# Javascript Testing Tactics
# How my JavaScript Tests differ from the README.
My name is Justin Searls
Please tweet me @searls &
Say hello@testdouble.com
## background
## background

* ⚡️ 20 minutes! ⚡️
## background

* ⚡️ 20 minutes! ⚡️

* ~~purposes of each type of test~~
## background

* ⚡️ 20 minutes! ⚡️
* ~~purposes of each type of test~~
* ~~integration tests~~
## background

* ⚡ 20 minutes! ⚡

* ~~purposes of each type of test~~

* ~~integration tests~~

* ~~frameworks vs. TDD~~
## background

* ⚡ 20 minutes! ⚡
* ~~purposes of each type of test~~
* ~~integration tests~~
* ~~frameworks vs. TDD~~
* a handful of situational tactics
## background

* ✨ 20 minutes! ✨
* ~purposes of each type of test~~
* ~integration tests~~
* ~frameworks vs. TDD~~
* a handful of situational tactics
* using Jasmine
## background

* ⚡ 20 minutes! ⚡
* ~~purposes of each type of test~~
* ~~integration tests~~
* ~~frameworks vs. TDD~~
* a handful of situational tactics
* using Jasmine
* generally applicable
## background

* ⚡ 20 minutes! ⚡

* ~~purposes of each type of test~~

* ~~integration tests~~

* ~~frameworks vs. TDD~~

* a handful of situational tactics

* using Jasmine

* generally applicable-ish
## background

* ⚡️ 20 minutes! ⚡️
* ~~purposes of each type of test~~
* ~~integration tests~~
* ~~frameworks vs. TDD~~
* a handful of situational tactics
* using Jasmine
* generally applicable-ish *ymmv*
## syntax
## syntax

### What I don't do
## syntax

### What I don't do

* use Jasmine's (RSpec-like) DSL
describe("Math", function(){
});
describe("Math", function(){
    var subject, result;
});
describe("Math", function()
{
  var subject, result;
  beforeEach(function()
  {
  
  
  
  
  
  
  
  
  
  
  
  


  });

});
describe("Math", function(){
    var subject, result;
    beforeEach(function(){
        subject = new Math();
    });
});
describe("Math", function(){
    var subject, result;
    beforeEach(function(){
        subject = new Math();
    });
    describe("#add", function(){
    });
});
describe("Math", function(){
  var subject, result;
  beforeEach(function(){
    subject = new Math();
  });
  describe("#add", function(){
    beforeEach(function(){
    });
    describe("#add", function(){
      beforeEach(function(){
      });
    });
  });
});
describe("Math", function(){
    var subject, result;
    beforeEach(function(){
        subject = new Math();
    });
    describe("#add", function(){
        beforeEach(function(){
            result = subject.add(4, 5);
        });
    });
});
describe("Math", function(){
    var subject, result;
    beforeEach(function(){
        subject = new Math();
    });
    describe("#add", function(){
        beforeEach(function(){
            result = subject.add(4,5);
        });
        it("adds", function(){
        });
    });
})
describe("Math", function(){
    var subject, result;
    beforeEach(function(){
        subject = new Math();
    });
    describe("#add", function(){
        beforeEach(function(){
            result = subject.add(4,5);
        });
        it("adds", function(){
            expect(result).toEqual(9);
        });
    });
});
## syntax

### What I don't do

* use Jasmine's (RSpec-like) DSL
* write my specs with JavaScript
I don't believe you.
## syntax

### What's the problem?
## syntax

### What's the problem?

* Jasmine DSL is not obvious
describe('thing', function(){});
describe('thing', function(){}

beforeEach(function(){});
describe('thing', function(){}

beforeEach(function(){}

afterEach(function(){}

describe('thing', function(){});

beforeEach(function(){});

afterEach(function(){});

it('does stuff', function(){});
describe('thing', function(){

beforeEach(function(){

afterEach(function(){

it('does stuff', function(){

expect(true).toBeTruthy();

});

});

});

});
describe('thing', function(){
  beforeEach(function(){
  
  afterEach(function(){
    
  it('does stuff', function(){
    expect(true).toBeTruthy();
  
  this.addMatchers({});

}
describe('thing', function(){
});

beforeEach(function(){
});

afterEach(function(){
});

it('does stuff', function(){
});

expect(true).toBeTruthy();

this.addMatchers({});

jasmine.createSpy().andCallThrough();
## syntax

### What's the problem?

* Jasmine DSL is not obvious
* test code is verbose, unwieldy
...  

expect(spec).toFinallyEnd();
}};
}};
}};
}};

}};
## syntax

### What's the problem?

- Jasmine DSL is not obvious
- test code is verbose, unwieldy
- those crying mustaches
## syntax

### What's the problem?

* Jasmine DSL is not obvious
* test code is verbose, unwieldy
* those crying mustaches 😞
## syntax

### What I do
## syntax

### What I do

* write specs in CoffeeScript
describe("Math", function(){
    var subject, result;
    beforeEach(function(){
        subject = new Math();
    });
    describe("#add", function(){
        beforeEach(function(){
            result = subject.add(4,5);
        });
        it("adds", function(){
            expect(result).toEqual(9);
        });
    });
});
describe "Math", ->
  beforeEach ->
    @subject = new Math()

describe "#add", ->
  beforeEach ->
    @result = @subject.add(4,5)

  it "adds", ->
    expect(@result).toEqual(9)
CoffeeScript basics*
CoffeeScript basics*

*Fear not, it's just JS.
var add = function(a,b) {
  return a + b;
};
add = (a, b) ->
    a + b
this.save();
@save()
var self = this;
save(function(){
    self.display("Yay!");
});
save =>

@display("Yay!")
### What I do

* write specs in CoffeeScript
* use the *-given DSL
describe "Math", ->
  beforeEach ->
    @subject = new Math()

describe "#add", ->
  beforeEach ->
    @result = @subject.add(4,5)

  it "adds", ->
    expect(@result).toEqual(9)
describe "Math", ->
  Given -> @subject = new Math()

describe "#add", ->
  When -> @result = @subject.add(4,5)
  Then -> @result == 9
describe("Math", function(){
    var subject, result;
    beforeEach(function(){
        subject = new Math();
    });
    describe("#add", function(){
        beforeEach(function(){
            result = subject.add(4,5);
        });
        it("adds", function(){
            expect(result).toEqual(9);
        });
    });
});
describe "Math", ->
  Given -> @subject = new Math()

describe "#add", ->
  When -> @result = @subject.add(4,5)
  Then -> @result == 9
## syntax

### What I do

* write specs in CoffeeScript
* use the *-given DSL
* jasmine-given ported rspec-given
## syntax

### What I do

* write specs in CoffeeScript
* use the *-given DSL
* `jasmine-given` ported `rspec-given`
* `mocha-given` ported `jasmine-given`
## test runner
## test runner

### What I don't do
## test runner

### What I don't do

* default plain HTML test runner
## test runner

### What I don't do

* default plain HTML test runner
* jasmine-maven-plugin
## test runner

### What I don't do

* default plain HTML test runner
* jasmine-maven-plugin
* jasmine-rails
## test runner

### What I don't do

- default plain HTML test runner
- jasmine-maven-plugin
- jasmine-rails
- any server-side-dependent plugin
## test runner

### What's the problem?
## test runner

### What's the problem?

* feedback isn't fast enough
## test runner

### What's the problem?

* feedback isn't fast enough
* build tools treat JS as 2nd-class
## test runner

### What's the problem?

* feedback isn't fast enough
* build tools treat JS as 2nd-class
* friction of server-side coupling
### What I do
## test runner

### What I do

* use Testem as my runner
## test runner

### What I do

* use Testem as my runner
* use Lineman to build my code
## test runner

### What I do

* use Testem as my runner
* use Lineman to build my code
* runs tests in < 300ms on each file change
TEST'EM 'SCRIPTS!
Open the URL below in a browser to connect.
http://localhost:7357/

Chrome 34.0
0/0 ✓

No tests were run :(}

[Press ENTER to run tests; q to quit]
## ajax & ui events
## ajax & ui events

### What I don't do
## ajax & ui events

### What I don't do

* start a fake server that can stub & verify XHRs
## ajax & ui events

### What I don't do

* start a fake server that can stub & verify XHRs
* monkey-patch the browser's XHR facilities
## ajax & ui events

### What I don't do

- start a fake server that can stub & verify XHRs
- monkey-patch the browser's XHR facilities
- invoke code by triggering UI events
## ajax & ui events

### What's the problem?
## ajax & ui events

### What's the problem?

* too integrated for unit tests
SAFE
(Smoke, Acceptance, Feature, End-to-End tests)
Test the system by using it just as a real user would so you can change implementation details with confidence you won't see false negatives.

NOOPE
(Functional, Integrated/Integration, Regression tests)
Don't write tests here. They'll be both coupled to your implementation and divorced from reality. Don't write or use testing tools that rely on your application's coupling to Rails, Ember, Angular, or any other application library or framework.

Discovery
(Unit tests)
TDD new units that you own, build adapters to separate you from third party libs and application frameworks

The Testing Pyramid, 2014 Edition
## ajax & ui events

### What's the problem?

* too integrated for unit tests
* test pain isn't actionable
  (e.g. “only mock what you own”)
## ajax & ui events

### What's the problem?

* too integrated for unit tests
* test pain isn't actionable (e.g. “only mock what you own”)
* raises concerns better handled by integrated tests
## ajax & ui events

### What's the problem?

* too integrated for unit tests
* test pain isn't actionable (e.g. “only mock what you own”)
* raises concerns better handled by integrated tests
* no pressure to improve private APIs
## ajax & ui events

### What I do
## ajax & ui events

### What I do

* wrap native/3rd party libs with objects I own
## ajax & ui events

### What I do

* wrap native/3rd party libs with objects I own
* in unit tests, mock wrappers away
## ajax & ui events

### What I do

* wrap native/3rd party libs with objects I own
* in unit tests, mock wrappers away
* test pain? -> improve wrapper API
## ajax & ui events

### What I do

* wrap native/3rd party libs with objects I own
* in unit tests, mock wrappers away
* test pain? -> improve wrapper API
* wrappers specify our dependence
## ajax & ui events

### What I do

* wrap native/3rd party libs with objects I own
* in unit tests, mock wrappers away
* test pain? -> improve wrapper API
* wrappers specify our dependence
* don’t (typically) test wrappers
TEST'EM 'SCRIPTS!
Open the URL below in a browser to connect.
http://localhost:7357/

Chrome 34.0
0/0 ✔

No tests were run :(

[Press ENTER to run tests; q to quit]
## asynchronous code
## asynchronous code

### What I don't do
## asynchronous code

### What I don't do

* write async tests for async code
## asynchronous code

### What I don't do

* write async tests for async code

```javascript
unit
```
## asynchronous code

### What's the problem?
## asynchronous code

### What's the problem?

* test yields execution control
## asynchronous code

### What's the problem?

* test yields execution control
* lots of noise in test setup
## asynchronous code

### What's the problem?

* test yields execution control
* lots of noise in test setup
* speed/timeout concerns
## asynchronous code

### What's the problem?

* test yields execution control
* lots of noise in test setup
* speed/timeout concerns
* hard to debug race conditions
## asynchronous code

### What's the problem?

* test yields execution control
* lots of noise in test setup
* speed/timeout concerns
* hard to debug race conditions
* many reasons for tests to fail
## asynchronous code

### What I do
## asynchronous code

### What I do

* only write async APIs when useful
### asynchronous code

#### What I do

- only write async APIs when useful
- extract callbacks out of app code and into decorators and mixins
## asynchronous code

### What I do

* only write async APIs when useful
* extract callbacks out of app code and into decorators and mixins
* consider promises over callbacks
## asynchronous code

### What I do

* only write async APIs when useful
* extract callbacks out of app code and into decorators and mixins
* consider promises over callbacks
* use **jasmine-stealth** to capture & discretely test callback functions
TEST 'EM SCRIPTS!
Open the URL below in a browser to connect.
http://localhost:7357/

No tests were run :(
## the dom
## the dom

### What I don't do
## the dom

### What I don't do

* say "(° 뿌°).Writer:iframe" and avoid testing DOM interactions
You can test anything with functions
## the dom

### What I don't do

* say "(\(\text{ʃ}^\text{・}_{\text{□}^\text{・}}\)) \text{ʃ} \text{ʃ} <\text{table/}>\text{ʃ}" and avoid testing DOM interactions
* use HTML fixture files
## the dom

### What I don't do

* say "(emiah) ▽ ▼<table/>▼" and avoid testing DOM interactions

* use HTML fixture files
## the dom

### What's the problem?
#### not testing DOM interactions
## the dom

### What's the problem?
#### not testing DOM interactions

* most JavaScript _is_ DOM my UI code
## the dom

### What's the problem?

#### not testing DOM interactions

* most JavaScript _is_ DOMy UI code
* low coverage limits suite's value
## the dom

### What's the problem?
#### not testing DOM interactions

- most JavaScript *is* DOM-y UI code
- low coverage limits suite's value
- you'll write *more* DOM-heavy code via path of least resistance
## the dom

### What's the problem?
#### not testing DOM interactions

* most JavaScript _is_ DOMy UI code
* low coverage limits suite's value
* you'll write *more* DOM-heavy code via path of least resistance
* hampers true outside-in TDD
## the dom

### What's the problem?
#### using HTML fixture files
the dom

What's the problem?

using HTML fixture files

* large input -> larger everything
## the dom

### What's the problem?

#### using HTML fixture files

* large input -> larger everything
* tests should push for small units
## the dom

### What's the problem?

#### using HTML fixture files

- large input -> larger everything
- tests should push for small units
- sharing fixtures leads to a "_Tragedy of the Commons_"
## the dom

### What I do
## the dom

### What I do

* treat the DOM like a 3rd-party dependency, minimizing exposure
## the dom

### What I do

* treat the DOM like a 3rd-party dependency, minimizing exposure
* create HTML fixtures inline with jasmine-fixture
## the dom

### What I do

* treat the DOM like a 3rd-party dependency, minimizing exposure
* create HTML fixtures inline with `jasmine-fixture`
* arrive at single-purpose DOM-aware functions
TEST "EM" SCRIPTS!
Open the URL below in a browser to connect.
http://localhost:7357/

Chrome 34.0
0/0 ✓

No tests were run :(
## less tactically

> Know _why_ you’re testing.
## less tactically

> Know *why* you’re testing.

Push through the pain before deciding what is worth testing.
## less tactically

> Know *why* you’re testing.

Push through the pain before deciding what is worth testing.

Easy-to-test code is easy-to-use.
Most JavaScript is hard-to-test.
## less tactically

> Know _why_ you’re testing.

Push through the pain before deciding what is worth testing.

Easy-to-test code is easy-to-use. Most JavaScript is hard-to-test.
## less tactically

> Know _why_ you’re testing.

Push through the pain before deciding what is worth testing.

Easy-to-test code is easy-to-use. Most JavaScript is hard-to-test.

There’s no Right Way™ in software, just thoughtful and thoughtless approaches.
Get up and running in minutes!

* [Docs](http://linemanjs.com)
* [Help](http://twitter.com/linemanjs)
My name is Justin Searls
Please tweet me @searls &
Say hello@testdouble.com