



Test-Driven Development for Portal Applications

Mark Vardy, Graham Harper
Application Architects
IBM Software Services for Lotus

Lotus software



Agenda

- **Introductions**
- **Benefits and principles of test-driven development**
- **Example application**
- **How do we test a portlet?**
- **Creating failing tests**
- **Making the tests pass (a.k.a. writing the portlet)**
- **Taking things further...**

Benefits and principles of test-driven development

Lotus software



Benefits of test-driven development

- Requires clear requirements first
- Forces developer to understand the requirements before writing code
- Forces design first, in particular interface design
- Bugs are detected earlier – and hence are cheaper to resolve
- Makes refactoring safer and more straightforward
- Provides regression tests

Principles of test-driven development

- Write a failing test first, then write code to make the test pass
- Make small steps
- Test frequently
- Only write enough code to make the test pass
- Tests must be automated



Our example

Lotus software



Our example portal application

- **A simple portal application to assist the beer festival attendee, which we will call “Beer-to-Beer Networking”**
- **Consists of a single portal page and four cooperating portlets (or should that be “porter-lets”?)**
- **Ideal for a “mobile” scenario, but we will not discuss that aspect here**

Beer-to-Beer Networking

Calendar of Beers

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

Book a Beer

Booking **Harveistoun Engine Oil** and **Wasatch Polygamy Porter**

Number of pints:

Time of first pint:

Book!

List of Beers

Beers available on the 9th

Name ▲	% vol ▲	
Dogfish Head Snowblower Ale	6.5	<input type="checkbox"/>
Dogfish Head Lawnmower Ale	4	<input type="checkbox"/>
Harveistoun Old Engine Oil	6	<input checked="" type="checkbox"/>
Ridgeway Santa's Butt	6	<input type="checkbox"/>
Seriously Bad Elf	9	<input type="checkbox"/>
Sick Duck	11	<input type="checkbox"/>
Wasatch Polygamy Porter	12	<input checked="" type="checkbox"/>

Select

How to get there



Beer-to-Beer Networking

- **A calendar portlet that broadcasts a public render parameter (PRP) when a date is selected**
- **A "List of Beers" portlet that shows those beers available on the selected day**
 - Select desired beers and submit
- **A "Book a Beer" portlet that allows those beers to be poured for you on the selected day**
- **A directions portlet that shows you how to get to the venue serving the selected beers (or possibly home again)**
 - Potential integration with Google Maps etc



Our chosen portlet: “List of Beers”

- Shows nothing unless a there is a current “selected day”
- If a day is selected, shows a list of available beers
- When beer names are selected and a button pressed, an event is generated
 - Designed to be consumed by portlets such as “Book a Beer”



How do we test a portlet?

Lotus software



Portal-specific challenges

- **Testing portlet classes in isolation with (e.g.) JUnit is difficult**
 - Stubbing / mocking all platform classes is a lot of work
 - Difficult to correctly monitor / drive portlet's “external contract”
 - Events in and out
 - Public render parameter interactions
 - Behaviour on minimise / maximise etc.
 - How to test that the markup generated is correct?
 - What about the small bits of logic that inevitably slip into JSPs?
 - How to exercise the client-side portions like
 - JavaScript validations
 - Ajax calls

- **We think you need to drive the user interface directly**

Testing by driving the user interface

- **UI-level tests (sometimes called “over-the-glass” tests)**
 - Allow all user interface elements to be exercised
 - Including client-side code
 - Are more intuitive for analysts, business users or non-programmer testers to create

- **Tooling exists to support this approach**
 - Selenium / WebDriver to automatically drive a browser
 - Concordion to allow “natural language” test specification

Disadvantages of approach

- **Tests can take a long time to write, which discourages developers from writing them**
 - Good examples and a well-developed framework can mitigate this

- **UI-level tests can take longer to run than JUnit, which discourages developers from running them**
 - Demonstrate the benefits to developers

- **Complexity of test code increases with complexity of application**
 - Again, an existing framework can mitigate this
 - Employ people who have done it before ;-)



Adding the portal specifics

- **Need to test the portlet's “contract” with other portal components**
 - Generic “counterparty” portlets can provide a user interface that allow test tools to:
 - Set public render parameters
 - Generate events for the portlet to receive
 - Check public render parameters set by the portlet
 - Receive and log events generated by the portlet
- **Need to be able to identify user interface elements within a specific portlet instance**
 - Create organisational naming and markup structure standards
 - Create a framework of helper code for text fixtures



Creating failing tests for our portlet

Lotus software



Defining the contract for our example portlet

- **What public render parameters will it use?**
 - Namespace and name
 - Payload

- **What events will it send and receive?**
 - Namespace, name, data type, aliases
 - Payload

- **User interface**
 - Defined by the wireframe & users

Creating the tests for the portlet

- **Concordion specification**
 - The tests defined in “natural language”
- **WebDriver fixtures**
 - Java code to execute the tests (run as JUnit)
 - A simple framework for a simple example
- **Other tools are available:**
 - Concordion: Fit/FitNesse, JUnit
 - WebDriver: Watij, Windmill(?)



Concordion

- Is “an open source tool for writing automated acceptance tests in Java” (from the web site)
- The tests are written in an HTML file in natural language
 - `<p>The field “name” must contain the value “Old Peculier”</p>`
- The tests can be understood and even specified without programming knowledge, e.g. by BAs or business specialists
- The tests are then “instrumented” with Concordion attributes
 - `<p>The field
“name” must contain the
value
“Old
Peculier”</p>`
- Concordion attributes call methods in the corresponding “fixture” Java class that implement the described behaviour
 - e.g. `openWebPage(String url), getFieldValue(String fieldName)`



WebDriver

- Provides an API for driving a web browser
- Supports Firefox, IE, headless browser
- Now part of Selenium (originally separate)
- Provides a wide range of classes & methods to navigate, locate items on a page etc.

- Example:

```
public String getFieldValue(String fieldName) {  
    WebDriver driver = new FirefoxDriver();  
    WebElement element =  
        driver.findElement(By.name(fieldName));  
    return element.getValue();  
}
```



Demo





Making the tests pass

Lotus software



Create the portlet

- **Satisfy the inter-portlet contract in the portlet's definition**
 - Declare supported public render parameters
 - Declare supported publishing and processing events

- **Apply naming and structural standards required by organisation and by test framework**
 - Labels for checkboxes
 - IDs (namespaced) on tags, e.g. headers

- **Satisfy the user interface tests, preferably one-by-one**
 - Correct markup to appear on page
 - Required user actions made available and behaving correctly



Demo





Taking things further...

Lotus software



Continuous integration

- **Important part of agile development**
 - Usually use servers such as Hudson, CruiseControl

- **Will typically run:**
 - Static analysis tools (e.g. checkstyle, PMD)
 - Standalone JUnit tests
 - Code coverage tools (e.g. Cobertura)

- **We would like it to run our UI-level tests as well**
 - Requires some additional effort in deployment automation



Deployment automation

- **Unlike JUnit tests of standalone classes, need to actually deploy to WebSphere Portal to run tests**
 - Will probably therefore schedule to run our tests less frequently
- **Need to deploy other Portal artefacts along with the portlet**
 - Page to host portlet
 - Counter party portlets to tests events and PRPs in and out
 - Wires to connect portlet and event counter parties
- **Can determine these requirements and generate deployment scripts automatically by parsing *portlet.xml***

Journey-level tests

- **So far we have concentrated on testing a single portlet**
 - How do we test an entire portal application?

- **We can create UI tests at the level of user journeys**
 - Idea is to test the pages, wires and interactions between portlets
 - Do not repeat testing of functionality within individual portlets
 - Test the correct “flow” of the user experience

- **Challenges:**
 - Distinguishing between portlets / portlet instances on a page
 - Checking correct portlets present
 - Driving user actions on correct portlet
 - Deploying the whole portal application for continuous integration





Questions?

Lotus software

