CSCC09F
Programming on the Web

Testing Web Applications

Whither Testing

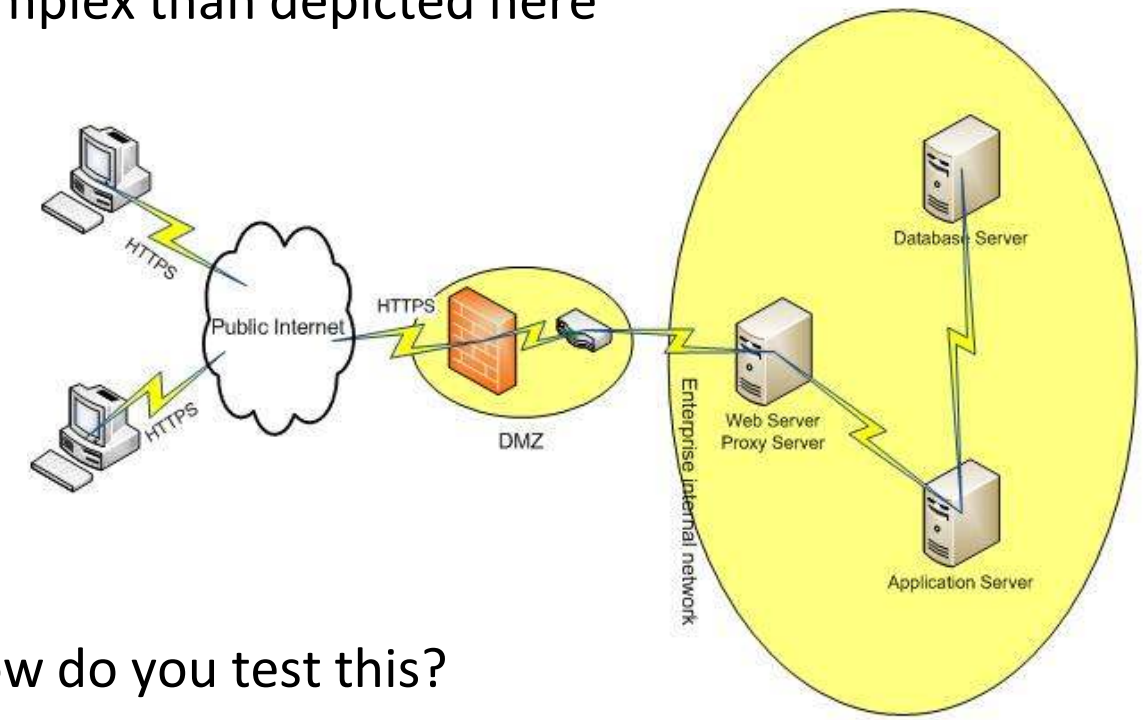
- Why no testing requirement for our assignments?
- Are Web apps so transparent and simple to write that we know they work without testing?
- Or maybe it is ok to let users be the testers for Web apps?
- Multi-tiered applications are hard to debug, but tend to be tested less. Does that make sense?
- A Web app's testability is in part reliant on how the app is designed – want design for testability.
- Unit, regression, functional, and other kinds of tests are just as important for Web apps as any other kind of app!

Whither Testing

- Is testing more or less important when moving from traditional server-powered apps using languages like Java to full-stack client-powered apps (SPA's)?
- JavaScript is an agile language – easy to change things, fewer constraints imposed by static checking
- But … that means more things might just happen to work by chance in one case and not another
- Testing at a comprehensive level is essential for building reliable and robust full-stack apps!
- But … testing is hard and time-consuming to set up – easy to leave it until the last moment or drop from the schedule

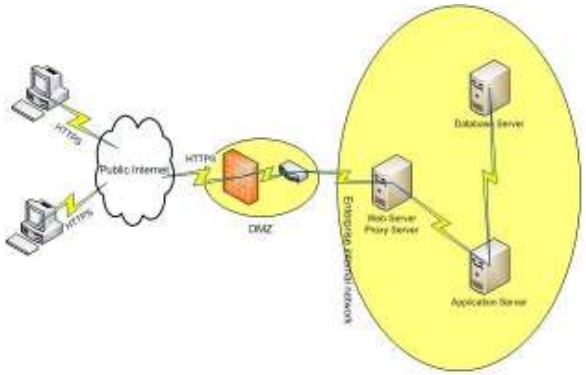
Tester's Dilemma

- The "N-tier architecture" of Web apps – actually more complex than depicted here



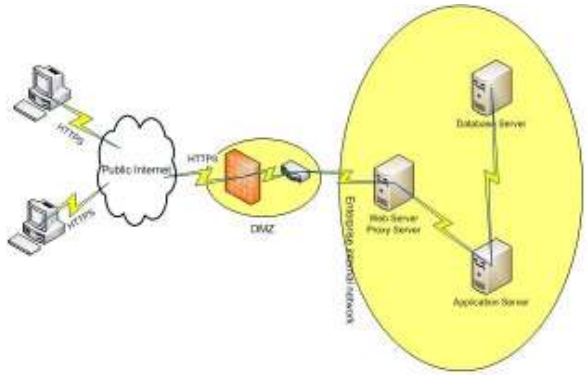
- How do you test this?

Black Box: Input-Output Testing



- Testing Proposal:
  - Trigger browser request (e.g. click a link, submit a form, run some JavaScript)
  - Capture the HTTP response body, e.g. HTML, and compare it with desired content
- Can this be automated? If not, pretty hopeless, since we have to regression test for every change
- What happens if the HTML changes, e.g. due to page redesign?

Designing for Testability



- Modularity or loose-coupling are important not just to support code development and maintainability … but also to support modular and efficient testing
- Ideally want to test each module independently, then integrate them into a complete app. Integration harder if modules depend-on/tamper-with each other's state
- Organizing code into coherent components like models, views, controllers, helps by giving us a roadmap of what happens where, and what kind of tests are required for each type (contrast with monolithic/spaghetti structure)

TDD and Unit Testing

- Developers are usually expected to provide their own suite of unit-tests to complement other kinds of testing such as functional, stress etc. performed by a separate SQA group
- Rather than “roll your own”, preferable to take advantage of a testing framework to aid in structuring your unit tests
- Testing tools are in transition as SPA’s rise in importance …
- HTTPUnit and HTMLUnit are widely-used tools for apps built with classic server-oriented architecture – that is, an app whose business logic is mostly implemented on the server-side without much client-side business logic (JavaScript)

Testing Tools: HttpUnit

- HttpUnit
  - Open Source Java library that works with Junit testing framework
  - Tests written in terms of HTTP requests and responses, e.g.:

```
WebConversation wc = new WebConversation();
WebResponse response = wc.getResponse(
    "http://mathlab…:28001/dishes");
```

  - response can be manipulated either as plain text (with getText()), or as a DOM (using getDOM())
  - Support for handling form submission, cookies, http authentication, and other header information, redirects, etc
  - Does not deal well with pages that use JavaScript, especially 3rd party JS libraries (like jQuery)

Testing Tools: HtmlUnit

- HtmlUnit
  - Open Source Java library that models returned-page rather than HTTP transaction (has largely taken over from HttpUnit)
  - Somewhat more HTML-oriented than HttpUnit (which can process non-HTML results)
  - Makes it easy to organize tests by category – e.g. click all anchor links, for forms fill in fields and submit
  - Works with Junit
  - Better JavaScript support than HttpUnit, but not guaranteed to work with 3rd party JS libraries such as jQuery or Backbone

Testing Tools: HtmlUnit

- Here's an example showing how we might test the login form for the Eatz app:

```
WebClient webClient = new
      WebClient(BrowserVersion.FIREFOX_10);
HtmlPage login =
  webClient.getPage("https://mathlab…/auth");
HtmlForm loginForm = login.getFormByName("login");
HtmlTextInput username =
      loginForm.getInputByName("username");
HtmlPasswordInput password =
      loginForm.getInputByName("password");
HtmlSubmitInput dolog = loginForm.getInputByValue("Login");
username.setValueAttribute("testing");
password.setValueAttribute("4Testing");
HtmlPage loginResult = dolog.click();
```

Unit Testing with SinonJS

- Enter SPA's – old test frameworks oriented toward HTML, not JS, but JS is the heart of an SPA …
- SinonJS is a library that provides standalone test spies, stubs, and mocks specifically for JavaScript
- Other libraries provide comparable unit-testing features, e.g. Mockery, but SinonJS appears to be the most popular
- SinonJS provides a testing framework that enables you to verify the behavior of the more complex parts of your app:
  - callbacks and their arguments
  - stubbing out parts of your application to reduce dynamic behavior and to test at an early stage (when other components aren't yet written)
  - present mock interfaces for things like AJAX requests

Unit Testing with SinonJS

- Sinon.js provides a toolkit for performing unit testing of SPA's including:
  - Function spies that give you visibility into arguments (actual parameters), return values, `this`, and exceptions without changing the behavior of the observed function
  - Stubs for functions that don't yet exist or that you want to isolate your test from (can even stub the console)
  - A fake XMLHttpRequest to enable you to test that your SPA is issuing the correct Ajax requests
  - Fake servers to test an app standalone without server support
  - Faking time to test time-sensitive logic such as setTimeout()

Unit Testing with SinonJS

- This function does what? Which properties would we want to unit-test for? How would you test for those properties?

```
function once(fn) {
  var returnValue, called = false;
  return function () {
    if (!called) {
      called = true;
      returnValue = fn.apply(this, arguments);
    }
    return returnValue;
  };
}
```

Unit Testing with SinonJS

- spy functions record their arguments, return value, value of **`this`** and any exceptions thrown
- Original function, being spied upon, behaves normally

```
it("calls original function", function () {
    var callback = sinon.spy();
    var proxy = once(callback);

    proxy();

    assert(callback.called);
});
```

Unit Testing with SinonJS

- How to verify that parameter `fn()` gets called just once?

```
it("calls param func only once", function ()
{
    var callback = sinon.spy();
    var proxy = once(callback);
    proxy();
    proxy();

    assert(callback.calledOnce);
    // ...or:
    // assert.equals(callback.callCount, 1);
});
```

Unit Testing with SinonJS

- Lots of problems arise due to improper binding of **this**

```
it("calls param fn with right this and args",
  function () {
    var callback = sinon.spy();
    var proxy = once(callback);
    var obj = {};
    proxy.call(obj, 1, 2, 3);

    assert(callback.calledOn(obj));
    assert(callback.calledWith(1, 2, 3));
});
```

Unit Testing with SinonJS

- Supports stubs for not-yet-written functions or functions that you want to test in isolation:

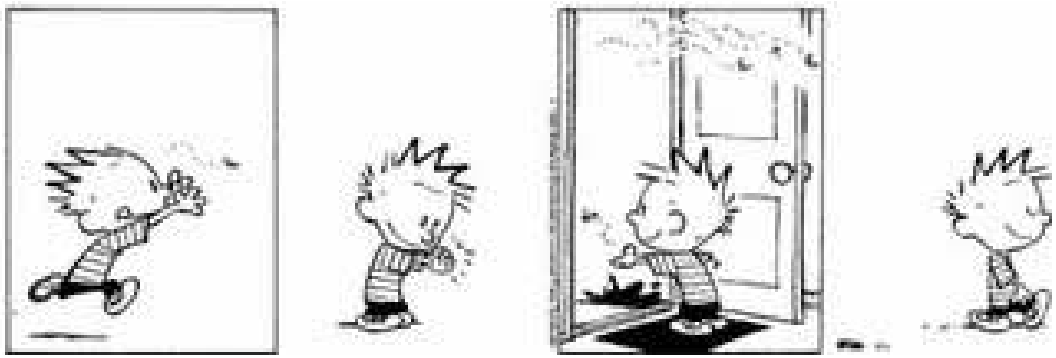
```
it("returns original-function return value ",
  function () {
    var callback = sinon.stub().returns(42);
    var proxy = once(callback);

    assert.equals(proxy(), 42);
});
```

Mock Objects

- How do you test with external resources, that may not be available in the test environment, or may incur a high per-use cost or time penalty?
- Substitute a "mock object" that implements same interface, but at a much lower cost (e.g. simplified implementation).
  - e.g. a mock Web-service that returns a valid but artificial set of dish geo-location values
- Risk of mock-object testing is that it does not faithfully implement desired interface, or behavior of actual system changes and mock version does not adapt (so tests pass, but system fails when used with real external resource)

Regression Testing

- What's the point?



- Developer fixes a bug (yaay), but in the process they reintroduce a previously-fixed bug or new bugs … much more common than you might expect!
- Regression tests to the rescue: whenever you find a bug, create a test that exposes it, add test to regression test suite, run regression suite after every code change

UI Regression Testing

- Regression testing could operate by:
  - Recording expected output (e.g. HTML) for each given input (HTTP request)
  - Regression test involves re-testing with the same input (HTTP request) and comparing output with the previous recorded value (e.g. HTML)
  - Still too much dependency on page layout/design/UI, but at least now your SQA person only has to walk through the test scenario once and then can play back N times
  - Example of tool to do this kind of testing: Selenium

UI Regression Testing

- What's wrong with the

  "assert(HTTP-in == HTML-out)" approach?
- Biggest issue is dependency on the details of HTML response (aka "screen scraping").
- How do we fix this?
- Could adopt a more CSS-like approach by marking certain parts of the page to denote their purpose/meaning, and then checking these markers rather than literal HTML
  - not a perfect solution, but reduces test dependency on exact string format of HTML which is fragile (reduces coupling)

Database Testing

- What makes DB-based app testing hard?
- DB updates persist in DB, but for many tests would like to start in a known initial state, not a state determined by prior n-1 test cases that have run
- Problem: resetting a DB to an initial-value state can be slow/expensive
- Lightweight in-memory DB's help, since they don't require slow disk operations of a traditional DB
- Can mock NoSQL DB's like Mongo with JavaScript tools
- Must be fast enough to be practical (else won't get used)

Full Stack Testing

- Beyond unit and regression tests, an app will require integration testing to ensure that when units (components) of your app stack are integrated together that the combined functionality works properly
  - ensures that inter-component interfaces are properly implemented
- Functional or end-to-end testing test overall functionality of your app from the user's perspective.
  - typically driven by app use-cases
  - often the most complex tests, which may require database mocking/resets, headless browsers, 3rd party Web service stubbing

Full Stack Testing

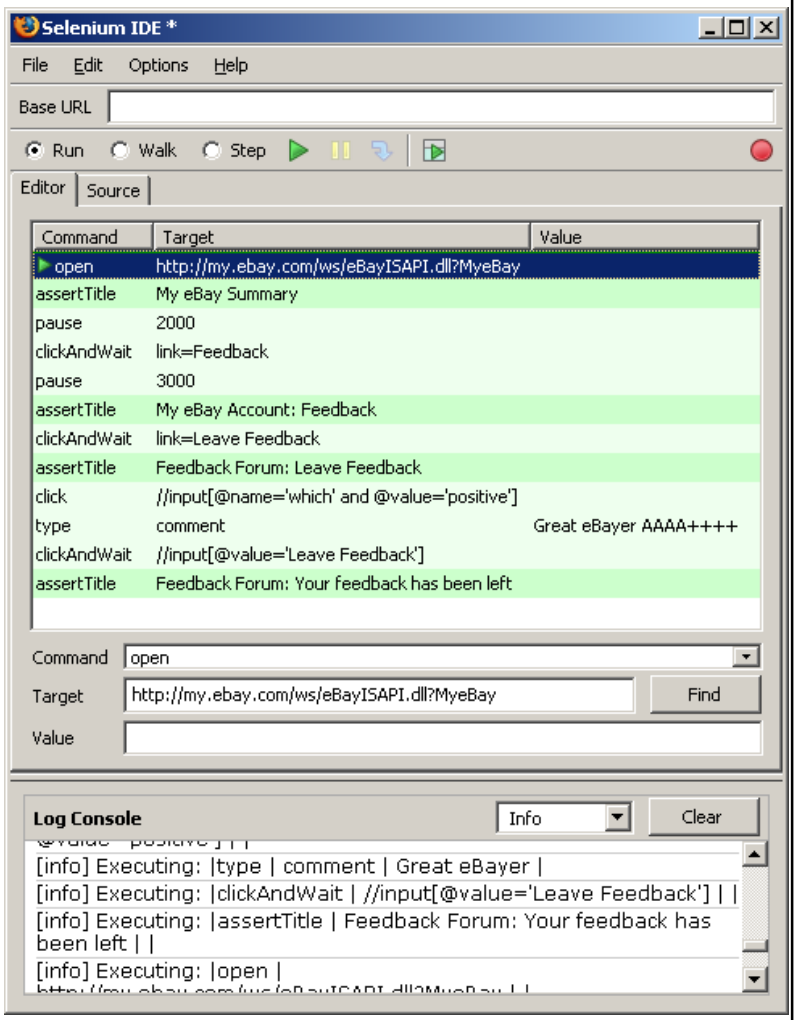
- Intern is an interesting library that automates a full-stack test by running a test suite across environments (browser, Node) and across platforms (various browsers)
- Selenium performs automated browser scripting (e.g. by record and playback)
- PhantomJS is designed specifically for running automated headless browser testing

Functional Testing: Intern

- Supports unit and functional testing as well as continuous integration
- Can run tests from browser or Node.js
- Browser testing mode supports mimicking of user interaction via a WebDriver browser extension
- Provides full code-coverage reporting so you know what's left to test
- Continuous integration support runs build-test cycle after every commit

Functional Testing: Selenium

- Open Source based on use of JavaScript running in browser iframe to interact with documents
- Cross-browser support
- UI interactions are recordable and scriptable
- Test scripts can be written in Java, Python, Ruby, C#, PHP
- Firefox extension Selenium IDE supports recording and playback



- Demo video: http://www.youtube.com/watch?v=JwJ7LfZiKGQ

Testing Tools: PhantomJS

- PhantomJS supports “headless” testing – that is, testing without a browser
  - since it cuts out browsers it tends to be a faster testing framework than Selenium
- Supports reading/writing of cookies, JavaScript execution
- Can runs tests written in a separate framework such as Mocha, Jasmine
- Includes a network monitor that can analyze network behavior and performance