Day three

June, Wednesday 24th

Day three, Wednesday 24th

- Intro to unit testing with RSpec
- The marvelous world of TDD

Why do I need testing?


POLICE

Overview

- Yeah, it's code that checks that your code works fine

- Think of it as code police

- Automated testing is the key to balance good testing with implementation speed

More overview

- It's the way to know that what we wrote is correct, basically
- Guarantees robustness and a specific behavior
- It allows you to sleep at night

Purposes

- Assure that what we are implementing now works
- Act as a guard for the future, whistle blowing if something breaks it

Testing types

- There are many types of testing
  - unit
  - integration
  - acceptance
  - … and more

- We'll mainly treat unit testing

How does it work?

- Prepare the scenario
- Do whatever it takes to prove that feature
- Check that everything behaved as expected

An example

Testing that an instance method only returns the positive numbers attached to that instance.

- Prepare build an instance with both positive, zeros and negative numbers
- Do call that instance method
- Check that the result of the method call only includes the positive numbers

LIVE CODING FROM BCN!!!!111

First testing example

LET'S GET READY TO RUMBLE

Exercise SL8

IRON HACK

Our friend RSpec

Motivation

Abstract the testing software and just write the tests we want done.

Let's add another tool to our toolbelt!

Install!

```
gem install rspec
```

Color is fun

```
echo “--color -f documentation” > .rspec
```

OMG our first RSpec

```ruby
describe "God" do
  it "should work with booleans" do
    expect(true).to be_truthy
  end

  it "should count properly" do
    expect(1+3).to eq(4)
  end

  it "should find stuff" do
    expect(%w{ September October }).to include('September')
  end
end
```

IRON HACK

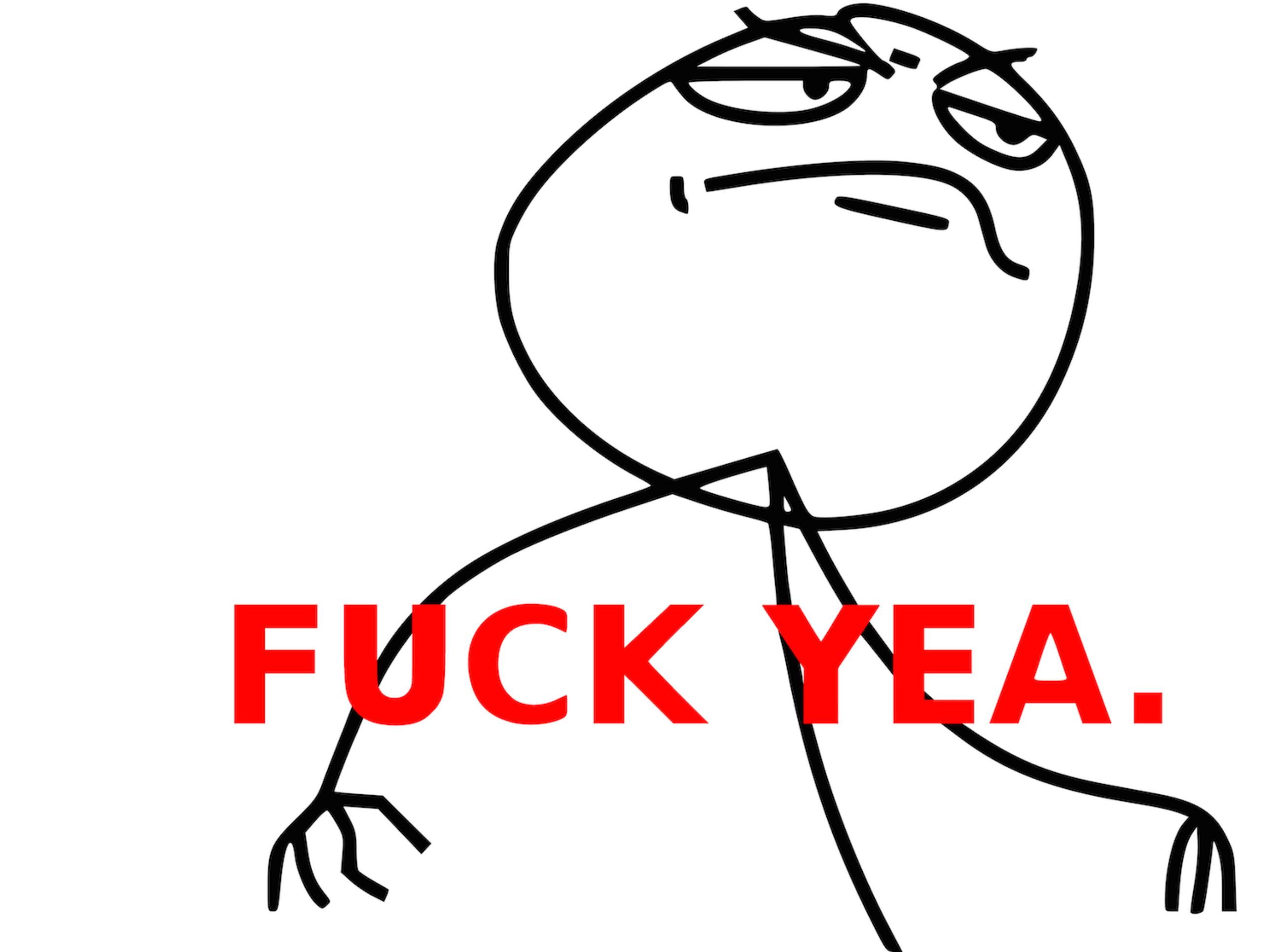
YEAH IT WORKS

```
 rspec god.rb

God
  should work with booleans
  should count properly
  should find stuff

Finished in 0.00351 seconds
3 examples, 0 failures
```

IRON HACK


FUCK YEA.

Let’s breathe

LIVE CODING FROM BCN!!!!111

Using RSpec

IRON HACK

LET'S GET READY TO RUMBLE

Exercise SL9

IRON HACK

Using RSpec right

- Use RSpec syntax accordingly
  - describe: for same semantic group e.g. a method, an attribute, a specific behaviour
  - context: for a situation where there is some specific data or information
  - before: for abstracting a common change of context between one or more specs

Writing good specs

- Explore all the different scenarios: leave nothing out of being tested!

- You're still writing code! Clear, simple…

- Writing too much specs is better than writing too few

An example, in RSpec

```ruby
class Numerifier
  attr_accessor :numbers

  def positive_numbers
    numbers.select { |number| number > 0 }
  end
end

describe Numerifier do
  before do
    @numerifier = Numerifier.new
  end

  describe "#positive_numbers" do
    it "should work with an empty array" do
      @numerifier.numbers = []
      expect(@numerifier.positive_numbers).to eq([])
    end

    it "should work with some numbers" do
      @numerifier.numbers = [1, 4, -5, 0, 3]
      expect(@numerifier.positive_numbers).to eq([1,4,3])
    end
  end
end
```

IRON HACK

TDD and BDD

Overview

- We used test-last mostly before (you remember, today morning?)

- But testing first helps us by thinking what we want, instead of how is it done

- Ask what you want to happen, then implement it!

TDD and BDD

- Two main approaches: TDD and BDD

- TDD is more focused on the tests

- BDD is more focused on the domain and OO design instead

On using it…

- Either TDD or BDD are possibilities, nothing is compulsory
- Extremes are bad
- Fundamentalist TDD/BDD != productive

IRON HACK

On using it…

- Take the best of both worlds
- Go for what is more needed at every moment
- Adapt it to your own taste!

LIVE CODING FROM BCN!!!!1!1

TDD and BDD

IRON HACK

LET'S GET READY TO RUMBLE

Exercises SL10 & SL11

IRON HACK