Web Applications Testing

Automated testing and verification

JP Galeotti, Alessandra Gorla
Why are Web applications different

Web 1.0: Static content
Client and Server side execution
Different components and technologies co-exist

Web 2.0: Dynamic contents
Heavy use of JavaScript and Ajax.
Users can collaborate, socialize and share...
Important qualities in Web applications

• Reliability

• Usability

• Performance

• Robustness/Scalability

• Security
Usability

• Assessing the ease of use and learnability of web application
  
  • Useful
  
  • Ease of use
  
  • Ease of learning
  
  • Pleasant look
Usability: Bad examples

A lot of information in the homepage

Several clicks before finding wanted information

Bad colors
Assessing Usability

- Some properties can be verified to assess usability:
  - Use colors that are compatible with color-blind people
  - Have a limited number of clicks to reach any information on the website
  - Provide a maximum number of links per page.
  - ...

Thursday, January 31, 13
Usability testing

• Usability testing techniques can be applied to web applications as well:
  
  • **Exploratory** testing to investigate mental model of users
  
  • **Comparison** testing to evaluate different options
  
  • **Usability** validation testing to assess the overall usability
Performance

• Assure that the web application can deal with expected workloads.

• E.g. I should expect usually between 100 to 1000 user requests per hour.

• Workloads can vary
Load test

- Simulate expected workload to test the ability of web application to deal with it.

- Goals:
  - Evaluate if the current infrastructure (HW and SW) is enough to satisfy the required SLA given the amount of expected workload.
  - Estimate how many requests can be currently handled.
  - Typically done when the system is ready.
Approach for Load Testing

- Identify performance acceptance criteria
  - Page should load fast --> Page should be fully loaded in less than 3 seconds.

- Identify key scenarios
  - E.g. Browse catalog; Select item; Add to cart; buy;

- Create a workload model
  - Define workload distribution, considering different scenarios

- Identify the target load levels

- Identify metrics

- Design specific tests

- Run tests

- Analyze results
Load test

- Historically tests were manually done (asking real users to perform tests)

- Now, lots of automation.
  
  - Tools can register activities of real users and create scripts to reproduce the same activities.

  - Scripts can be configured to create a set of virtual users who do same activities at the same time.
LoadRunner from Mercury (now HP)

Industrial standard for load testing.

Can emulate the load generated by any number of users, from few to several thousands.

Supports different types of applications and systems: web server, database, ERP, firewall...
JMeter - the open source alternative

• GUI desktop application designed to load test functional behavior and measure performance. It was originally designed for testing Web Applications but has since expanded to other test functions.

• It can be used to simulate a heavy load on a server, network or object to test its strength or to analyze overall performance under different load types

• Remote Distributed Execution: it is possible to generate load using multiple test servers (by running multiple server components of JMeter remotely).

  • Possible to control it by a single JMeter GUI to gather results.
Create a test plan: Sampler

Create HTTP/HTTPS requests to a web server.

Possible to specify whether JMeter should parse HTML for resources (e.g. images, frames...)

---

Thursday, January 31, 13
Logic controllers

Determine the order in which Samplers are processed.

- Execute sequentially
- Execute samplers in loops
- Alternate among different samplers
- Random
- Conditionally execute

...
Listeners

Meant to “listen” to the test results. Provide means to view, save and read saved test results.
Robustness/Scalability

- Assure that software behaves in acceptable way even in presence of situations that were not mentioned in the requirements.
  
  - The most important variable for web apps is the workload.

- Scalability: Managing several requests with QOS. Being able to deal with workloads that are heavier than originally considered.
Stress testing

Highest estimated workload: 1000 requests per second

Test with 2000, 5000, 10000 ... requests per second
Security testing

• Goal: Assure that the application protects data and maintains functionality as intended.

• Mostly manual, little automation support

Manual

Security experts try to actually break the system and find vulnerabilities

Automation

Static analysis techniques: scan application code and detect likely security flaws

Fuzz testing techniques: create random/malformed inputs to make app crash or find security hole
Fuzzers

• Generators use combinations of static fuzzing vectors (values that are known to be dangerous), or totally random data.

• A fuzzer would try combinations of attacks on:
  
  • numbers (signed/unsigned integers/float): zero, possibly negative or very big numbers
  
  • chars (URLs, command-line inputs): escaped, interpretable characters/instructions (e.g. quotes and commands for SQL requests)
  
  • metadata: (user-input text)
  
  • pure binary sequences

• Usually black-box

• White-box fuzzers start from well-formed inputs and use dynamic symbolic execution to generate new ones.
Reliability

- Assure that the system performs and maintains its functions in normal circumstances.

- Classic functional, structural and model based techniques can be used for web applications as well.
  - Different automation support to implement and run unit and system tests

- Major challenge: Assure that a web application works correctly in different environments (OS, Browsers ...)
  - Cross browser compatibility
Unit testing - HttpUnit

• **HttpUnit**: JUnit for web

• It models HTML documents, and provides an API that allows to invoke pages, fill out forms, click links...

```java
@Test
public void homePage() throws Exception {
    final WebClient webClient = new WebClient();
    final HtmlPage page = webClient.getPage("http://htmlunit.sourceforge.net");
    Assert.assertEquals("HtmlUnit - Welcome to HtmlUnit", page.getTitleText());

    final String pageAsXml = page.asXml();
    Assert.assertTrue(pageAsXml.contains("<body class="composite">"));

    final String pageAsText = page.asText();
    Assert.assertTrue(pageAsText.contains("Support for the HTTP and HTTPS protocols"));

    webClient.closeAllWindows();
}
```

Limited JavaScript support
Unit testing - JSUnit

• JUnit for Javascript

```html
<html>
    <script type='text/javascript'>
        function setUp(){
            // perform fixture set up
        }
        function tearDown() {
            // clean up
        }
        function testOneThing(){
            // instantiating a SystemUnderTest, a class in the drw namespace
            var sut = new drw.SystemUnderTest();
            var thing = sut.oneThing();
            assertEquals(1, thing);
        }
        function testAnotherThing(){
            var sut = new drw.SystemUnderTest();
            var thing = sut.anotherThing();
            assertNotEquals(1, thing);
        }
    </script>
</html>
```
Unit testing

• Other tools:
  
  • HtmlUnit (same as HttpUnit, but better Javascript support)
  
  • Canoo Web testing
    
    • Built on top of HtmlUnit

    • Makes it easier to organize and run test cases

  • ...
Unit testing - QUnit

• Advanced Javascript unit testing framework

```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>QUnit Example</title>
  <link rel="stylesheet" href="/resources/qunit.css">
</head>
<body>
  <div id="qunit"></div>
  <div id="qunit-fixture"></div>
  <script src="/resources/qunit.js"></script>
  <script src="/resources/tests.js"></script>
</body>
</html>
```
System test

Easily capture and replay actions
Selenium IDE
Selenium RC

Windows, Linux, or Mac (as appropriate)...

// You may use any WebDriver implementation. Firefox is used here as an example
WebDriver driver = new FirefoxDriver();

// A "base url", used by selenium to resolve relative URLs
String baseUrl = "http://www.google.com";

// Create the Selenium implementation
Selenium selenium = new WebDriverBackedSelenium(driver, baseUrl);

// Perform actions with selenium
selenium.open("http://www.google.com");
selenium.type("name=q", "cheese");
selenium.click("name=btnG");

// Get the underlying WebDriver implementation back. This will refer to the
// same WebDriver instance as the "driver" variable above.
WebDriver driverInstance = (WebDriverBackedSelenium) selenium).getWrappedDriver();

// Finally, close the browser. Call stop on the
// WebDriverBackedSelenium instance
// instead of calling driver.quit(). Otherwise, the JVM will continue running after
// the browser has been closed.
selenium.stop();

APIS in Java, Python, Ruby, Php, JavaScript ...
Cross browser compatibility

- Elements in Web pages may appear differently on different combinations
- Functionality may work on some browsers and not on others
- Most of the issues are related to CSS and JavaScript
Example of layout issue
Cross browser testing

• Load specific web page on relevant combinations of browsers, OS and resolutions and compare the output.

• Provide URL to a service, and have your web page loaded in the requested combinations.

• A set of “screenshots” as a result.
Comparing results

Screenshots

Document Object Models

Can be automated. Tricky, DOM APIs are different across browsers

mostly manual
Webmate

http://www.st.cs.uni-saarland.de/webmate/