

Client Side BDD

With

Jasmine

@jamescarr

<http://blog.james-carr.org>

Software Engineer,
Carfax, INC.

describe { James Carr }

- Agile software development with XP since 2005
- Started TDD'ing javascript at the same time (with jsUnit)
- Passionate at finding better ways of developing software
- Hard of hearing, so please speak LOUDLY

**Before we dig into BDD
in javascript, it might be
useful for us to first
tackle the obvious
question that might be
on your mind...**

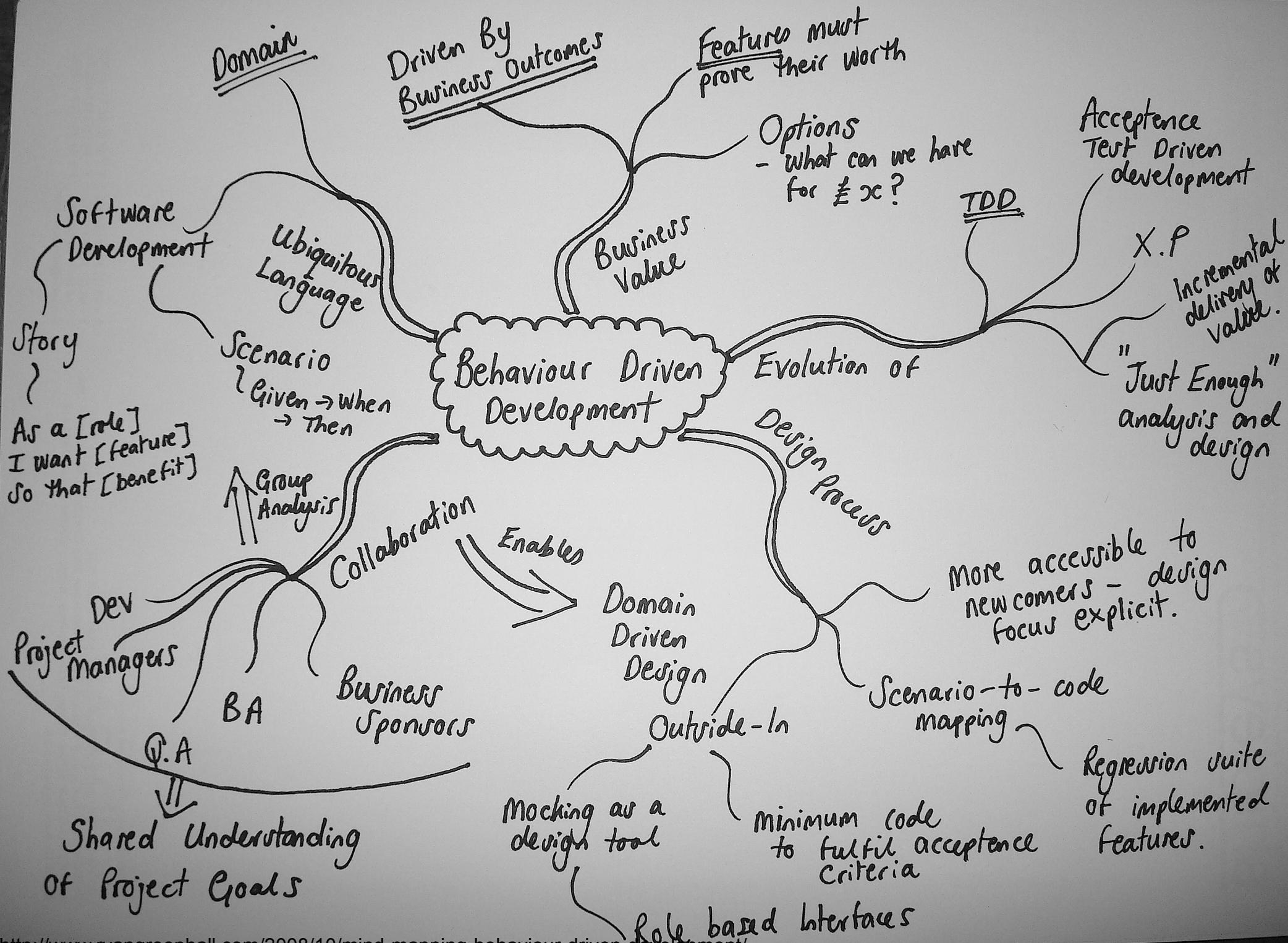
What is BDD?

BDD is TDD Done Right

Original intention of TDD was to use tests to drive the design of a system, not verify that it works



But BDD goes beyond this and includes much more, combining domain driven design and incremental business analysis



Goal Oriented

- Focus on the goals of different stakeholders
- Use Feature Injection as a means of realizing those goals
 - Versus focusing on features first
- Use examples (preferably close to the ubiquitous language of the business domain) to describe the application
- Automate those examples for quick feedback

At the Unit Level

- Describe each behavior using a sentence that makes sense
- Think in terms of how a client would use the API. In fact, think of a specification as less of a test and more as living, breathing documentation
- Describe behavior, working your way from the outside in from the top most level

Terminology

- Specification (spec) – The overall unit of behavior you are testing (identified by top level describe)
- Context – the specific context of your examples, sets the initial context
 - Identified by both top level and nested describes, each context builds on top of it's outer context
- Exemplar – an example of specific functionality that falls within the context
- Expectation – a verification of behavior
 - In plain English: expect $2+2$ to equal 4

Context

- Context matters
- A way to establish the surrounding context that the system is operating in
- Essentially all the surrounding conditions that are assumed to be met for the given scenario
- Given...
- Not always needed.

Exemplar

- An example of usage
- The real target of what you are describing
- A good example would be when you look at the javadocs for a method and see an example of how it works
- An action that will have some observable outcome

Expectation

- A verification of the expected result of an action
- Observable behavior
- Outcome of behavior might be an interaction with another entity or component in the system
- In this case we might verify collaboration between components using spies

The Focus of this Talk

- No lessening of importance on the full ecosystem of BDD. Remember, if you're not working closely with stakeholders, you're not doing BDD
- Focus on the unit level and how to describe behavior of javascript related applications
- How you can get these specs running within your current build system.

BDD/TDD in Javascript

- Javascript has a wide variety of TDD/BDD frameworks
 - Junit
 - YUITest
 - JsTestDriver
 - Jsspec
 - Jspec (my favorite, however discontinued)
 - Screw.Unit
 - Jasmine
- Each of these have their own pros/cons too numerous to delve into great detail, but here's some of the features I like:
 - YUITest has the feature to post results to a url
 - JsTestDriver is really more of a runner that can run tests/specs in an IDE
 - Jspec has a built-in DSL that resembles rspec



BDD for JavaScript

Jasmine is a behavior-driven development framework for testing your JavaScript code. It does not depend on any other, that you can easily write tests.

```
describe("Jasmine", function() {  
  it("makes testing JavaScript awesome!", function() {  
    expect(yourCode).toBeLotsBetter();  
  });  
});
```

Jasmine can be run anywhere you can execute JavaScript: a static web page, your continuous integration environment, Find out more in the [documentation](#).

Downloads

<http://pivotal.github.com/jasmine/>

Why Jasmine?

- Community
 - Lots of plugins and extensions
- Support for Nested Contexts
- Very well supported and actively developed
- Support for describing asynchronous behavior

Standalone Runner

- No special tools needed to get started, just download standalone distribution and extract
- <http://pivotal.github.com/jasmine/download.html>
- Open the SpecRunner.html and you're running specs
- You'll have to manually add and remove specs and javascript as needed

Alternative Runners

- Ruby Gem
- Node.js based runner
- Maven Plugin
- More on these later, let's first focus on the actual framework and writing specs

A Simple Walkthrough

- All the runners come with a demo project that mimics the classical Player example that was used eons ago with rspec
- A little more interesting: HTML5 placeholder emulator
- `git@github.com:jamescarr/html5-placeholder-emu.git`

```
describe("HTML5 Placeholder", function(){
  beforeEach(function(){
    input = $('<input value="" placeholder="First Name">');
  });

  describe("Element with empty value", function(){
    it("should display text from placeholder in field", function(){
      placeholder.emulate(input);

      expect(input).toHaveValue('First Name');
    })
  })
})
```

Up front, don't try to account for everything, just do enough to get running. You can cover other conditions later. :)

Now run, see it fail and make it pass.

```
(function(ctx){  
  ctx.placeholder = {  
    emulate: function(input){  
      var placeholder = $(input).attr('placeholder');  
      if(placeholder){  
        $(input).val(placeholder);  
      }  
    }  
  };  
})(this);
```

This passes the first example, but we know we're not done. How? Now is a good time to write the next example...

```
describe("Element with empty value", function(){
  beforeEach(function(){
    placeholder.emulate(input);
  });

  it("should display text from placeholder in field", function(){
    expect(input).toHaveValue('First Name');
  })

  it("should add the class placeholder to the element", function(){
    expect(input).toHaveClass('placeholder');
  })
})
```

Here we noticed that each example will have an initial context of having the input placeholder emulated, so we extract that to `beforeEach` (which will be executed before each example)

We then iteratively work our way in, describing each feature as needed.

You can see where this is going... feel free to view the full result at
<https://github.com/jamescarr/html5-placeholder-emu>

In Detail

- In the previous examples, each describe is a “context”
- Each “it” is an example
- Examples live inside of a context, with beforeEach running before each example
 - Contexts will inherit anything that goes on in their outer contexts (in the previous examples, beforeEach in the outer context runs, then the beforeEach in the inner context will run)

Ruby Gem

- There is also a ruby gem that runs specs through autospec as well
- This is decent if you are a rubyist (in fact, it's freaking popular as heck in the rails community)
 - `gem install jasmine`
- Creates a Rakefile with two tasks
 - `rake jasmine`
 - `rake jasmine:ci`

Maven Plugin

<https://github.com/searls/jasmine-maven-plugin>



jamescarr

[Dashboard](#)

[Inbox](#) 0

[Account Settings](#)

[Log Out](#)

[Explore GitHub](#)

[Gist](#)

[Blog](#)

[Help](#)



😊 searls / [jasmine-maven-plugin](#)

[Watch](#)

[Fork](#)

[50](#)

[20](#)

Source

[Commits](#)

[Network](#)

[Pull Requests \(0\)](#)

[Issues \(7\)](#)

[Wiki \(1\)](#)

[Graphs](#)

Branch: master

[Switch Branches \(3\)](#)

[Switch Tags \(6\)](#)

[Branch List](#)

Maven plugin to execute Jasmine Specs. Creates your HTML runners for you, runs headlessly, outputs JUnit XML — [Read more](#)

<http://about.me/searls>

[Downloads](#)

HTTP

Git Read-Only

<https://github.com/searls/jasmine-maven-plugin>



This URL has **Read-Only** access

Merge branch 'develop'



searls (author)

March 25, 2011

commit	bb568f757e4e69c9195b
tree	d9ada51f0ef5a6cfb94f
parent	6d13cbca129f1f86dfal
parent	fe87cc6c643f07ce3502

<https://github.com/searls/jasmine-maven-plugin>

[jasmine-maven-plugin](#) /

Archetype

```
mvn archetype:generate \  
-DarchetypeRepository=http://searls-maven-repository.googlecode.com/svn/trunk/snapshots \  
-DarchetypeGroupId=com.github.searls \  
-DarchetypeArtifactId=jasmine-archetype \  
-DarchetypeVersion=1.0.2-SNAPSHOT \  
-DgroupId=com.acme \  
-DartifactId=my-jasmine-project \  
-Dversion=0.0.1-SNAPSHOT
```

Generates two source directories:

/src/main/javascript

/src/test/javascript

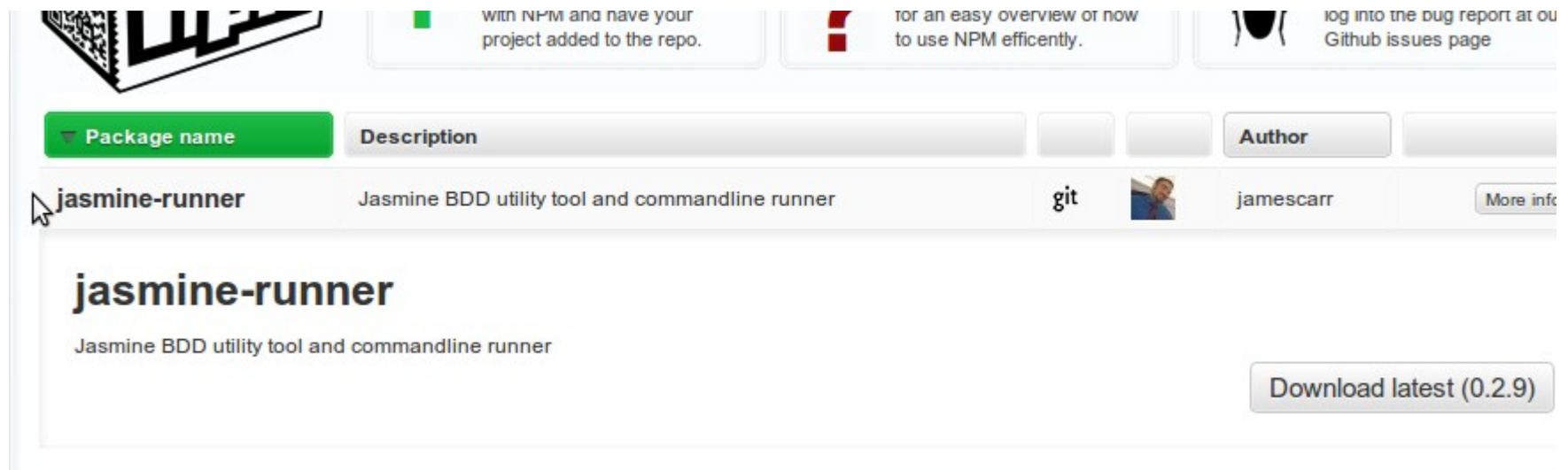
More Details

- Runs as part of the mvn test target
- Packaged war places contents of src/main/javascript under the context root
- Configurable
- Plugs in with WTP for you eclipsers :)

Warning:
Shameless Plug

jasmine-runner

- <https://github.com/jamescarr/jasmine-tool>
- Written in node.js, 100% javascript
- Available via npm (<http://npmjs.org>) package management



The screenshot shows the npm package page for 'jasmine-runner'. At the top, there are three promotional banners: one with a QR code, one about adding projects to the repo, and one about logging into the bug report page. Below these is a table with columns: Package name, Description, git, Author, and More info. The first row shows 'jasmine-runner' with the description 'Jasmine BDD utility tool and commandline runner', the 'git' logo, the author's profile picture and name 'jamescarr', and a 'More info' link. Below the table, the package name 'jasmine-runner' is displayed in large text, followed by the description 'Jasmine BDD utility tool and commandline runner'. A 'Download latest (0.2.9)' button is located at the bottom right.

Package name	Description	git	Author	More info
jasmine-runner	Jasmine BDD utility tool and commandline runner	git	jamescarr	More info

jasmine-runner
Jasmine BDD utility tool and commandline runner

Download latest (0.2.9)

jasmine-runner

- `npm -g install jasmine-runner`
- `jasmine` (same as the ruby gem)
 - `init` – creates a project directory
 - `run` – same behavior as “`rake jasmine`” with gem
 - `mon` – runs a server and monitors for changes
 - `ci` – starts server, runs browser and exits
 - Not complete
 - Will soon generate junit xml

jasmine-runner

- Awesome new feature: coffeescript support
- See <http://jashkenas.github.com/coffee-script/>
- Specs and js files that end with coffee are compiled down to js and served to the client side on the fly when specs run
- Mix and Match: You can have specs written in coffeescript with real code written in js

More Dirty Details

Matchers

- Used to verify expectations
- Comes bundled with many core matchers
 - `ToBe(expected)` - compares the actual to the expected using `===`
 - `toEqual(expected)` – uses common sense equality
 - `toContain(expected)` – checks that the expected element is in the actual array
 - `toMatch(expected)` – compares actual to expected using regular expression
 - `toBeLessThan/toBeGreaterThan` – obvious :)
 - `toBeDefined()` - verified the variable is not undefined
 - `toBeFalsy/toBeTruthy`
 - All matchers can be inversed by using `not`
 - Example: `expect(3).not.toEqual(5)`

Writing Your Own Matchers

- Helps clarify the behavior being described by being more descriptive and focused
- Matcher function is a simple function that takes expected as an argument and returns true/false based on some comparison to this.actual.

- Example: toBeLessThan:

```
toBeLessThan: function(expected) {  
  return this.actual < expected;  
};
```

- To add matchers to a suite, call `this.addMatchers({})` within a before or it block.

Before and After

- Simply put, allow setting up or tearing down the context
 - `beforeEach()` - do before each exemplar executes
 - `afterEach()` - do after each exemplar executes
- How can I setup or tear down the entire context rather than each exemplar?

Spies

- Jasmine comes bundled with the capability of using spies to fake, stub, and mock behaviors
- Provides ability to both stub and spy
 - `spy(x, 'methodName').andCallThrough()` - spies the method call and calls the real method
 - `AndReturn`, `andThrow`, `andCallFake()`
- Specific matchers:
 - `toHaveBeenCalled`, `toHaveBeenCalledWith`
- Has some other useful properties:
 - `callCount` – number of times the spied method was called
 - `mostRecentCall.args` – returns argument array from the most recent call to the spy
 - `argsForCall[i]` – returns the arguments for a specific call to the spy

Spying Example

- The most common example is we want to verify an ajax call took place
- So let's describe a searcher object that wraps calls to a search service

SearcherSpec.js

```
describe("Searcher", function(){
  var callback = function(){};
  beforeEach(function(){
    spiedJquery = spyOn(jQuery, 'post');
    searcher = new Searcher();
  })
  it("should make a post request for the search term",
  function(){
    searcher.search('beer', callback);

    expect(spiedJquery, 'post')
    .toHaveBeenCalledWith('/search', {term:'beer'}, callback);
  })
})
```

Searcher.js

```
function Searcher(){}  
Searcher.prototype.search =  
function(term, cb){  
    $.post('/search', {term:term}, cb);  
}
```


Customize Spy Behavior

- `spyOn(x, 'method').andCallThrough()`: spies on AND calls the original function spied on
- `spyOn(x, 'method').andReturn(arguments)`: returns passed arguments when spy is called
- `spyOn(x, 'method').andThrow(exception)`: throws passed exception when spy is called
- `spyOn(x, 'method').andCallFake(function)`: calls passed function when spy is called

CI Integration

- Different approaches
 - jasmine gem has a ci command that reports results through rspec
 - Maven plugin runs specs and reports results as junit style xml
 - <https://github.com/searls/jasmine-maven-plugin>
 - Jasmine-tool currently has a ci command that will dump results to the console
 - More on the way. For java shops, your best bet is probably the maven plugin as you can capture the junit xml it produces

Tips

- Try to avoid mocking ajax calls.
 - JQuery pubsub
 - <https://github.com/phiggins42/bloody-jquery-plugins/>
- Focus on behavior, not “ensuring things work”
- Try to nest contexts to identify different concerns.
 - Provides a good guide on areas where behavior could be split into separate components
- Don't think like a java developer. :)

Extensions

- <https://github.com/pivotal/jasmine/wiki/Related-projects>
- A few favorites:
 - jasmine-jquery
 - jasmine-species
 - <http://rudylattae.github.com/jasmine-species/>
 - jasmine-ajax

Questions?

Thank You!

Please Remember to
Turn In Your
Evaluations



Code

All of the examples from this presentation can be
found at

[http://github.com/jamescarr/javascript-bdd-
presentation](http://github.com/jamescarr/javascript-bdd-presentation)