Overview
Rspec & Capybara

Objective

- Practical Behaviour-Driven Development
- Test-Driven Development with RSpec
- To become comfortable with the common tools used by Rubyists
- To learn and embrace the practices of successful Ruby/Rails developers

Requirements & Objectives

What you'll need on your system to play along...

Requirements

- Git (http://git-scm.com/)
- RVM (https://rvm.io) [Optional]
- Ruby 1.9.3 (or 2.0)
- RubyGems
- Any code/text editor

Material Conventions
Part 1 - Ruby

- This 60 to 80 minute training consists of a mix of lecture time, guided exercises and some labs (in class if we have the time, take home if we don't)

- For the guided exercises you will see a green "follow along" sign on the slides

- For the labs you'll see an orange sign with the lab number

Testing

Unit, Integration, Acceptance, BDD & TDD

Testing
A Means To An End

“Our highest priority is to satisfy the customer through early and continuous delivery of valuable software”

http://agilemanifesto.org/principles.html

Testing
A Means To An End

- Testing traditionally done at the end of development “if time permitted” (hello waterfall)
- No language support or frameworks (back in the old days)
- Started from the “inside” with Unit Testing Frameworks
- Lots of well tested units, we were still left with a mess at the outer layers
- BDD came in to try to reverse the testing approach

http://c2.com/cgi/wiki?TenYearsOfTestDrivenDevelopment

Testing
A Means To An End

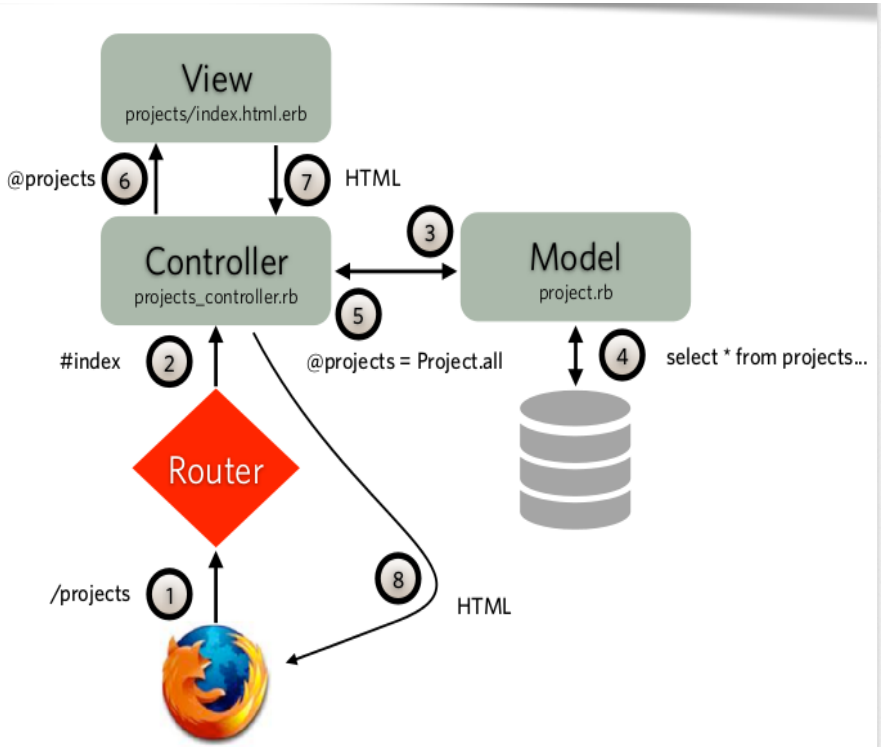
- BDD testing frameworks are DSLs (built on top of Unit Testing Frameworks) to “get the words rights”
- Most examples still use Units (class & methods) to teach BDD. Therefore developers still start at the inside.
- Rails showed early own that Web Application Testing CAN be automated
- Integration testing still hard to define for most developers
- Acceptance testing is NOT integration testing (unless you mean integrating with your users)

http://c2.com/cgi/wiki?TenYearsOfTestDrivenDevelopment

Request Handling

The Request-Response Pipeline

- User requests /projects
- Rails router forwards the request to projects_controller#index action
- The index action creates the instance variable @projects by using the Project model all method
- The all method is mapped by ActiveRecord to a select statement for your DB
- @projects returns back with a collection of all Project objects
- The index action renders the index.html.erb view
- An HTML table of Projects is rendered using ERB (embedded Ruby) which has access to the @projects variable
- The HTML response is returned to the User

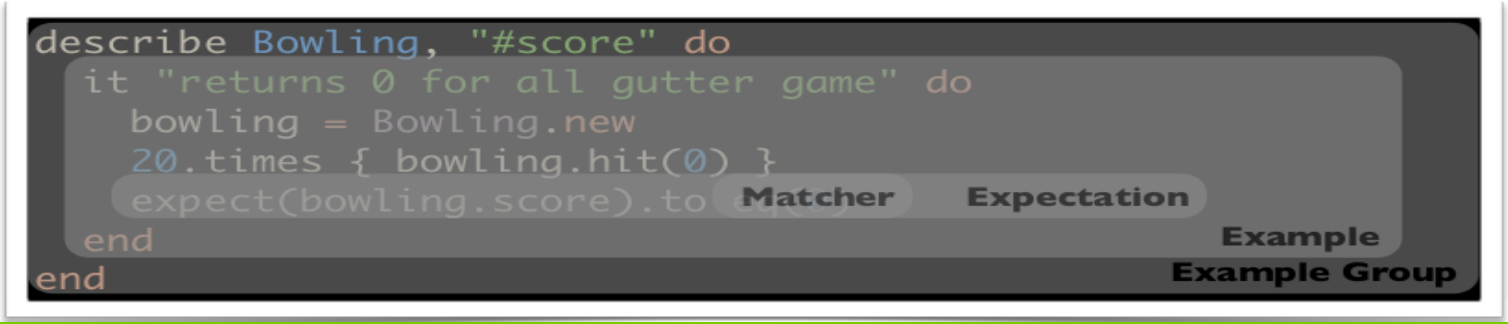


BDD

Behavior-Driven Development

RSpec
Ruby's BDD Framework

- RSpec is the most popular BDD framework for Ruby
- Created by Steven Baker in 2005, enhanced and maintained by David Chelimsky until late 2012
- RSpec provides a DSL to write executable examples of the expected behavior of a piece of code in a controlled context
- RSpec uses the method describe to create and Example Group
- Example groups can be nested using the describe or context methods



RSpec
Matchers

- RSpec comes built in with a nice collection of matchers, including:

```
be_true  # passes if actual is truthy (not nil or false)
be_false # passes if actual is falsy (nil or false)
be_nil   # passes if actual is nil
be       # passes if actual is truthy (not nil or false)

expect { ... }.to raise_error
expect { ... }.to raise_error(ErrorClass)
expect { ... }.to raise_error("message")
expect { ... }.to raise_error(ErrorClass, "message")

expect { ... }.to throw_symbol
expect { ... }.to throw_symbol(:symbol)
expect { ... }.to throw_symbol(:symbol, 'value')

be_xxx          # passes if actual.xxx?
have_xxx(:arg) # passes if actual.has_xxx?(:arg)
```

RSpec
Matchers

- and ...

```
be_empty

be(expected) # passes if actual.equal?(expected)
eq(expected) # passes if actual == expected

== expected     # passes if actual == expected
eql(expected)   # passes if actual.eql?(expected)
equal(expected) # passes if actual.equal?(expected)

be >  expected
be >= expected
be <= expected
be <  expected
=~ /expression/
match(/expression/)
be_within(delta).of(expected)

be_instance_of(expected)
be_kind_of(expected)
```

https://www.relishapp.com/rspec/rspec-expectations/v/2-13/docs/built-in-matchers

Test-Driven Development

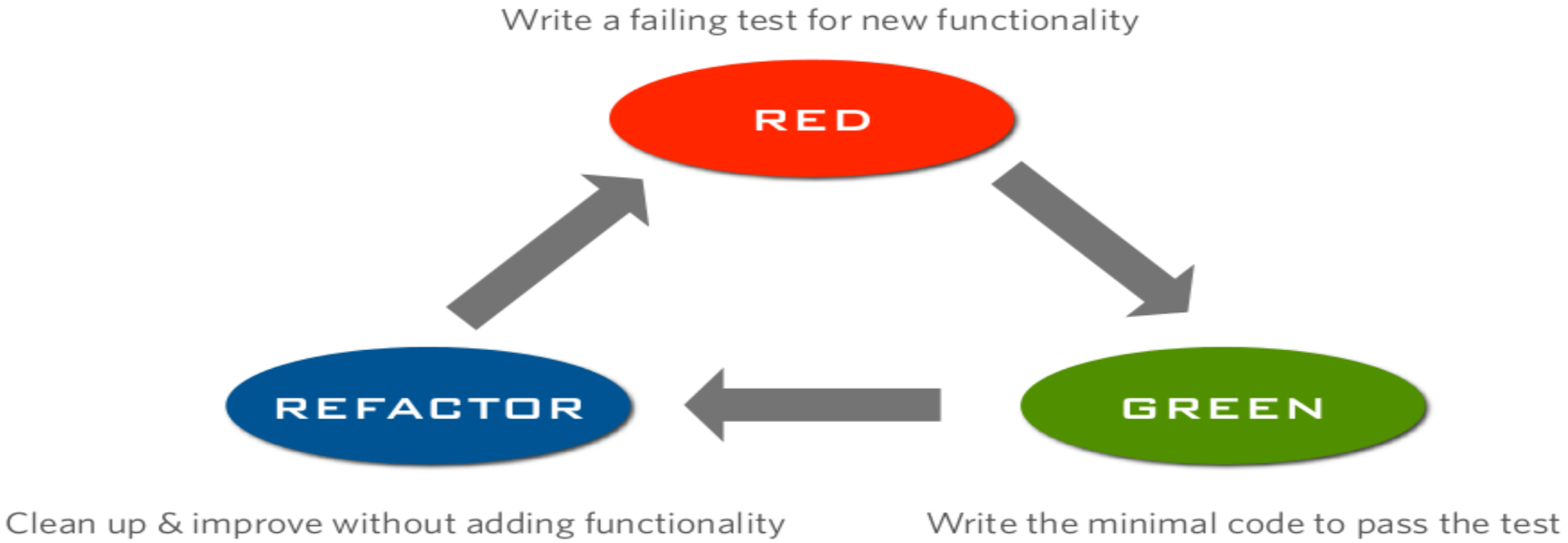
Drive your Development with Tests

- TDD is not *really* about testing
- TDD is a design technique
- TDD leads to cleaner code with separation of concerns
- Cleaner code is more reliable and easier to maintain
- TDD creates a tight loop of development that cognitively engages us
- TDD gives us lightweight rigor by making development, goal-oriented with a clear goal setting, goal reaching and improvement stages
- The stages of TDD are commonly known as the Red-Green-Refactor loop

Test-Driven Development

Drive your Development with Tests

- The Red-Green-Refactor Loop:



Test-Driven Development

Evolve your Code with Tests

- Let's work through a simple TDD/BDD exercise using RSpec
- We'll design a simple shopping cart class
- We'll start by creating a new folder for our exercise and adding a .rvmrc file and a Gemfile

```
/>mkdir rspec-follow-along
/>cd rspec-follow-along
/>echo 'rvm use 1.9.3@rspec-follow-along' > .rvmrc
/>touch Gemfile
/>mkdir spec
/>mkdir lib
```

```
source 'https://rubygems.org'

group :test do
  gem 'rspec'
end
```

Test-Driven Development
Evolve your Code with Tests

- With our project configure for RVM and a Gemfile in place we can re-enter the directory to

```
/>cd ..
/>cd rspec-follow-along/
Using /Users/user/.rvm/gems/ruby-1.9.3-p374 with gemset rspec-follow-along
bsb in ~/Courses/code/rspec-follow-along using ruby-1.9.3-p374@rspec-follow-along

/> bundle
Using diff-lcs (1.1.3)
Using rspec-core (2.12.2)
Using rspec-expectations (2.12.1)
Using rspec-mocks (2.12.2)
Using rspec (2.12.0)
Using bundler (1.2.3)
Your bundle is complete! Use `bundle show [gemname]` to see where a bundled gem is installed.
```

Test-Driven Development

Evolve your Code with Tests

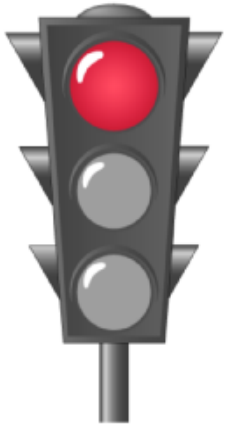
- We'll start the RGR loop with the simplest possible failure: "There is no Cart!"
- Create the file cart_spec.rb in the spec directory with the following contents:

```
describe Cart do


end
```

Test-Driven Development

Evolve your Code with Tests

- Let's run the specs using the rspec command and passing the spec directory as an argument
- Have we arrived at the RED state in our red-green-refactor cycle?



```
/>rspec spec
/Users/bsb/Courses/code/rspec-follow-along/spec/cart_spec.rb:1:in `<top (required)>':
uninitialized constant Cart (NameError)
```

Hint: if you have a failure with no tests it typically means that you need a test (but let's ignore that for a second...)

Test-Driven Development
Evolve your Code with Tests

- Let's create the Cart class in a /lib folder and require it in our spec:

```
class Cart
end
```

```
require_relative '../lib/cart.rb'

describe Cart do
end
```

- Now we have no failures but also we have no specs...

```
/> rspec spec
No examples found.


Finished in 0.00006 seconds
0 examples, 0 failures
```

Test-Driven Development

Evolve your Code with Tests

- Let's craft our first real test to drive the development of the Cart
- The spec to tackle is: "An instance of Cart when new contains no items"

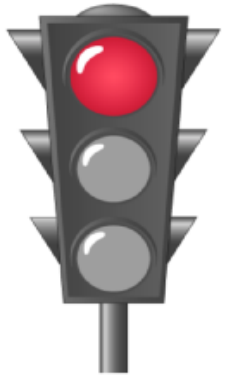
```
require_relative '../lib/cart.rb'

describe Cart do
  context "a new cart" do
    it "contains no items" do
      expect(@cart).to be_empty
    end
  end
end
```

Test-Driven Development

Evolve your Code with Tests

Follow Along

- If we run the specs we can see a failure:



```
/> rspec spec
F

Failures:
  1) Cart a new cart contains no items
     Failure/Error: expect(@cart).to be_empty
     NoMethodError:
       undefined method `empty?' for nil:NilClass
     # ./spec/cart_spec.rb:6:in `block (3 levels) in <top (required)>

Finished in 0.00243 seconds
1 example, 1 failure
```

Now we have our first "real" test-driven failure
(and that is a good thing!)

Test-Driven Development

Evolve your Code with Tests

- One of the mantras of BDD is to “get the words right”
- If you noticed on the last run the spec output read as “Cart a new cart contains no items”
- RSpec is flexible enough to allow us to pass a string to be prefixed to the describe block to make tailor the output to our needs

```
require_relative '../lib/cart.rb'

describe "An instance of", Cart do
  context "when new" do
    it "contains no items" do
      expect(@cart).to be_empty
    end
  end
end
```

Test-Driven Development

Evolve your Code with Tests

- If we run the specs we can see that the output now matches the desire spec wording

```
/> rspec spec
F

Failures:

  1) An instance of Cart when new contains no items
     Failure/Error: expect(@cart).to be_empty
     NoMethodError:
       undefined method `empty?' for nil:NilClass
     # ./spec/cart_spec.rb:6:in `block (3 levels) in <top (required)>'

Finished in 0.00154 seconds
1 example, 1 failure

Failed examples:

rspec ./spec/cart_spec.rb:5 # An instance of Cart when new contains no items
```

Test-Driven Development

Evolve your Code with Tests

- The output shows that we have two failures, one implicit and one explicit
- Explicit Failure: We are assuming that a cart has an #empty? method
- Implicit Failure: The instance variable @cart has not been initialized

```
Failures:

  1) An instance of Cart when new contains no items
     Failure/Error: expect(@cart).to be_empty
     NoMethodError:
       undefined method `empty?' for nil:NilClass
     # ./spec/cart_spec.rb:6:in `block (3 levels) in <top (required)>'
```

Test-Driven Development
Evolve your Code with Tests

- We'll start by addressing the fact that our test fixture hasn't been setup
- Just adding the line @cart = Cart.new wouldn't be very TDDish
- What we should do is first make the failure explicit by writing a test for it!

```
require_relative '../lib/cart.rb'

describe "An instance of", Cart do

  it "should be properly initialized" do
    expect(@cart).to be_a(Cart)
  end

...
```

Remember our initial failure with no tests?

Test-Driven Development

Evolve your Code with Tests

- Now we have two valid failing tests to pass, let's get on with it!

```
...

  1) An instance of Cart should be properly initialized
     Failure/Error: expect(@cart).to be_a(Cart)
       expected nil to be a kind of Cart
     # ./spec/cart_spec.rb:6:in `block (2 levels) in <top (required)>'

  2) An instance of Cart when new contains no items
     Failure/Error: expect(@cart).to be_empty
     NoMethodError:
       undefined method `empty?' for nil:NilClass
     # ./spec/cart_spec.rb:11:in `block (3 levels) in <top (required)>'

Finished in 0.00233 seconds
2 examples, 2 failures
```

Test-Driven Development

Evolve your Code with Tests

- We'll pass the test by adding the line @cart = Cart.new in a before-each block:

```
describe "An instance of", Cart do

  before :each do
    @cart = Cart.new
  end
```

```
...
  1) An instance of Cart when new contains no items
     Failure/Error: expect(@cart).to be_empty
     NoMethodError:
       undefined method `empty?' for #<Cart:0x007fc98316cb60>

...
```

Test-Driven Development

Evolve your Code with Tests

- Let's add a skeleton empty? method to the Cart class:

```
class Cart
  def empty?
    nil
  end
end
```

```
...
  1) An instance of Cart when new contains no items
     Failure/Error: expect(@cart).to be_empty
       expected empty? to return true, got nil
     # ./spec/cart_spec.rb:15:in `block (3 levels) in <top (required)>'

...
```

Test-Driven Development

Evolve your Code with Tests

- Now we can comply with the spec by providing an implementation of our Cart
- In this case we are using a Hash to hold our items and delegating to the @items#empty? method

```ruby
class Cart

  def initialize
    @items = {}
  end

  def empty?
    @items.empty?
  end
end
```

```
/> rspec spec
..

Finished in 0.00196 seconds
2 examples, 0 failures
```

We've reached the GREEN state

Test-Driven Development

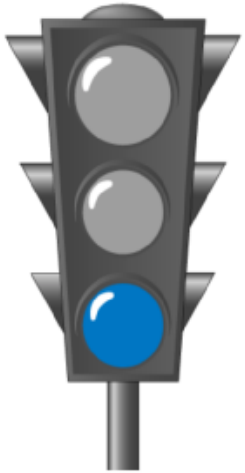
Drive your Development with Tests

- In the REFACTOR state we concentrate on making the current implementation better, cleaner and more robust
- It is very likely that early on in the development there won't be much to refactor
- The need for refactoring is a side-effect of increasing complexity and interaction between classes and subsystems
- Refactoring can also introduce implementation specific specs or reveal holes in your previous specs

Test-Driven Development

Evolve your Code with Tests

- Let's use Ruby's Forwardable module to simplify the delegation of the collection methods to the @items Hash:



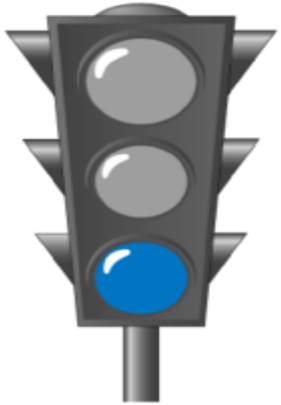
```ruby
class Cart
  extend Forwardable
  def_delegator :@items, :empty?
  def initialize
    @items = {}
  end
end
```

```
/>rspec spec
..

Finished in 0.00365 seconds
2 examples, 0 failures
```

Test-Driven Development

Evolve your Code with Tests

- The RSpec command provides several arguments to tailor the run and output of your specifications
- For example to see the group and example names in the output use --format documentation



```
/> rspec --format documentation

An instance of Cart
  should be properly initialized
  when new
    contains no items

Finished in 0.00224 seconds
2 examples, 0 failures
```

Test-Driven Development

Evolve your Code with Tests

- Lab 1.0 consists of 4 specs to be implemented in a TDD fashion:
  - “An new and empty cart total value should be $0.0”
  - “An empty cart should no longer be empty after adding an item”
  - “A cart with items should have a total value equal to the sum of each items’ value times its quantity”
  - “Increasing the quantity of an item should not increase the Carts’ unique items count”

Rspec Part - II

Test-Driven Development

Evolve your Code with Tests

- Let's Create a Rails Project

  `rails new tasks -T -d mysql`

- Edit your Gemfile, and add following lines and run bundle install

```
group :development, :test do
  gem 'rspec-rails', '~> 3.0.0'
  gem 'factory_girl_rails'
  gem 'capybara'
  gem 'database_cleaner'
end
```

Test-Driven Development
Evolve your Code with Tests

Run generators

Next, run the RSpec generator to create the initial skeleton:

```
rails generate rspec:install
```

Configure Capybara

To make Capybara available from within RSpec specs, add the following line to `spec/rails_helper.rb`:

```
# spec/rails_helper.rb
require 'capybara/rails'
```

Your Capybara feature specs will also need a home. Add the `spec/features/` directory:

Test-Driven Development
Evolve your Code with Tests

Configure `database_cleaner`

Next, make the following adjustments to `spec/rails_helper.rb` to integrate the `database_cleaner` gem:

```
config.use_transactional_fixtures = false
```

Test-Driven Development
Evolve your Code with Tests

Configure `database_cleaner`

Next, make the following adjustments to `spec/spec_helper.rb` to integrate the `database_cleaner` gem:

```
RSpec.configure do |config|

  config.before(:suite) do
    DatabaseCleaner.clean_with(:truncation)
  end

  config.before(:each) do
    DatabaseCleaner.strategy = :transaction
  end

  config.before(:each, :js => true) do
    DatabaseCleaner.strategy = :truncation
  end

  config.before(:each) do
    DatabaseCleaner.start
  end

  config.after(:each) do
    DatabaseCleaner.clean
  end

end
```

Test-Driven Development
Evolve your Code with Tests

Include the `factory_girl` methods

To make the `factory_girl` gem's methods (e.g., `build` and `create`) easily available in RSpec examples, add this line to the top of your `RSpec.configure` block in `spec/rails_helper.rb`:

```
RSpec.configure do |config|
  config.include FactoryGirl::Syntax::Methods

  # other configurations below...
end
```

Test-Driven Development
Evolve your Code with Tests

Generating specs

The next time you run `rails generate resource`, Rails should already be configured to generate specs under `spec/` and factories under `spec/factories`.

Let's try that out with the proverbial Rails blog example:

```
rails generate resource post title:string content:text published:boolean
```

Test-Driven Development

Evolve your Code with Tests

If everything's correctly configured, you should see something close to this:

```
invoke  active_record
create    db/migrate/20140630160246_create_posts.rb
create    app/models/post.rb
invoke    rspec
create      spec/models/post_spec.rb
invoke      factory_girl
create        spec/factories/posts.rb
invoke  controller
create    app/controllers/posts_controller.rb
invoke    erb
create      app/views/posts
invoke    rspec
create      spec/controllers/posts_controller_spec.rb
invoke    helper
create      app/helpers/posts_helper.rb
invoke      rspec
create        spec/helpers/posts_helper_spec.rb
invoke    assets
invoke      coffee
create        app/assets/javascripts/posts.js.coffee
invoke      scss
create        app/assets/stylesheets/posts.css.scss
invoke  resource_route
 route    resources :posts
```

Test-Driven Development

Evolve your Code with Tests

Note the factory at `spec/factories/posts.rb` , the model spec at `specs/models/post_spec.rb`, and the controller spec at `specs/controllers/post_controller_spec.rb` .

With the new specs in place, try running `rspec` (you can also run `rake`) to see the results. So far you should only see a few pending specs:

```
**

Pending:
  PostsHelper add some examples to (or delete) ./spec/helpers/posts_helper_spec.rb
    # Not yet implemented
    # ./spec/helpers/posts_helper_spec.rb:14
  Post add some examples to (or delete) ./spec/models/post_spec.rb
    # Not yet implemented
    # ./spec/models/post_spec.rb:4
```

Test-Driven Development
Evolve your Code with Tests

To update the test db

```
rake db:test:prepare
```

Let's start from the model, open /spec/models/post.rb and edit it like this

```
require 'spec_helper'
describe Post do
  it "is valid with title and boby"
  it "is not valid with an empty title"
end
```

Test-Driven Development

Evolve your Code with Tests

If you run "bundle exec rspec spec/models/post_spec.rb" you will see a message like this:

```
Pending:
  Post is valid with title and boby
    # Not yet implemented
    # ./spec/models/post_spec.rb:4
  Post is not valid with an empty title
    # Not yet implemented
    # ./spec/models/post_spec.rb:5
```

Test-Driven Development

Evolve your Code with Tests

That happens because we don't have defined the test. Open the file and add the following lines, note that we are using FactoryGirl to define the Post ActiveRecord object.

```
require 'spec_helper'
describe Post do
  before(:each) do
    @post = FactoryGirl.build(:post)
  end
  it 'is valid with title and content' do
    @post.should be_valid
  end
  it 'is not valid with an empty title' do
    @post.title = nil
    @post.should_not be_valid
  end
end
```