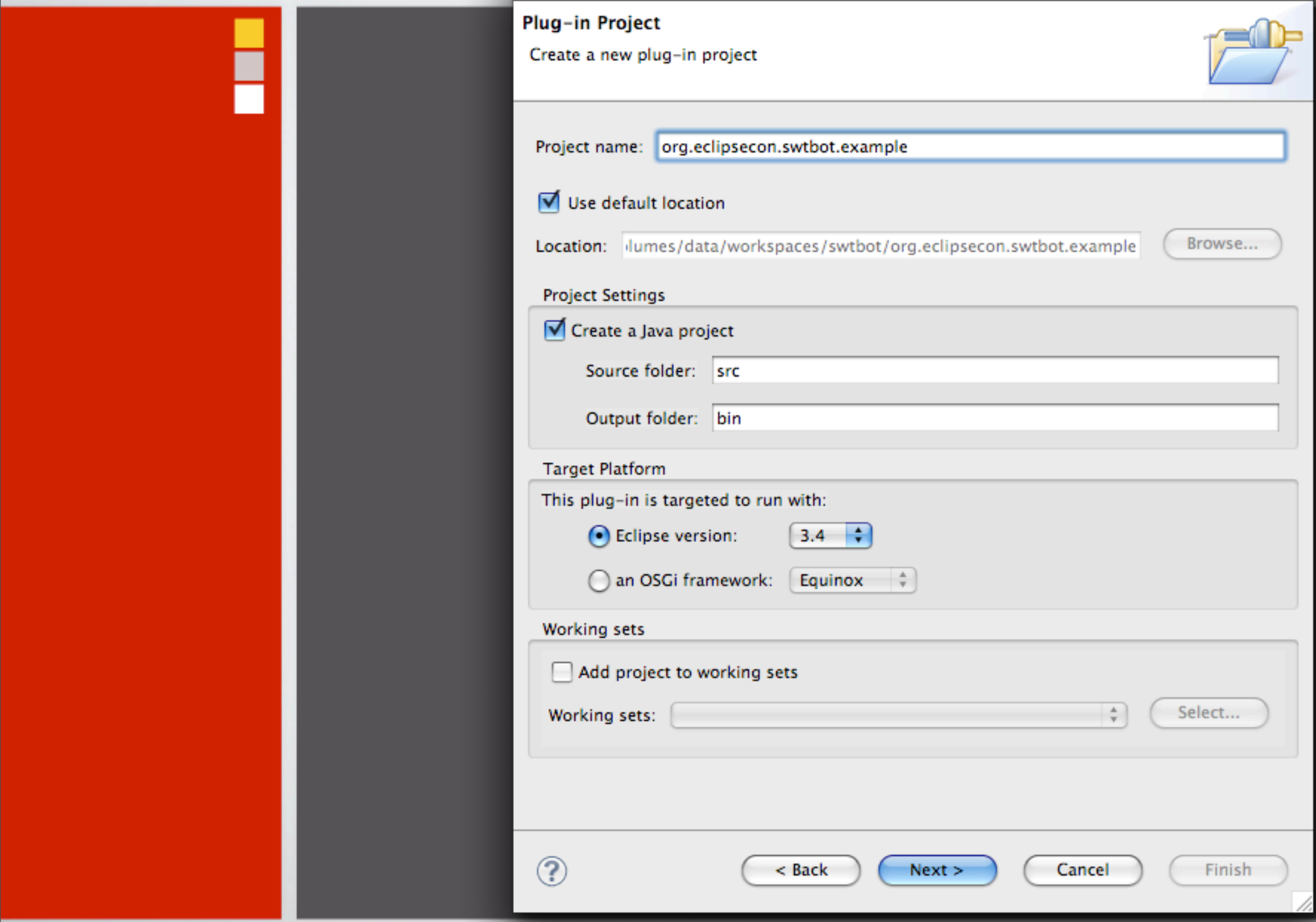
Setting up the Environment

Setting up the Environment

- Eclipse 3.4
- SWTBot update site
  - http://www.eclipse.org/swtbot/downloads.php

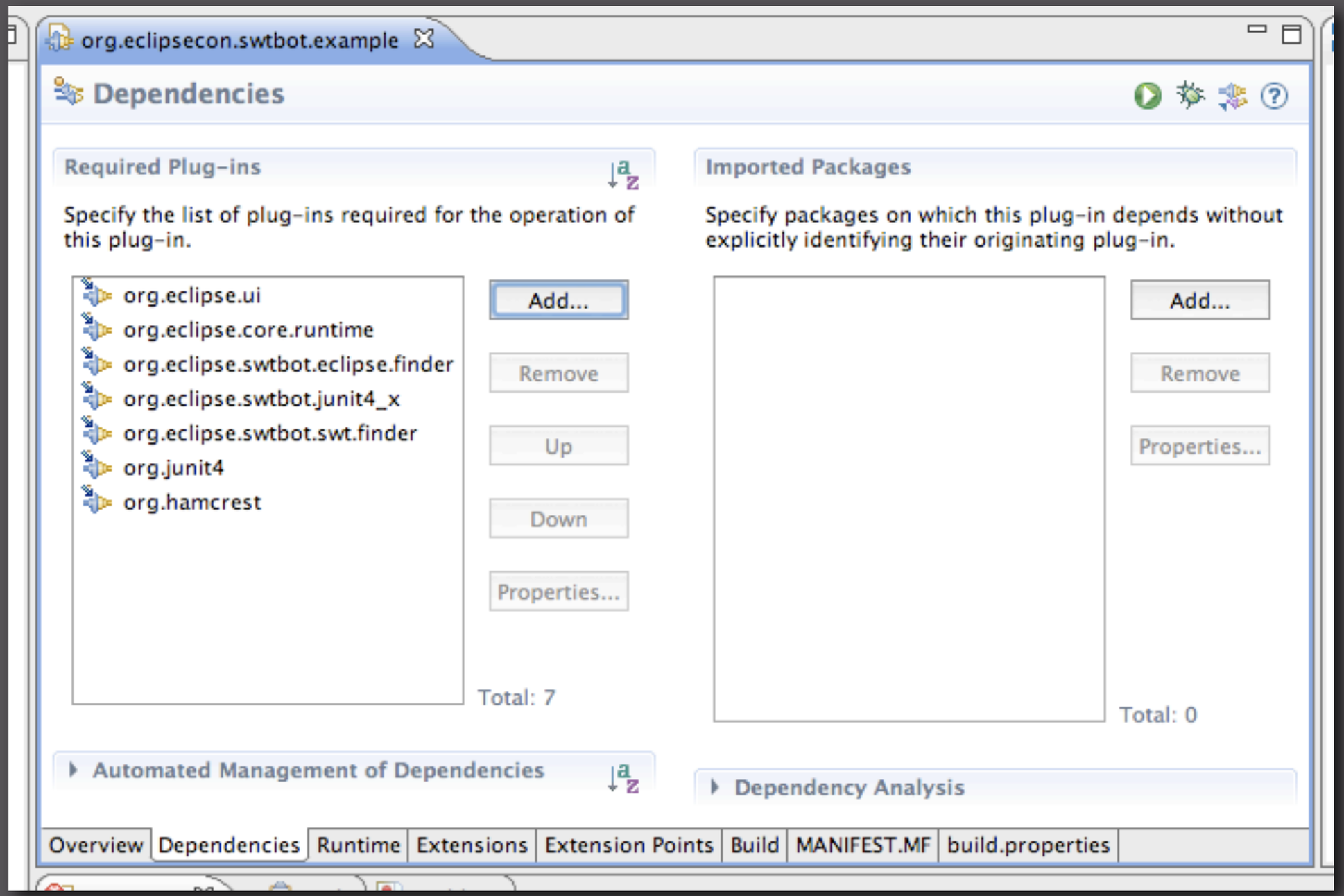
Create a plugin project

- “org.eclipsecon.swtbot.example”



Setup Dependencies

- org.eclipse.ui
- org.eclipse.core.runtime
- org.eclipse.swtbot.eclipse.finder
- org.eclipse.swtbot.junit4_x
- org.eclipse.swtbot.swt.finder
- org.junit4
- org.hamcrest



Basic SWTBot API

The first red and green bar

A “hello world” test!

Setup for the test

- close the “Welcome Page”

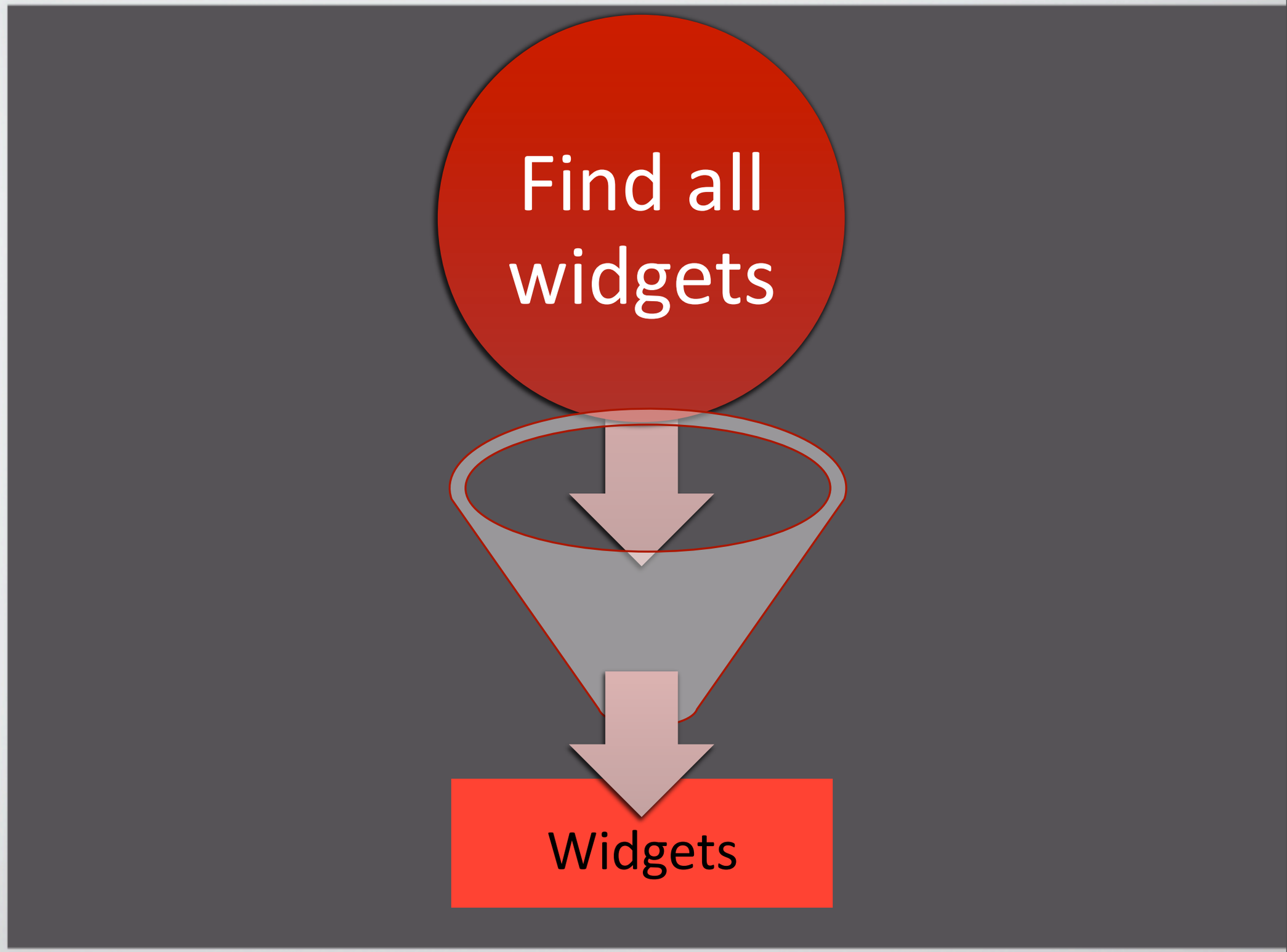
Create tests for

- creating a java project “MyFirstProject”
- creating a java class “org.eclipsecon.project.HelloWorld”
- type in a program that prints “Hello, World”
- execute the program
- verify that the program printed “Hello, World”

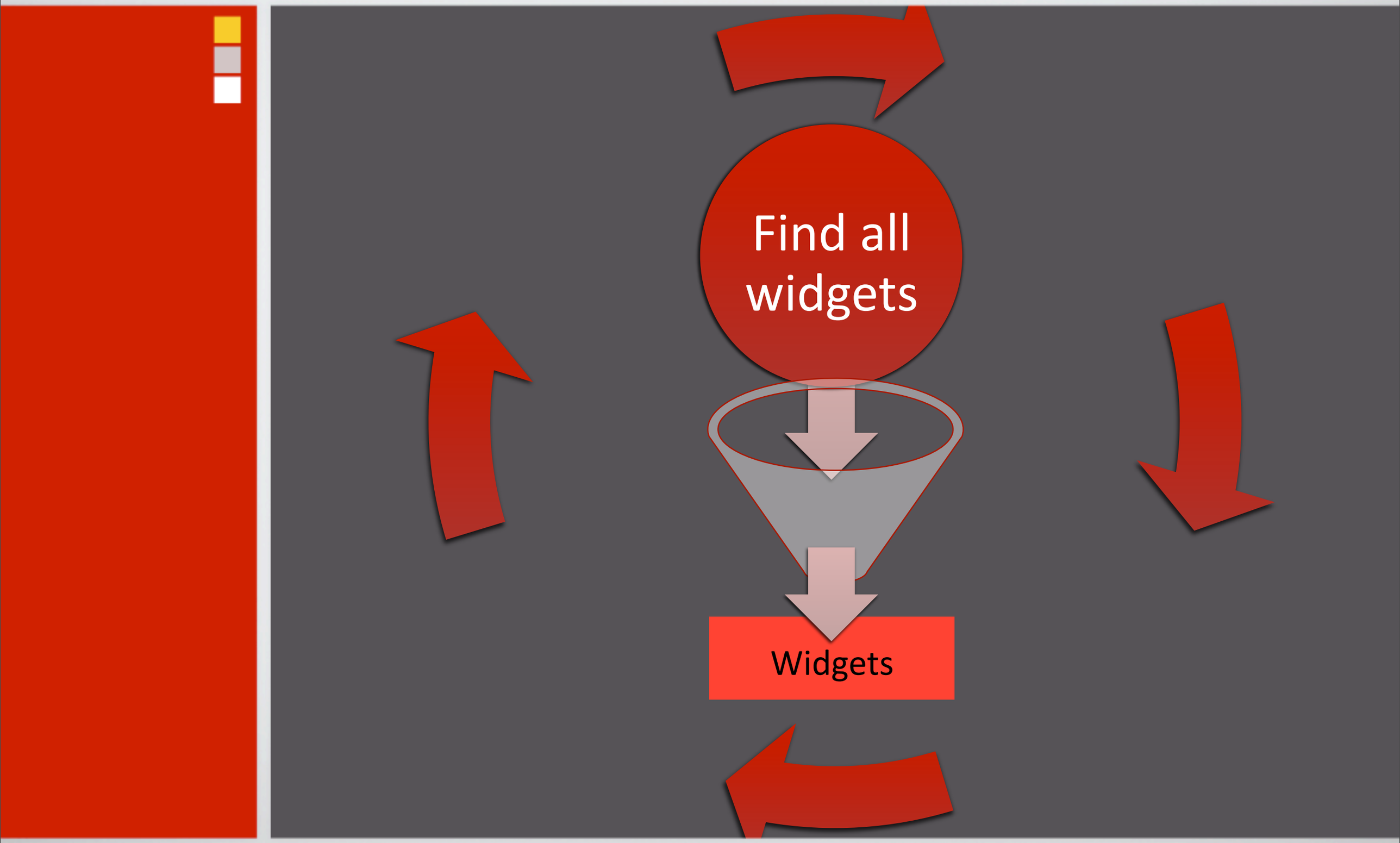
Teardown for the test

- delete the project

How does it work?



Redundancy and Failure Proofing



Find all widgets

- Depth first traversal of UI elements

1. Find top level widgets
    1. Find children of each widget
2. For each child do (1) and (2)

Creating matchers(simple)

- `withText("Finish")`
- `withLabel("Username:")`
- `withRegex("Proceed to step (.*)")`
- `widgetOfType(Button.class)`
- `withStyle(SWT.ARROW, "SWT.ARROW")`

...Creating matchers(combination)

- `allOf(matchers...)`
- `anyOf(matchers...)`
- `not(matcher)`
- `allOf(anyOf(matchers...), matchers...)`

Handling long running operations

- describe a condition
- poll for the condition at intervals
- wait for it to evaluate to `true` or `false`
- of course there's a timeout

Handling Waits(SWTBot.java)

```
private void waitUntil(ICondition condition, long timeout, long interval) {

    long limit = System.currentTimeMillis() + timeout;
    condition.init((SWTBot) this);
    while (true) {
        try {
            if (condition.test())
                return;
        } catch (Throwable e) {
            // do nothing
        }
        sleep(interval);
        if (System.currentTimeMillis() > limit)
            throw new TimeoutException("Timeout after: " + timeout);
    }
}
```

The whole thing put together

The End User API

Finding widgets (SWTBot.java)

```
public SWTBotTree treeWithLabelInGroup(String l, String g, int i) {

      // create the matcher
      Matcher matcher =
            allOf(
                  widgetOfType(Tree.class), withLabel(l), inGroup(g)
            );
      // find the widget, with redundancy built in
      Tree tree = (Tree) widget(matcher, index);
      // create a wrapper for thread safety
      // and convinience APIs
      return new SWTBotTree(tree, matcher);
}
```

Thread Safety

- Tests should run in non-ui thread
- query state of a widget
- change state of a widget

Thread Safety (Query state)

```
public class SWTBotCheckBox {
public boolean isChecked() {
   // evaluate a result on the UI thread
   return syncExec(new BoolResult() {
      public Boolean run() {
         return widget.getSelection();
      }
   });
   }
}
```

Thread Safety(change state)

```
public class SWTBotCheckBox {
    public void select() {
        asyncExec(new VoidResult() {
            public void run() {
                widget.setSelection(true);
            }
        });
        notifyListeners();
    }

    protected void notifyListeners() {
        notify(SWT.MouseDown);
        notify(SWT.MouseUp);
        notify(SWT.Selection);
    }
}
```

Building Abstractions

Features and capabilities of tests (Page Objects)

- Project Explorer
- The Editor
- The Console View
- The main menu bar, tool bar

Features and capabilities of tests (Domain Objects)

- Create a project
- Delete a project
- Create a class
- Execute a class
- more...

Page Objects

http://code.google.com/p/webdriver/wiki/PageObjects

Page Objects... should

- Represent the services offered by the page to the test developer
- Internally knows the details about how these services are offered and the details of UI elements that offer them
- Return other page objects to model the user's journey through the application
- Different results of the same operation modeled differently

Page Objects... should not

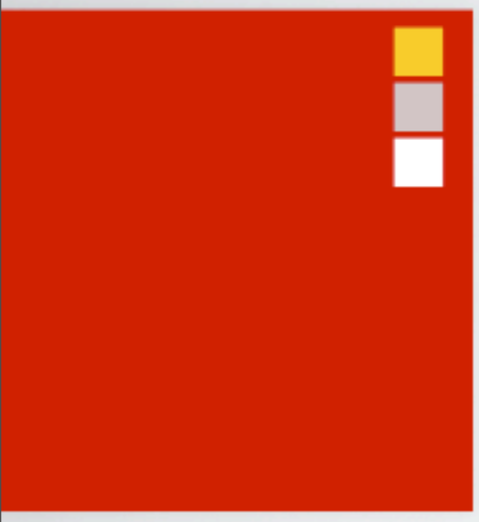
- Expose details about user interface elements
- Make assertions about the state of the UI

Page Objects (implementation)

```
public class LoginPage {

public HomePage loginAs(String user, String pass) {
    // ... clever magic happens here
}

    public LoginPage loginAsExpectingError(String user, String pass) {
        // ... failed login here, maybe because one or both of
        // username and password are wrong
    }

    public String getErrorMessage() {
        // So we can verify that the correct error is shown
    }
}
```

Page Objects (usage)

```
// the bad test
public void testMessagesAreReadOrUnread() {
    Inbox inbox = new Inbox(driver);
    inbox.assertMessageWithSubjectIsUnread("I like cheese");
    inbox.assertMessageWithSubjectIsNotUndread("I'm not fond of tofu");
}

// the good test
public void testMessagesAreReadOrUnread() {
    Inbox inbox = new Inbox(driver);
    assertTrue(inbox.isMessageWithSubjectIsUnread("I like cheese"));
    assertFalse(inbox.isMessageWithSubjectIsUnread("I'm not fond of tofu"));
}
```



```
   LoginPage login = new LoginPage();
HomePage home = login.loginAs("username", "secret");
SearchPage search = home.searchFor("swtbot");
assertTrue(search.containsResult("http://eclipse.org/swtbot"));
```

Exercise: Page Objects

- Refactor the tests in the form of a PageObject

Domain Objects

Domain Objects... should

- Represent the operations that can be performed on concepts

Domain Objects

```
public class JavaProject {
   public JavaProject create(String projectName){
      // create a project and return it
   }
   public JavaProject delete(){
      // delete the project and return it
   }
   public JavaClass createClass(String className){
      // create a class and return it
   }
}
```


Going ahead

- Commonly used functionality bundled as convenience API
- Eclipse Forms
- GEF!
- Use “real” events instead of “mocks”

Questions ?

newsgroup: news://news.eclipse.org/eclipse.swtbot

web: eclipse.org/swtbot

gmail: KetanPadegaonkar

Resources

- http://flickr.com/photos/stuart100/288880576/
- http://flickr.com/photos/60373916@N00/229233928/
- http://www.flickr.com/photos/jfravel/1001472806/
- http://www.flickr.com/photos/54323936@N00/245650981/
- http://www.flickr.com/photos/aknacer/3196381450/
- http://www.flickr.com/photos/buttersweet/33684613/
- http://www.flickr.com/photos/pintuck/283795079/
- http://www.flickr.com/photos/o3bor/209925927/
- http://www.flickr.com/photos/leoniewise/3369871669/