



We
code
in
PEACE

DEVOXX™
the java™ community conference



Javascript Unit Testing & Build Integration



Wouter Groeneveld

Agile software developer at Cegeka

wouter.groeneveld@gmail.com

Twitter: @cegeka



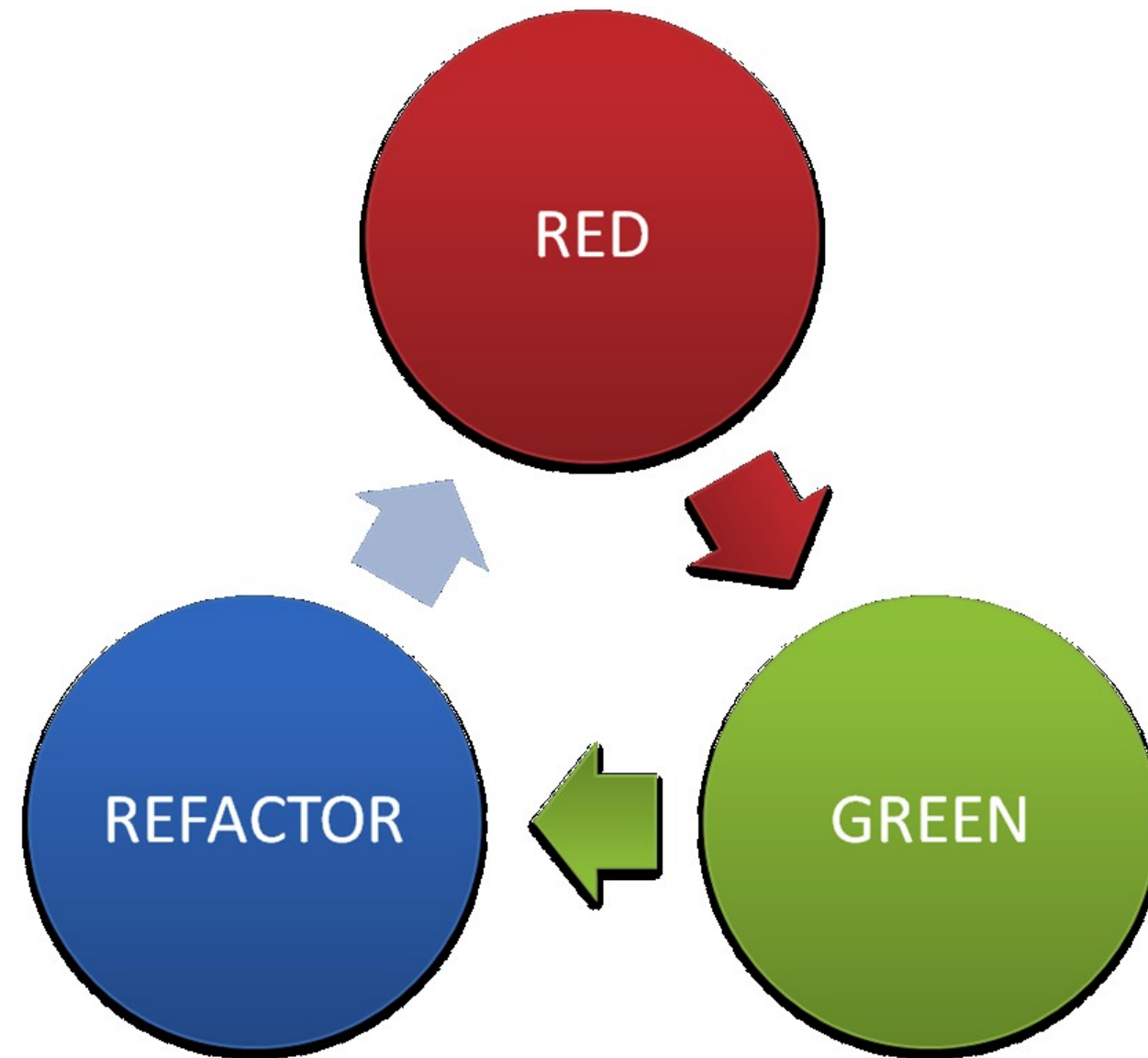
Wouter Groeneveld

Who am I?

Agile software developer @ Cegeka for 5 years;

- a.k.a. dynamic languages fanboy
- a.k.a. (alas...) Javascript guru
(taught various JS & engineering courses, I used to be just the “expert”)
- Likes to inspire others and improve himself.

Working with JavaScript

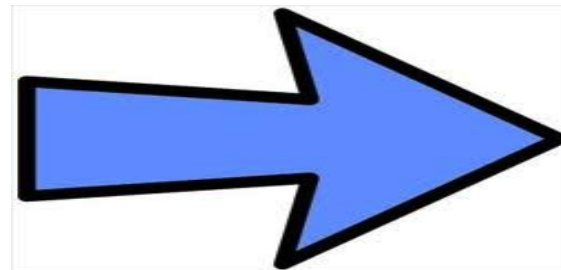
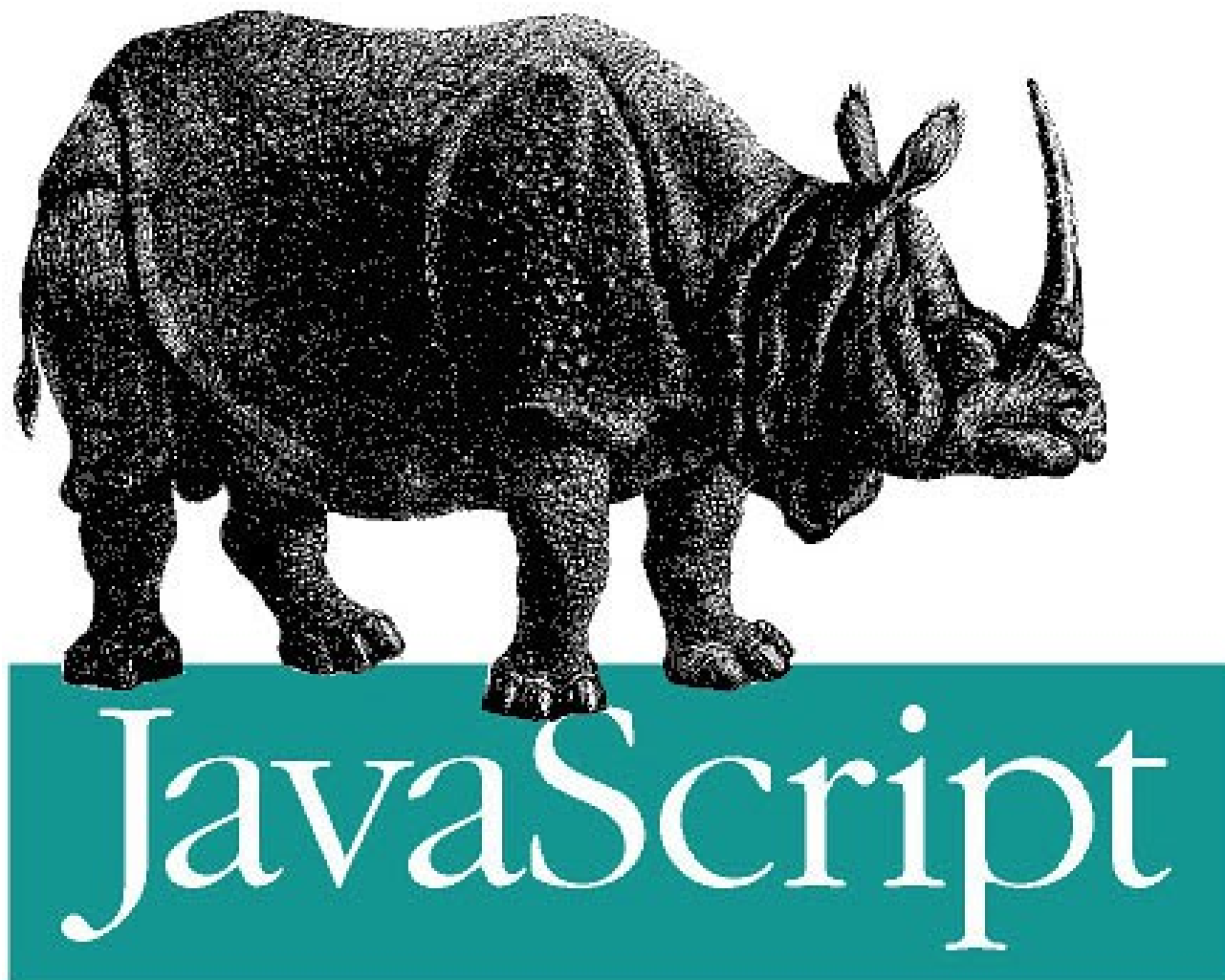


JS <> Java

- JS is a dynamic language
- JS has functions as first-class citizens
- JS' callback mechanics works different
- JS does not have block level scoping
- JS does not have classes (prototypal inheritance)
- ...

JS <> Java

- Something **NEW** that we're **AFRAID** of?



JS <> Java

Things that make it even harder to **test** properly:

- Lack of *standard* (browser quirks, “the one” testing/DOM/...API, ...)
- Lack of ***integration*** with usual build cycle
- *How do I... @Test in JS?* `mvn clean install?` ...

So “doing JS” equals...

- “let's get it over with”?
- Nasty, completely **untestable code**
- UI & Domain **interleaved**

All violating separation of concerns?



PLANTS VS. ZOMBIES



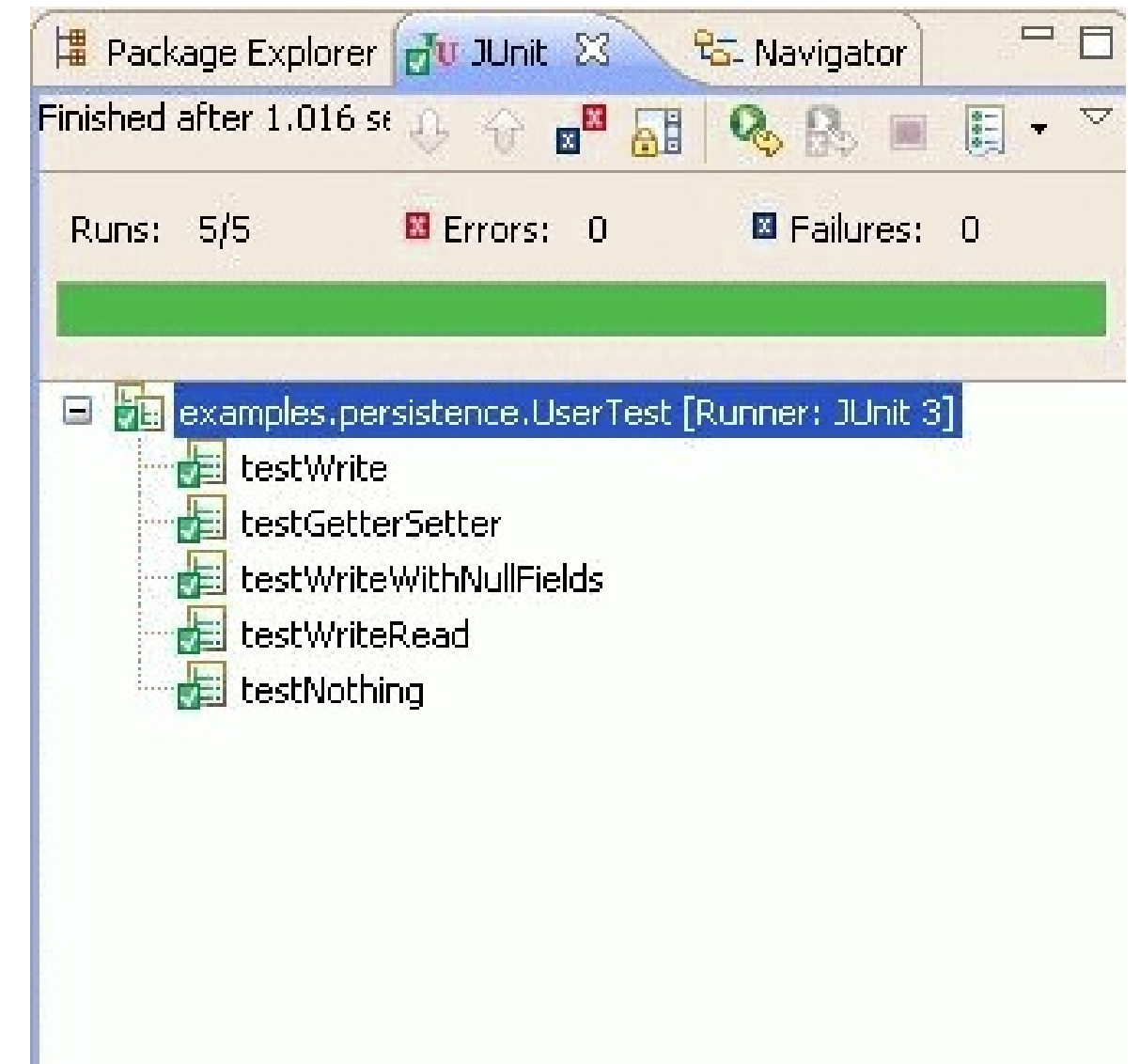
Zombies (very ugly)

```
$(function() {  
    // what's this counters?  
    var zombie = 3;  
    var plant = 0;  
  
    function updateHealth() {  
        $("#health").html(zombie);  
        // domain logic alert  
        if(zombie <= 0) {  
            $("#zombie").hide("slow");  
        }  
    }  
    ...  
});
```

JS In need of the TDD cycle

How can we **improve** upon this?

- **Dynamic** langs even need **more tests!**
(no compile-time errors)
- **Forces** us to think about design
& separation when implementing
- We already **have the skill**
& apply it to Java (or lang. x) everyday?!



Improving zombies (1)

```
var Zombies = (function() {  
  
    var Zombie = {  
        getHit: function(damage) {  
            this.health -= damage;  
        },  
        ...  
    }  
  
    return {  
        Create: function() { return Object.create(Zombie); }  
    }  
})();
```

Improving zombies (1)



What happened?

- Create a **Module** to **test** separately
- **Include** the module in the HTML file
- Encapsulate **logic**, as we would in Java

So now I can **test** “module.js” as a **separate unit**.

Improving zombies (1)



What's a “*module*”? What are *closures*? Aaarghhh ?!!?!?! /bail

<https://github.com/jefklak/javascript-courses>



JavaScript
The Good Parts

But... how? Use Jasmine instead of **JU**nit

```
public class MyTest {  
  
    @Before  
    public void setUp() { ... }  
  
    @Test  
    public void plantShouldGrow() {  
        Plant myPlant = new Plant();  
        myPlant.water();  
        Assert.assertEquals(1, myPlant.size);  
    }  
  
}
```

```
describe("my test suite", function() {  
  
    beforeEach(function() { ... });  
  
    it("should grow when watered", function() {  
        var myPlant = Object.create(Plant);  
        myPlant.water();  
        expect(myPlant.size).toEqual(1);  
    });  
});
```


But... how?

...

// you can create your **own matchers**

```
expect(expression).matcher(arguments);
```

// you can **nest tests** all the way you want

```
describe("module 1", function() {  
  it("should this or that", ...);  
  describe("submodule from 1", function() {  
    it("should do that or whatever", ...);  
  });  
});
```

Output results



Jasmine 1.2.0 revision 1337005947

● ● ● X ● X X ●

Failing 3 specs

8 specs | 3 failing

plants versus zombies! creating zombies should not 'be dead' when created.

Expected true not to be truthy.

Error: Expected true not to be truthy.

```
at new jasmine.ExpectationResult (file:///localhost/Users/jefklak/Dropbox/werk/jscrip/testing/examples/java/src/test/javascript/lib/jasmine.js:102:32)
at null.toBeTruthy (file:///localhost/Users/jefklak/Dropbox/werk/jscrip/testing/examples/java/src/test/javascript/lib/jasmine.js:1194:29)
at null.<anonymous> (file:///localhost/Users/jefklak/Dropbox/werk/jscrip/testing/examples/java/src/test/javascript/specs/moduleSpec.js:30:43)
at jasmine.Block.execute (file:///localhost/Users/jefklak/Dropbox/werk/jscrip/testing/examples/java/src/test/javascript/lib/jasmine.js:1024:15)
at jasmine.Queue.next_ (file:///localhost/Users/jefklak/Dropbox/werk/jscrip/testing/examples/java/src/test/javascript/lib/jasmine.js:2025:31)
at jasmine.Queue.start (file:///localhost/Users/jefklak/Dropbox/werk/jscrip/testing/examples/java/src/test/javascript/lib/jasmine.js:1978:8)
at jasmine.Spec.execute (file:///localhost/Users/jefklak/Dropbox/werk/jscrip/testing/examples/java/src/test/javascript/lib/jasmine.js:2305:14)
at jasmine.Queue.next_ (file:///localhost/Users/jefklak/Dropbox/werk/jscrip/testing/examples/java/src/test/javascript/lib/jasmine.js:2025:31)
at jasmine.Queue.start (file:///localhost/Users/jefklak/Dropbox/werk/jscrip/testing/examples/java/src/test/javascript/lib/jasmine.js:1978:8)
at jasmine.Suite.execute (file:///localhost/Users/jefklak/Dropbox/werk/jscrip/testing/examples/java/src/test/javascript/lib/jasmine.js:2450:14)
```

So... *what* should I test?

What do you think?

42 ?

Everything of course!

- Logic & related behavior
- **UI** & related behavior

Okay, how do I test UI stuff?

Also with  Jasmine

- Use **custom matchers** (for jQuery)
<https://github.com/velesin/jasmine-jquery>
- Use **fixtures**
- (If needed, use **mocks** and **spies**)

What about  ?

Too slow, too cumbersome, not specific enough

Plants - size: **1**



Tools



Improving zombies (2)



What happened?

- Create a **UI Module** to **test** separately
- **Include** the module in the HTML file (we now have 2)
- Test as an “integration” test by using fixtures

Improving zombies (2)

```
it("should kill the zombie after shooting it like hell", function() {  
  
    $("#can").click(); // give the plant some power  
    $("#plant").click().click().click(); // shoot 3 times  
  
    expect($("#zombie")).not.toExist();  
});
```

Back to our integration problem

- Running JS tests in-browser using `SpecRunner.html` is cumbersome
- Generates different output: no JUnit XML Reports
- “one command to test them all”: `mvn test`

m***oven***

option #1: headful

Run `SpecRunner.html` in a *real browser*

```
Jasmine 1.1.0 revision 1330200206
```

```
• •
```

```
Passing 2 specs
```

```
extjs4 unit test example
```

```
  is true
```

```
  should create another button when clicked on 'knopke'
```

= *sloooooow & in need of parsing results (?)*

option #2: headless – Jasmine JS Runner

Run JS specs in JVM & convert results to JUnit equivalent

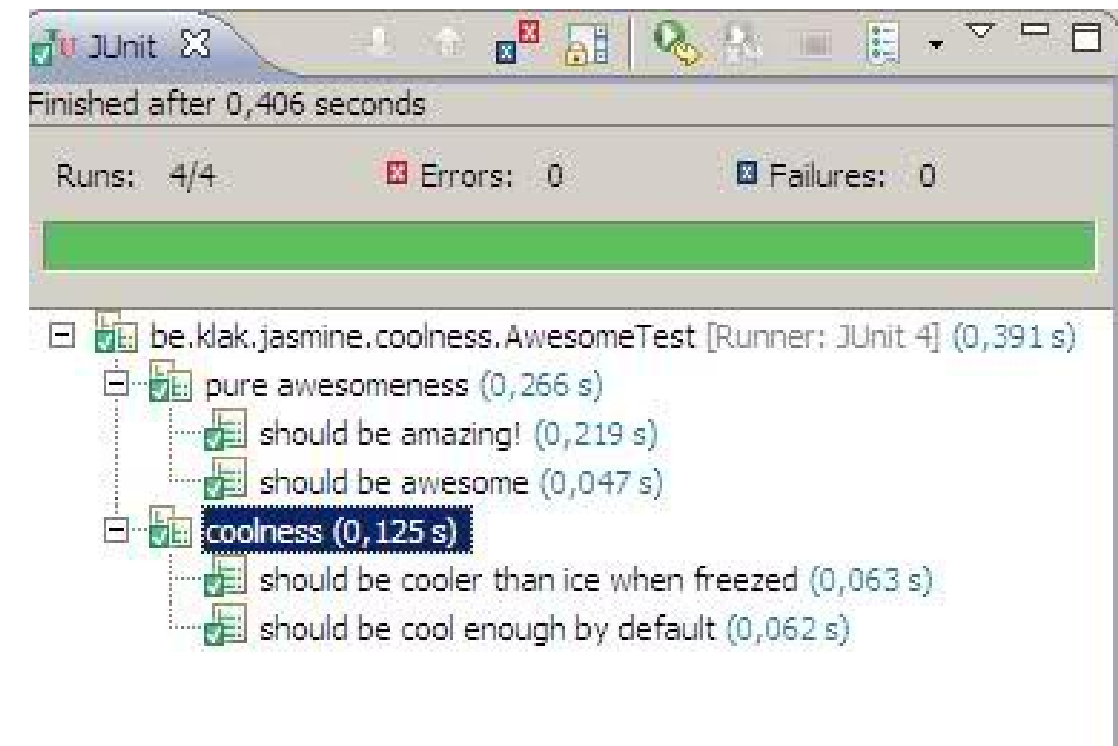
<https://github.com/jefklak/jasmine-junit-runner>

```
describe("pure awesomeness", function() {
  it("should be amazing!", function() {
    expect(stuff).toEqual("amazing");
  });

  it("should be awesome", function() {
    expect(moreStuff).toBe("awesome");
  });
});

describe("coolness", function() {
  it("should be cooler than ice when freezed", function() {
    var coolness = CoolingRepository.beCool();
    coolness.freeze();
    expect(coolness.coolnessRatio).toBe(-100);
  });

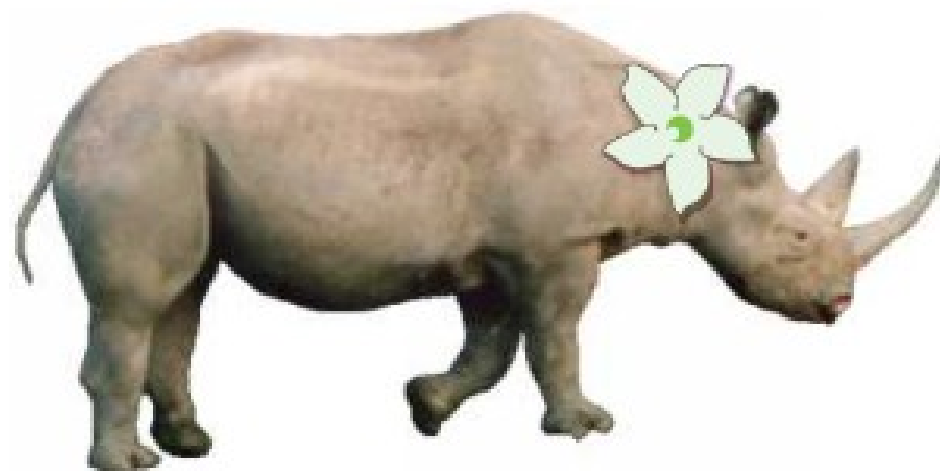
  it("should be cool enough by default", function() {
    expect(CoolingRepository.beCool().coolnessRatio).toBe(-5);
  });
});
```



Option #2 – how does this work?

- Use *Env.js* to emulate `DOM (window)` in JS
- Use *Rhino* to evaluate JS in JVM context
- Parse Jasmine results using a custom reporter
- Convert to JUnit's `Runner` objects; auto-integrates in your IDE

JavaScript BDD



Option #2 – the Java test eq.

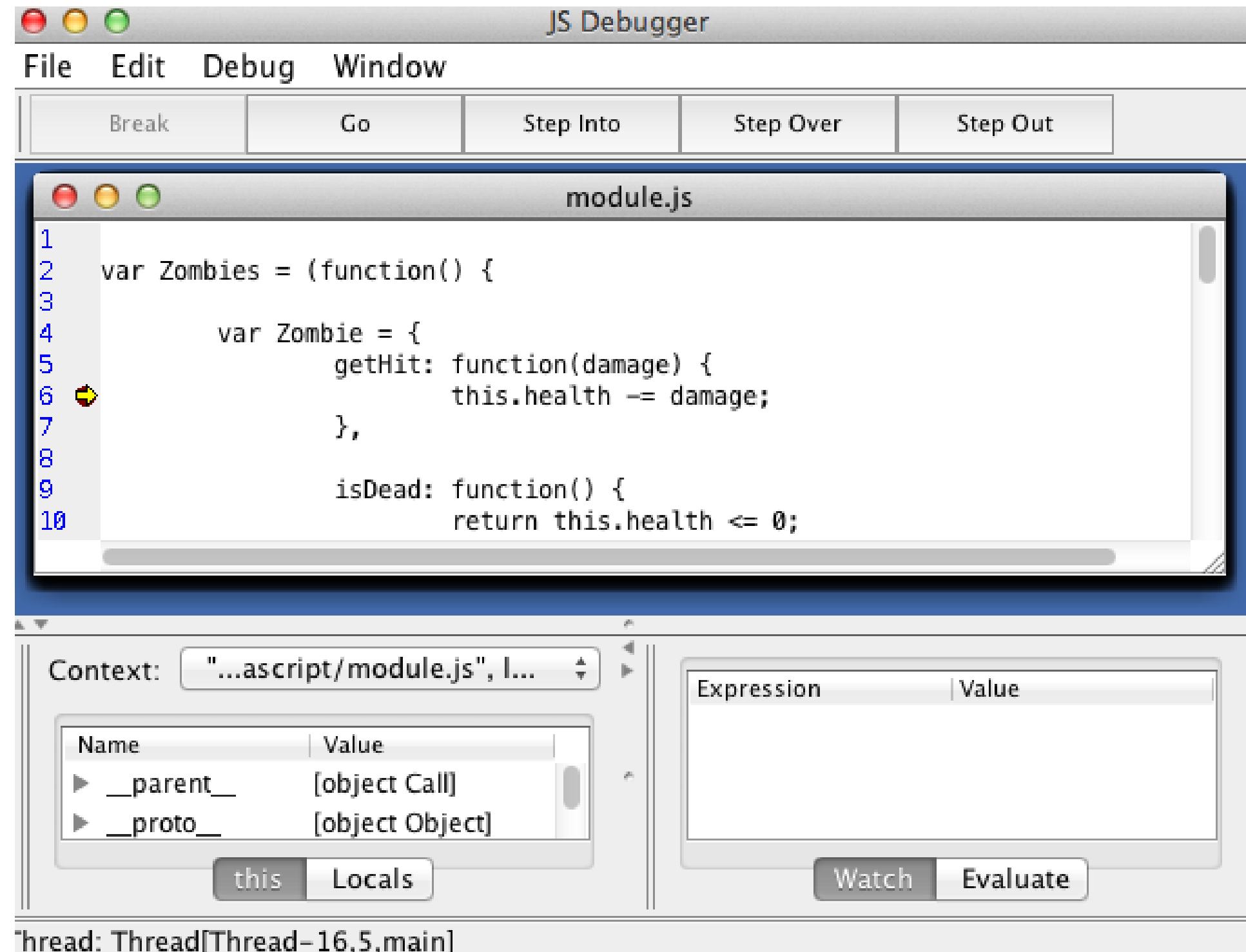


```
@RunWith(JasmineTestRunner.class)
@JasmineSuite(sources = { "module.js" }, sourcesRootDir = "src/main/javascript")
public class ModuleTest {


}
```

Option #2 – “severe” disadvantages

- bad **browser behavior emulation**
- Requires a Java file for every spec file (for now)
- **Debugging?**
@JasmineSuite(
 debug = true)
- Still slow



Option #3 – a “real” headless browser

- Run SpecRunners in 
- Use a custom reporter to rewrite Jasmine results into JUnit XML structure
- Integrate in maven with `exec-maven-plugin`



Option #3 – starting jasmine in phantom

```
page.open(specrunner, function(status) {  
    if (status === "success") {  
        utils.core.waitFor(function() { // wait for this to be true  
            return page.evaluate(function() {  
                return typeof(jasmine.phantomjsXMLReporterPassed) !==  
"undefined";  
            });  
        }, function() { // once done...  
            // Retrieve the result of the tests
```


Option #3 – “severe” disadvantages

- JS runs in sandbox mode within phantom;
no (easy) way to access objects from scope
- so **not usable within JVM** or JUnit directly (IDE)
- **Debugging**? Possibly worse...

Browser quirks can only be tested with



Conclusion

How should I test JS:

- Like you're used to: **test-first** with  Jasmine (instead of JUnit)

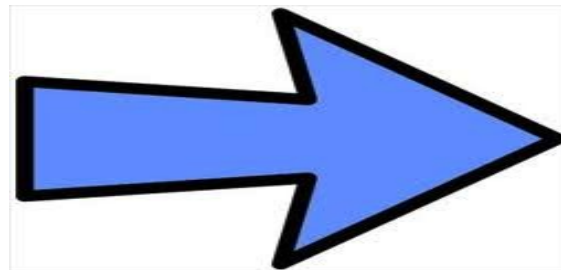
What should I test:

- Domain logic – use modules!
- UI logic – use jQuery & custom matchers
- Browser specific quirks: with acceptance tests!

Conclusion

How should I integrate the whole thing:

- Use a headless browser which produces JUnit XML Reports
- Non-UI tests can also be integrated in IDE with the *jasmine JUnit runner*

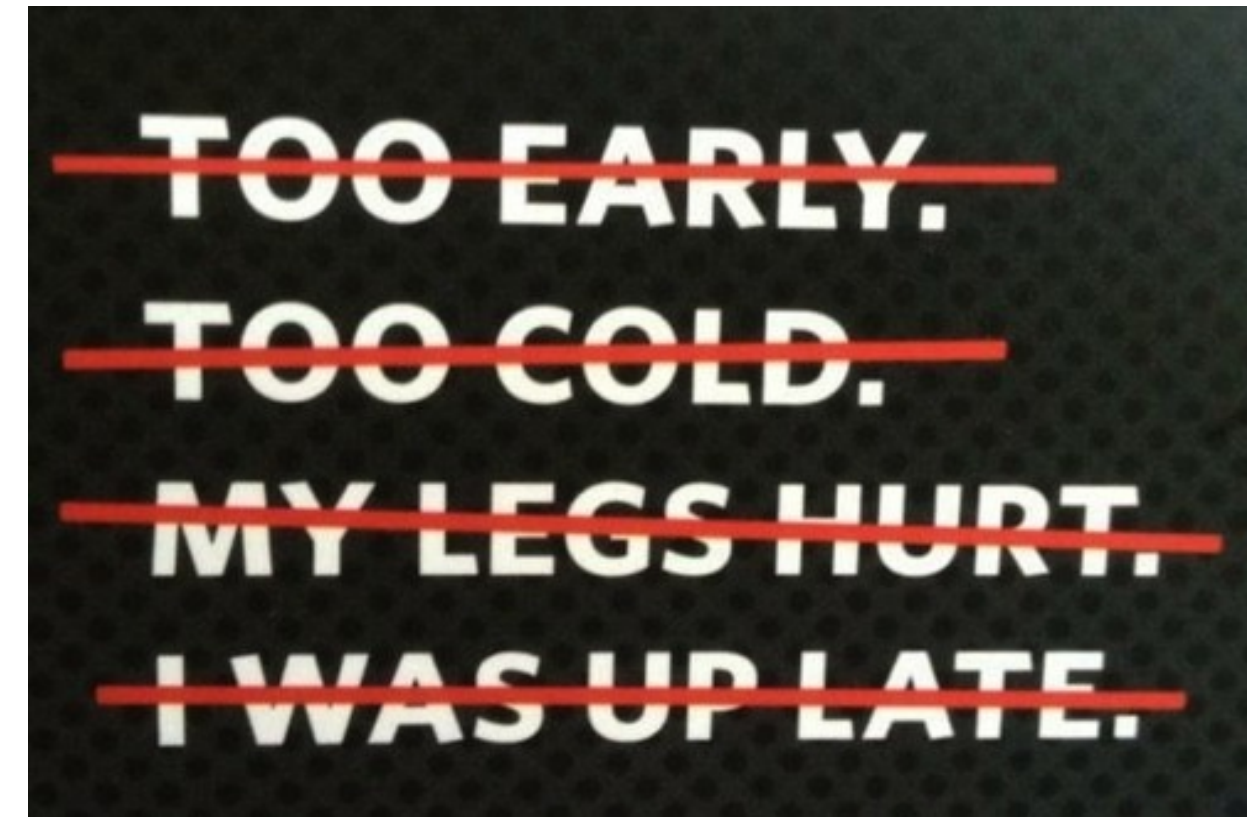


<?xml?>

The “no excuses” slide

These are no valid excuses for not testing your JS stuff:

- I don't know how or what (you do now)
- “But it's only a small event handler” - yeah, so?
- “But my JS has been covered by Selenium already!” - really?
- Browser specific quirks: with acceptance tests!
- But it was late and we had pizza and...





Q&A