

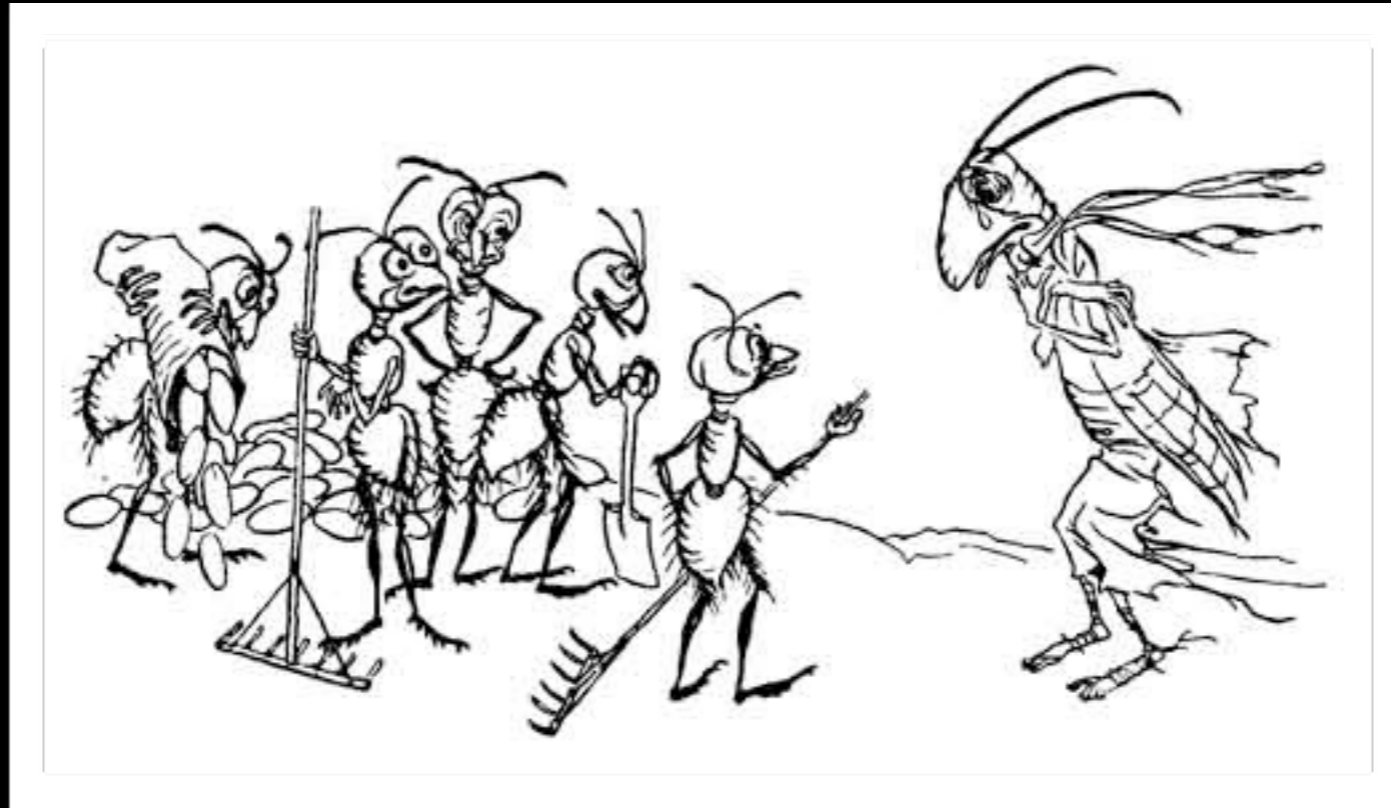
Test Driven Development

Presented by
Clayton Lengel-Zigich

Integrum



Methodology



<http://mythfolklore.net/aesopica/>

In TDD, Computer
Programs you!

```
>> Library.send_late_notices
```

```
>> Library.send_late_notices
```

```
NameError: uninitialized constant
```

```
Library
```

```
>> Library.send_late_notices
NameError: uninitialized constant
Library
>> Library.send_late_notices
```

```
>> Library.send_late_notices
```

```
NameError: uninitialized constant  
Library
```

```
>> Library.send_late_notices
```

```
NoMethodError: undefined method  
`send_late_notices' for Library:Class
```

```
>> Library.send_late_notices
NameError: uninitialized constant
Library

>> Library.send_late_notices
NoMethodError: undefined method
`send_late_notices' for Library:Class

>> Library.send_late_notices
```

```
>> Library.send_late_notices
NameError: uninitialized constant
Library

>> Library.send_late_notices
NoMethodError: undefined method
`send_late_notices' for Library:Class

>> Library.send_late_notices
“sent!”
```

TDD Workflow

TDD Workflow

Red (test)

TDD Workflow

Red (test)

Green (code)

TDD Workflow

Red (test)

Green (code)

Re-factor (improve)

```
# RSpec
describe "Item Checkout" do
  it "should have a status of 'out'" do
    @item.checkout
    @item.status.should == "out"
  end
end
```

Red (test)

```
# RSpec
describe "Item Checkout" do
  it "should have a status of 'out'" do
    @item.checkout
    @item.status.should == "out"
  end
end
```



failblog.org

```
def checkout  
  status = "out"  
end
```

Green (code)

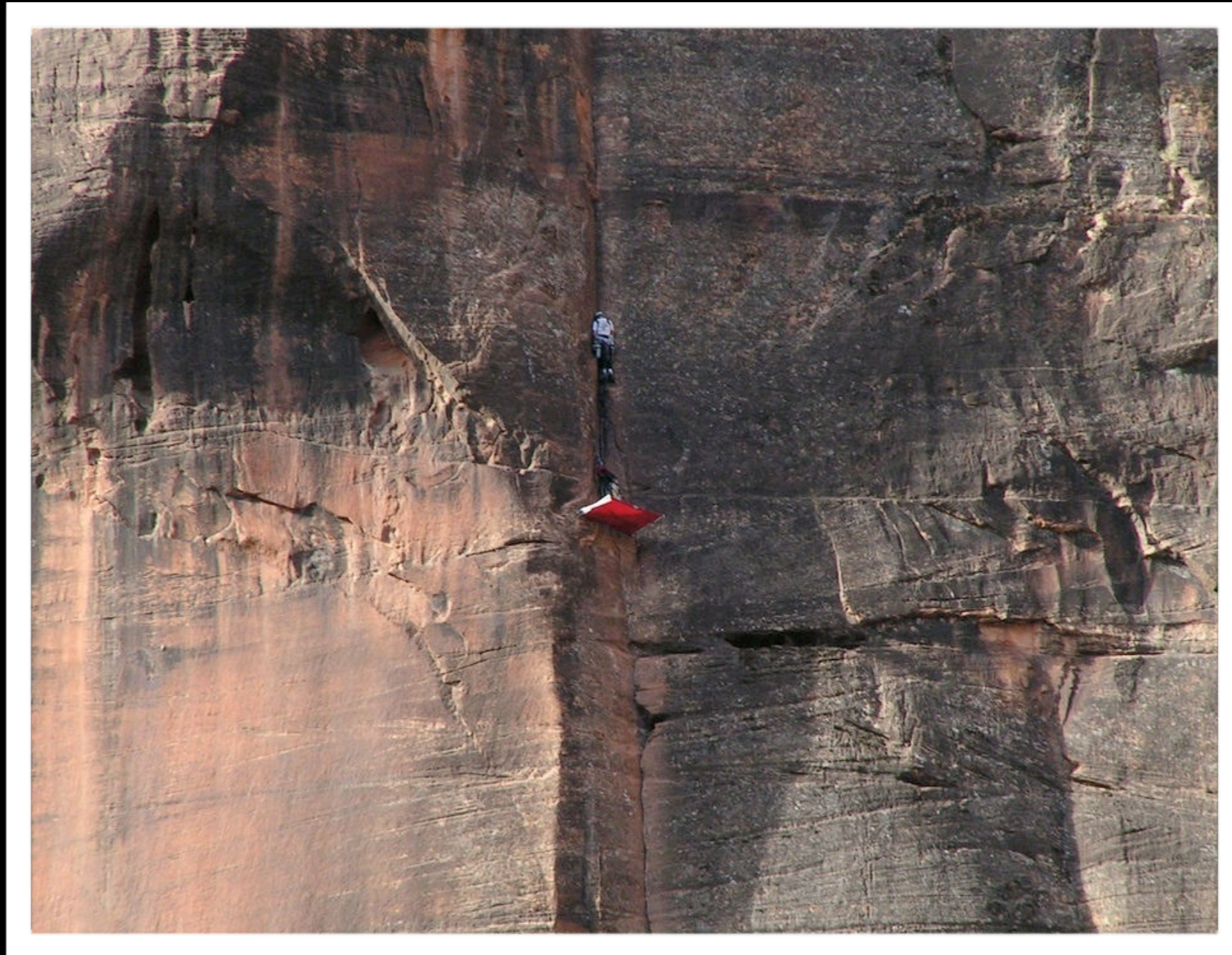
```
def checkout  
  status = "out"  
end
```

PASS

PASS

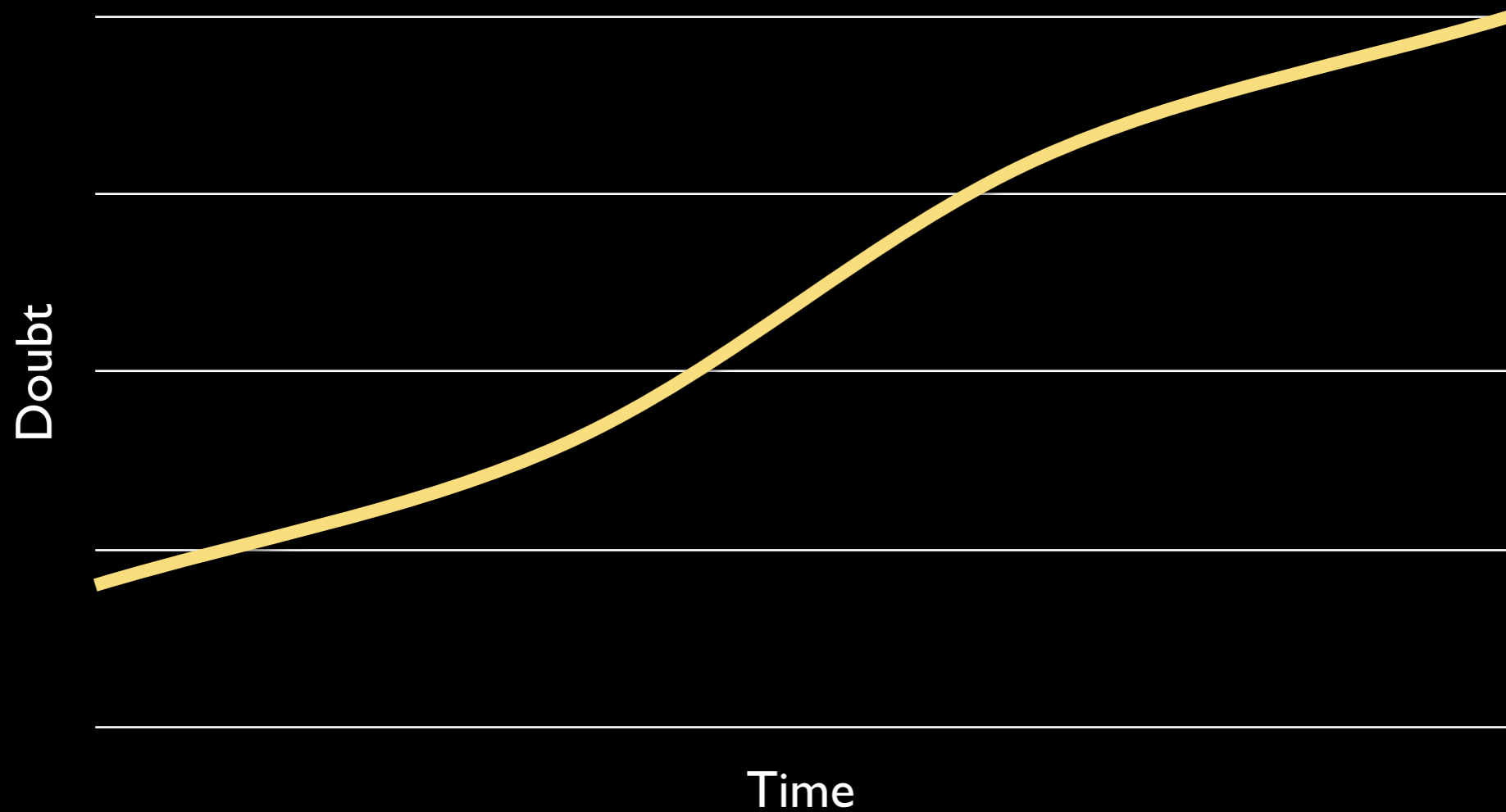
Re-factor (improve)

TDD Benefits

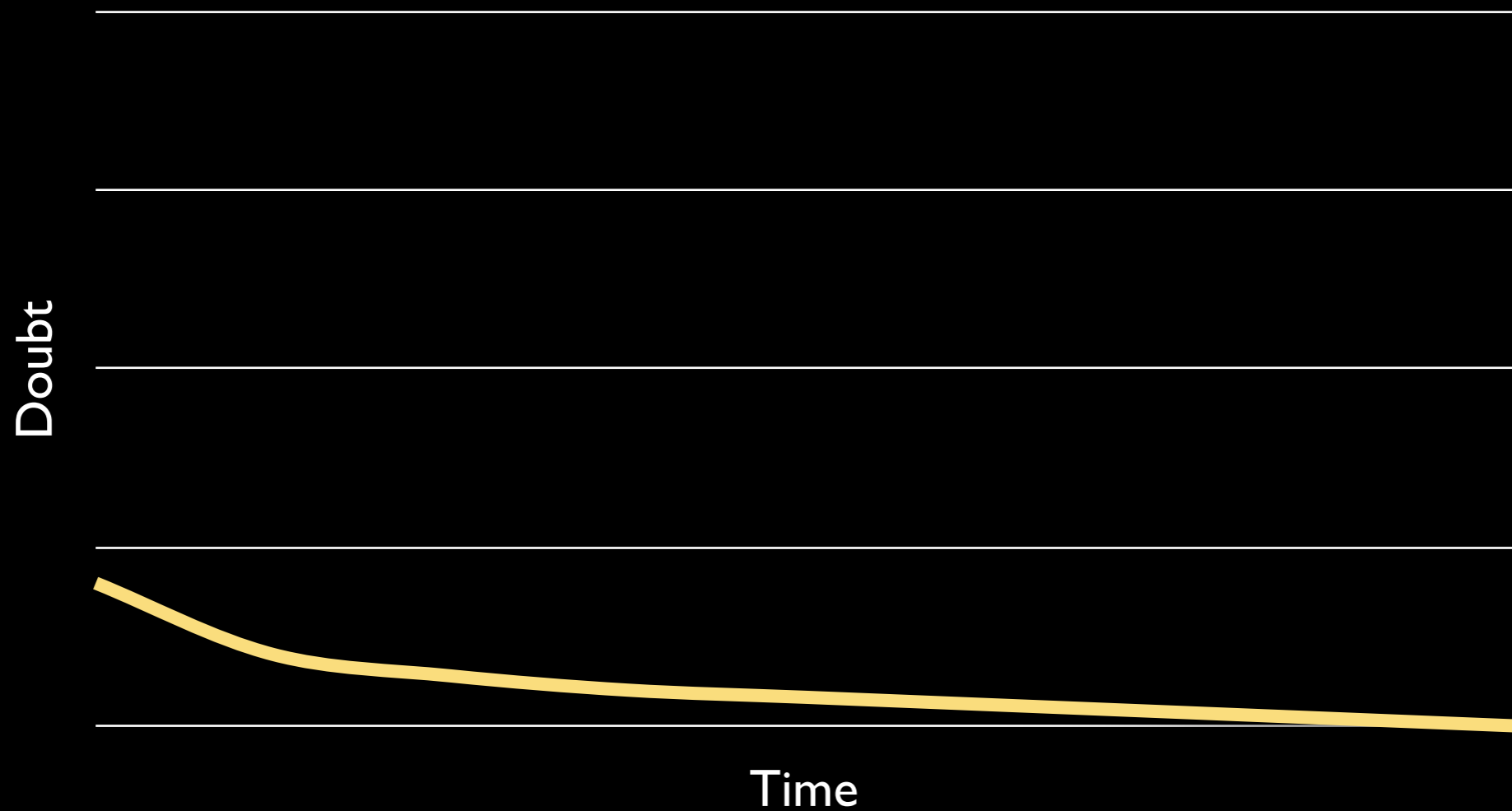


<http://www.flickr.com/photos/slayer23/2133360871/>

Typical TAD Approach



Typical TDD Approach



TDD Eliminates Doubt

TDD Encourages Re-factoring

- Quality Tests
- Immediate Feedback
- Be Creative/Experimental

TDD Pitfalls

- Meaningless Tests
- Brittle Tests
- Complicated Problems

```

describe "A Bad Test" do
  before :each do
    @dog = Dog new
    @dog.stub!(:name).and_return("Fido")
    @dog.stub!(:last_visit).and_return(6.months.ago)

    @vet = mock_model(Vet)
    @doctor = mock_model(Doctor)
    @doctor.stub!(:vet).and_return(@vet)
    @owner = mock_model(Owner)
    @owner.stub!(:preferred_doctor).and_return(@doctor)
    @owner.stub!(:other_pets).and_return([mock_model(Cat), mock_model(Lizard)])
  end
  it "should update the visit, and get shots and thank the doctor" do
    @dog.should_receive(:get_temperature).and_return(98)
    @owner.should_receive(:questions).and_return("everything's normal")
    @doctor.should_receive(:give_treat)
    @dog.should_receive(:shots).and_return("success")

    @dog.visit_vet

    @dog.last_visit.should == Date.today
  end
end
end

```



<http://www.flickr.com/photos/metrolibraryarchive/3215452689/>

KISS

~~KISS~~

Excuses

- Knowledge
- Time
- Ego

BehaviorDD

- “What We Got” vs. “How we got”
- Functional Tests vs. Unit Tests
- Human Friendly
- Durable Tests

Scenario: A visit to the vet

Given an existing owner with the following animals:

name	species	last_visit
Fido	Dog	2008-06-01
Izzie	Lizard	2009-03-01

And the owner's preferred doctor is "Mike"

And "Mike" works for "Acme Veterinary Co."

When the owner visits the vet

Then the dog should get shots

And the lizard should not get shots

*“I have achieved success without
working hard or making sacrifices” - ???*

<http://claytonlz.com/survey.html>

claytonlz.com

twitter.com/claytonlz

integrumtech.com