Functional Testing

1

Test Automation

- Frequent regression testing
- earlier feedback on problems (lower cost to fix)
- enables continuous delivery
- statistics

2

Multiple Browser Problem

- Something works in Chrome but not IE
- different operating systems
- different plugins (e.g. Flash)

3

Difficulties

- can be hard if the UI is constantly changing
- can automate some level of UI testing, but perhaps not things like "does it look good", or "is XYZ element in the correct place"
- some requirements still need a human to verify them

4

Selenium

- Started as JS plugin that ran in browser
- could run in multiple browsers
- shortcomings of a JS lib - sandbox restrictions
- Selenium 2 - WebDriver

5

Supported browsers:

- Google Chrome 12.0.712.0+

- Internet Explorer 6, 7, 8, 9 - 32 and 64-bit where applicable
- Firefox 3.0, 3.5, 3.6, 4.0, 5.0, 6, 7
- Opera 11.5+
- HtmlUnit 2.9 (headless, limited JS support)
- Android - 2.3+ for phones and tablets (devices & emulators)
- iOS 3+ for phones (devices & emulators) and 3.2+ for tablets (devices & emulators)

Selenium IDE

- plugin for firefox
- recommended only for prototyping / debugging
- record/playback capability
- very scripted tests, virtually no reuse

Types of Tests

- Static content
  - does a page have the expected title
  - the expected image
  - the expected footer, etc

Types of Tests

- Links
  - checking for broken links, and/or that links go to the correct location

---

Types of Tests

- Functional Tests
  - user interaction with the site
  - form submission
  - usually multiple steps
  - registration, login, create listing, place bid, etc.

Assert vs. Verify

- assert fails the test
- verify logs a error but continues with the test

11

Locators

- locators are a mechanism for finding something in the DOM of a page
- Identifier
- Name
- XPATH
- CSS

12

Locators - Identifier

- id=loginForm

```
<form id="loginForm">
    <input name="username" type="text">
    <input name="password" type="password">
    <input name="continue" type="submit" value="Login">
    <input name="continue" type="button" value="Clear">
</form>
```

13

Locators - Name

- name=username

```
<form id="loginForm">
 <input name="username" type="text">
 <input name="password" type="password">
 <input name="continue" type="submit" value="Login">
 <input name="continue" type="button" value="Clear">
</form>
```

14

Locators - XPath

```
<html>
  <body>
   <form id="loginForm">
    <input name="username" type="text" />
    <input name="password" type="password" />
```

```
    <input name="continue" type="submit" value="Login" />
    <input name="continue" type="button" value="Clear" />
   </form>
  </body>
 <html>
```

- /html/body/form[1] - Absolute path (would break if the HTML was changed only slightly)
- //form[1] - First form element in the HTML
- //form[@id='loginForm'] - The form element with attribute named 'id' and the value 'loginForm'

15

Locators - XPath

```
<html>
  <body>
   <form id="loginForm">
    <input name="username" type="text" />
    <input name="password" type="password" />
    <input name="continue" type="submit" value="Login" />
    <input name="continue" type="button" value="Clear" />
   </form>
  </body>
 <html>
```

- //form[input/@name='username'] - First form element with an input child element with attribute named 'name' and the value 'username'
- //input[@name='username'] - First input element with attribute named 'name' and the value 'username'
- //form[@id='loginForm']/input[1] - First input child element of the form element with attribute named 'id' and the value 'loginForm'

16

Locators - XPath

```
<html>
  <body>
   <form id="loginForm">
    <input name="username" type="text" />
    <input name="password" type="password" />
    <input name="continue" type="submit" value="Login" />
    <input name="continue" type="button" value="Clear" />
   </form>
  </body>
 <html>
```

- //input[@name='continue'][@type='button'] - Input with attribute named 'name' and the value 'continue' and attribute named 'type' and the value 'button'
- //form[@id='loginForm']/input[4] - Fourth input child element of the form element with attribute named 'id' and value 'loginForm'

17

Xpath resources

- W3Schools XPath Tutorial http://www.w3schools.com/Xpath/
- W3C XPath Recommendation http://www.w3.org/TR/xpath

18

Firefox Add-ons for XPath help

- XPath Checker - suggests XPath and can be used to test XPath results. (https://addons.mozilla.org/en-US/firefox/addon/1095?id=1095)
- Firebug - (https://addons.mozilla.org/en-US/firefox/addon/1843)

19

Page Object Pattern

- DRY
- class that represents a page (or components of a page) and the elements on that page
- can also have 'actions' defined for a given page
- test method "drives" the page objects and performs assertions

20

Sample Test

```
1  /***
2   * Tests login feature
3   */
4  public class Login {
5
6          public void testLogin() {
7                  selenium.type("inputBox", "testUser");
8                  selenium.type("password", "my supersecret password");
9                  selenium.click("sign-in");
10                 selenium.waitForPageToLoad("PageWaitPeriod");
11                 Assert.assertTrue(selenium.isElementPresent("compose butt
12                                 "Login was unsuccessful");
13         }
14 }
```

21

---

problems with this approach

- no separation between the test method and the IDs - tightly coupled into the test

- doesn't allow reuse of the "login" functionality
- doesn't allow reuse of the locators
- difficult to maintain as test suite grows and application changes

22

---

proposed solution:

```
/**
 * Page Object encapsulates the Sign-in page.
 */
public class SignInPage {

        private Selenium selenium;

        public SignInPage(Selenium selenium) {
                this.selenium = selenium;
                if(!selenium.getTitle().equals("Sign in page")) {
                        throw new IllegalStateException("This is not sign
                                        +selenium.getLocation());
                }
        }

        //...continued
```

23

```
        /**
         * Login as valid user
         *
         * @param userName
         * @param password
         * @return HomePage object
         */
        public HomePage loginValidUser(String userName, String password)
                selenium.type("usernamefield", userName);
                selenium.type("passwordfield", password);
                selenium.click("sign-in");
                selenium.waitForPageToLoad("waitPeriod");

                return new HomePage(selenium);
        }
}

```

24

```
/**
 * Page Object encapsulates the Home Page
 */
public class HomePage {

        private Selenium selenium;

        public HomePage(Selenium selenium) {
                if (!selenium.getTitle().equals("Home Page of logged in u
                        throw new IllegalStateException("This is not Home
                                        "is: " +selenium.getLocation());
                }
        }

        public HomePage manageProfile() {
                // Page encapsulation to manage profile functionality
                return new HomePage(selenium);
        }

        /*More methods offering the services represented by Home Page
        of Logged User. These methods in turn might return more Page Obje
        for example click on Compose mail button could return ComposeMail

}
```


---

now the test looks more like this:

```
/***
 * Tests login feature
 */
public class TestLogin {

        public void testLogin() {
                SignInPage signInPage = new SignInPage(selenium);
                HomePage homePage = signInPage.loginValidUser("userName",
                Assert.assertTrue(selenium.isElementPresent("compose butt
                                "Login was unsuccessful");
        }
}
```

Geb

- Groovy Domain Specific Language wrapper on top of web driver
- JQuery-like selectors
- Page Object first class citizen (and extension with Modules)
- http://www.gebish.org/

Geb Selectors

```
// match all 'div' elements on the page
$("div")

// match the first 'div' element on the page
$("div", 0)

// match all 'div' elements with a title attribute value of 'section'
$("div", title: "section")

// match the first 'div' element with a title attribute value of 'section
$("div", 0, title: "section")

// match all 'div' elements who have the class 'main'
$("div.main")

// match the first 'div' element with the class 'main'
$("div.main", 0)
```


Scripting with Geb

```
import geb.Browser
Browser.drive {
    go "http://google.com/ncr"

    // make sure we actually got to the page
    assert title == "Google"

    // enter wikipedia into the search field
    $("input", name: "q").value("wikipedia")

    // wait for the change to results page to happen
    // (google updates the page dynamically without a new request)
    waitFor { title.endsWith("Google Search") }

    // is the first link to wikipedia?
    def firstLink = $("li.g", 0).find("a.l")
    assert firstLink.text() == "Wikipedia"

    // click the link
    firstLink.click()

    // wait for Google's javascript to redirect to Wikipedia
    waitFor { title == "Wikipedia" }
}
```

Module Objects

```
// modules are reusable fragments that can be used across pages that can
// here we are using a module to model the search function on the home an
class GoogleSearchModule extends Module {

    // a paramaterised value set when the module is included
    def buttonValue

    // the content DSL
    static content = {

        // name the search input control “field”, defining it with the jQ
        field { $("input", name: "q") }

        // the search button declares that it takes us to the results pag
        // parameterised buttonValue to define itself
        button(to: GoogleResultsPage) {
            $("input", value: buttonValue)
        }
    }
}
```


Page Objects

```
class GoogleHomePage extends Page {

    // pages can define their location, either absolutely or relative to
    static url = "http://google.com/ncr"

    // “at checkers” allow verifying that the browser is at the expected
    static at = { title == "Google" }

    static content = {
        // include the previously defined module
        search { module GoogleSearchModule, buttonValue: "Google Search"
    }
}
```


Page Objects

```
class GoogleResultsPage extends Page {
    static at = { title.endsWith "Google Search" }
    static content = {
        // reuse our previously defined module
        search { module GoogleSearchModule, buttonValue: "Search" }

        // content definitions can compose and build from other definitio
        results { $("li.g") }
        result { i -> results[i] }
        resultLink { i -> result(i).find("a.l") }
        firstResultLink { resultLink(0) }
    }
}

class WikipediaPage extends Page {
    static at = { title == "Wikipedia" }
}
```

Script with Page Objects

```
Browser.drive {
    to GoogleHomePage
    assert at(GoogleHomePage)
    search.field.value("wikipedia")
    waitFor { at GoogleResultsPage }
    assert firstResultLink.text() == "Wikipedia"
    firstResultLink.click()
    waitFor { at WikipediaPage }
}
```

References

- http://seleniumhq.org/docs/
- https://github.com/iainrose/page-objects
- http://mattoncloud.blogspot.com/2011/06/selenium-webdriver-linux-headless-ruby.html
- http://blog.andresteingress.com/wp-content/uploads/2011/04/Geb_Confess.pdf
- http://www.gebish.org/manual/current