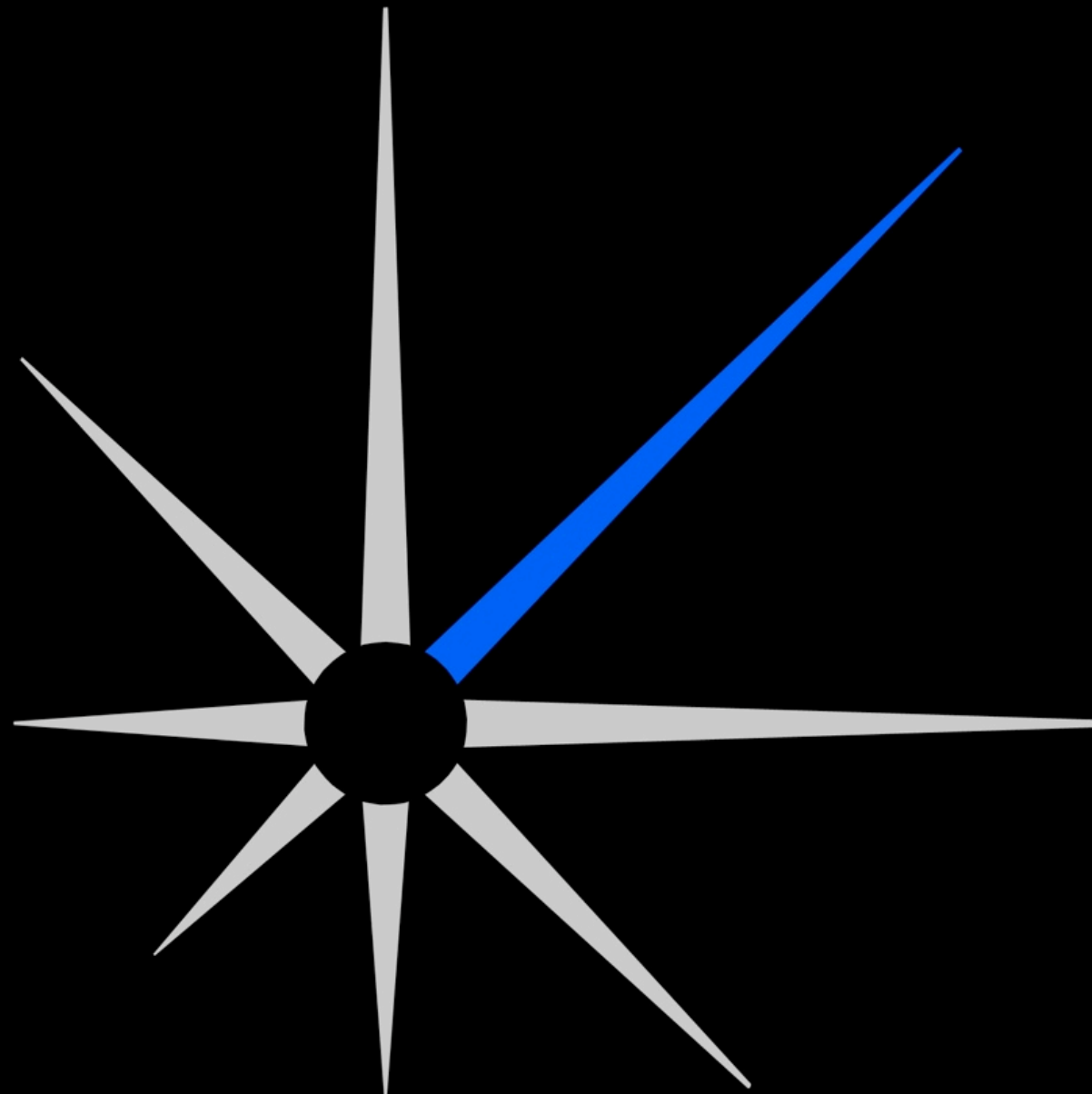


TDD and JavaScript



8th light

Jim Suchy

Software Craftsman

jsuchy@8thlight.com || @jsuchy

Manifesto for Software Craftsmanship

Raising the bar.

As aspiring Software Craftsmen we are raising the bar of professional software development by practicing it and helping others learn the craft. Through this work we have come to value:

Not only working software,
but also **well-crafted software**

Not only responding to change,
but also **steadily adding value**

Not only individuals and interactions,
but also **a community of professionals**

Not only customer collaboration,
but also **productive partnerships**

That is, in pursuit of the items on the left we have found the items on the right to be indispensable.

© 2009, the undersigned.
this statement may be freely copied in any form,
but only in its entirety through this notice.

Craftsmen value well-crafted software.

How do I ensure I'm creating well-crafted software?

Testing, and TDD in particular, helps me to produce well-crafted software.

Test Driven Development



Warm and Fuzzy!
Does anyone practice TDD?

Benefits

- 0) Eliminates the fear of changing code
- 1) The system should be executable at all times
- 2) We avoid using a debugger
- 3) We build a suite of regression tests, allowing us to refactor without fear
- 4) Documents how the system works in the best way; the tests cannot lie
- 5) All of your code is testable, and therefore decoupled, leading to better design just about every time

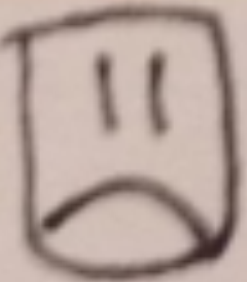
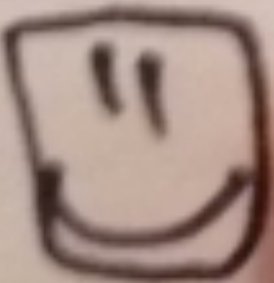

Uncle Bob's Three Laws ①

1. Write NO production code except to pass a failing test.
2. Write only enough of a test to demonstrate a failure
3. Write only enough production code to pass the test.

- 1) You are not allowed to write any production code unless it is to make a failing unit test pass.
- 2) You are not allowed to write any more of a unit test than is sufficient to fail. Compilation failures are failures.
- 3) You are not allowed to write any more production code than is sufficient to pass the one failing unit test.

Core of TDD

③

- **RED**: test fails 
- **GREEN**: test passes 
- **REFACTOR**
↳ CLEAN Code + Tests 

Wash, rinse, repeat.

These cycles are meant to be short, a minute or two.

This rhythm makes programming fun. You always have a goal that can be accomplished in the next minute or two.

If you are pair programming, you can use this cycle as a natural way to pass the keyboard.

So TDD is great, makes my job fun. Now I'm coding away, blissfully TDDing my RoR application... when I need some dynamic behavior in my web application.

JavaScript === Evil?



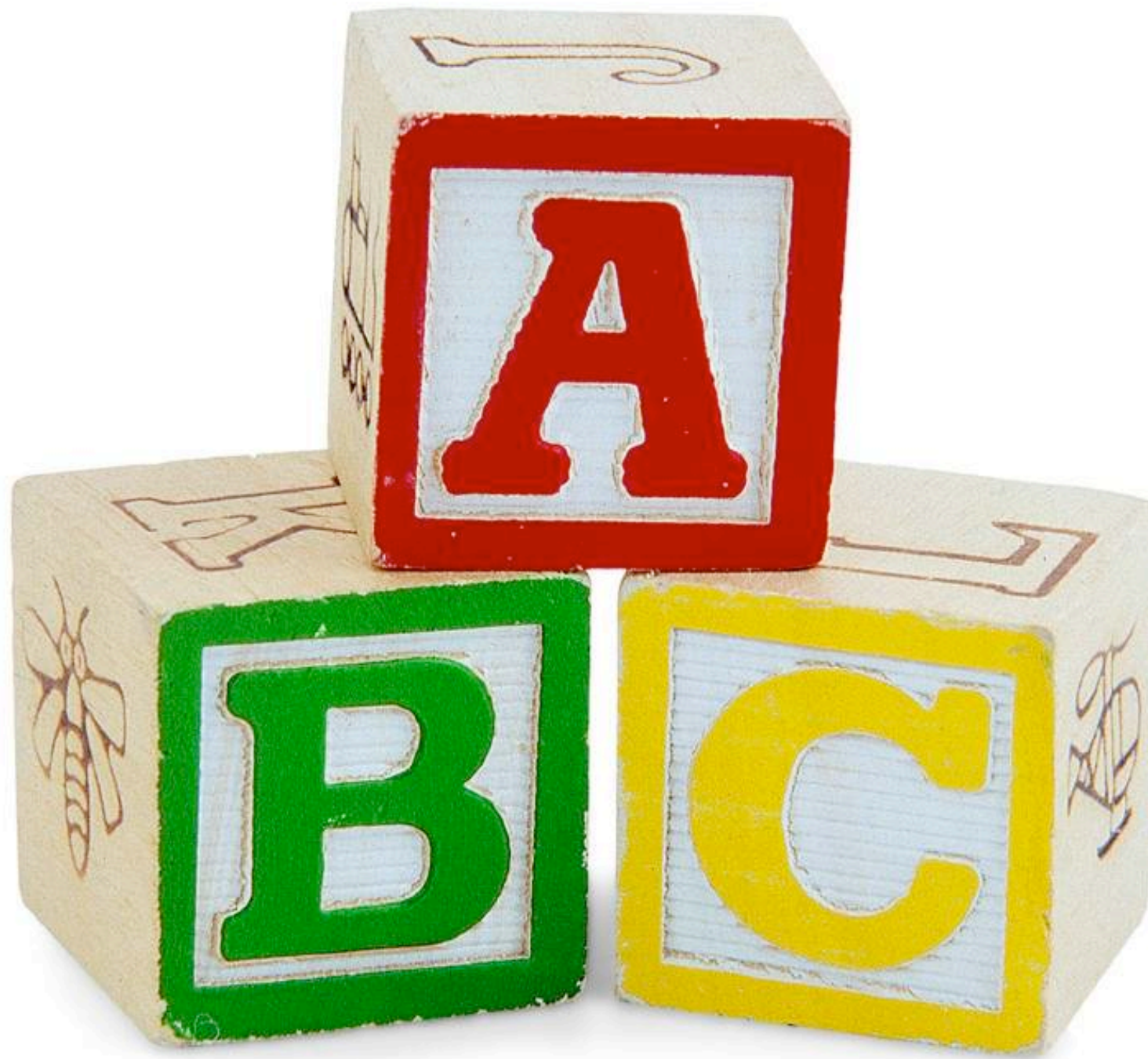
Evil? Bad? Or just misunderstood?
How many hate JavaScript? I used to
Some poor design decisions!

There are some bad parts of the language.



Global Variables

Global variables are evil. They are visible in every scope, and can be changed by any part of the program.
Used for linkage; all compilation units are loaded into a common global object.



Lack of Block Scope

Variables declared in a block are not visible outside the block. Not so in JavaScript, even though the syntax is telling you that it should.

Semicolon Insertion

```
return  
{  
  foo: “bar”  
}
```

JavaScript wants to help you out here. This code will return undefined, not an object with a foo member.

=== vs ==



EQUALITY

Demons. Nazis. Ninjas. Zombies.
One size fits all.

If operands are of different types, == and != attempt to coerce the values using complicated and unmemorable rules.
Just don't use == and !=

So there are all of these bad features in JavaScript. What's the effect?

Buggy JavaScript Code!



Is this the real reason we hate JavaScript though?

Why do we really hate
JavaScript?

Working with the DOM sucks!

But working with the DOM would suck in any language.
There are frameworks (jQuery, Prototype, et al) that hide this ugliness. Use them!

It's (perceived to be) difficult to test...

We're tied to a browser.
We have debugging tools; but I don't want to be debugging JavaScript!

Forces us to leave the
comfort of our
language of choice.

Ruby, C#, VB.NET, ASP, PHP, whatever...
Frameworks like Rails (w/ RJS) tries to even hide JavaScript from developers
I think that this is the wrong approach to JavaScript.

JavaScript has many good parts!



<Show book's thickness>
There are some really great features!
Let's take a look at some of them.

Functions



Functions are first class objects.
First lambda / functional language to really go mainstream.

Dynamic Language with Loose Typing



- 1) Avoid complex class hierarchies
- 2) Avoid casting
- 3) Add methods on the fly!

Expressive object literal notation

```
var javascriptGuru = {  
  firstName: "Douglas",  
  lastName: "Crockford"  
}
```

JSON has become a standard, and that uses this notation

Portable



Every computer with a browser can run it!

So there are some good and bad parts of JavaScript. How do we mitigate those bad parts?
How do we keep our code clean, decoupled and bug-free?

Are you testing your JavaScript?



Are you test driving your javascript?
Green band burns when not testing my JS!

There are a ton of JavaScript testing frameworks available

This suggests to me that there are many people trying to find a better way to test their JavaScript.

RhinoUnit



[RhinoUnit](#) is run from an ANT scriptdef task using the Rhino engine - and uses all the helpful things that ANT provides for that.

The biggest advantage I see in using [RhinoUnit](#) is that it easily runs as part of the build, and runs very fast.

ensure that a function has been called (by wrapping it with `assert.mustCall()`)

ensure that the global namespace isn't polluted by poor variable scoping

JSLint has been included to ensure that your Javascript files follow (Douglas Crockford's) best practices.

YUI Test



testing framework for browser-based JavaScript solutions, based on xUnit

Asynchronous tests for testing events and Ajax communication.
DOM Event simulation

ScrewUnit

```
describe("a nested describe", function() {
  var invocations = [];

  before(function() {
    invocations.push("before");
  });

  describe("a doubly nested describe", function() {
    before(function() {
      invocations.push('inner before');
    });

    it("runs before in all ancestors prior to an it", function() {
      expect(invocations).to(equal, ["before", "inner before"]);
    });
  });
});
```

RSpecy!
A Javascript BDD Framework with nested describes, a convenient assertion syntax, and an intuitive test browser

JSUnit



Unit Testing framework for client-side (in-browser) JavaScript. It is essentially a port of [JUnit](#) to JavaScript.

(another) JSUnit



Simple framework to write repeatable tests in JavaScript. It is an instance of the xUnit architecture for unit testing frameworks. JSUnit is a port of JUnit 3.8.1

JsUnitTest

Blue Ridge

Cross Check

Test Case

The list goes on...

A craftsman builds his
own tools

Let's write our own!

<http://github.com/jsuchy/JsSimpleUnit/tree/master>

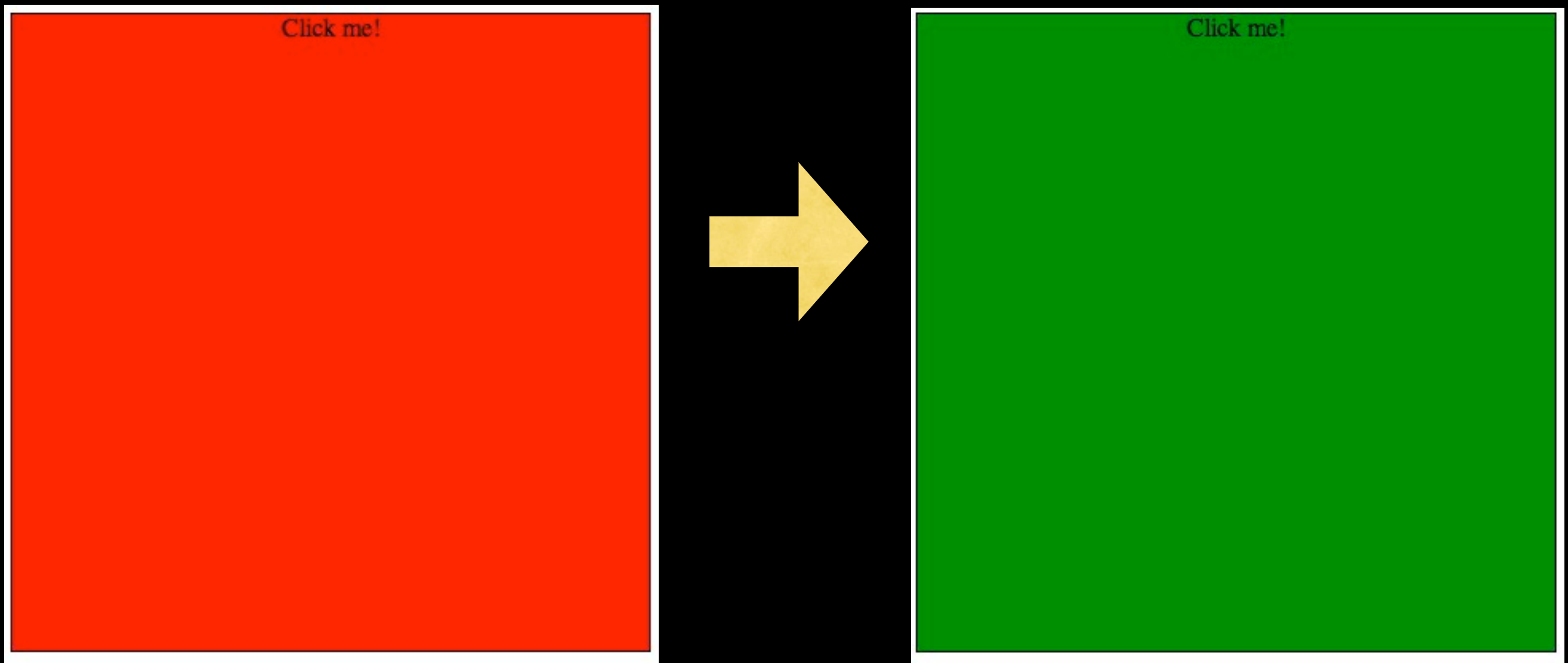
Why?

I wanted to prove that JavaScript is inherently easier to test than one may think.

I want something really simple; I just need an AssertEquals

Go to code here. Test-drive the creation of simple testing framework.

Simple Use Case



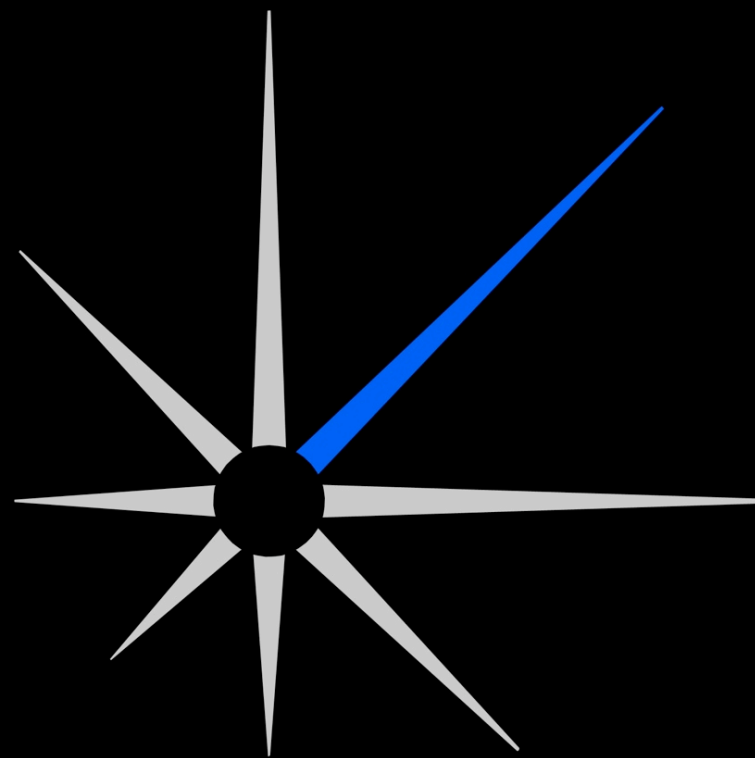
You can follow along at
<http://github.com/jsuchy/TddJavaScriptAgile2009>



Software CRAFTSMANSHIP north america

Tomorrow!
Across the street at the Swissotel

Thank you!



8th light

Jim Suchy

jsuchy@8thlight.com || @jsuchy