MOVING INTO UNIT TESTING

using Rspec

RSpec, a Domain-Specific Language for testing

- DSL: small programming language that simplifies one task at expense of generality
  - examples so far: migrations, regexes, SQL
- RSpec tests are called *specs*, and inhabit spec directory

app/models/*.rb	spec/models/*_spec.rb
app/controllers/ *_controller.rb	spec/controllers/ *_controller_spec.rb
app/views/*/*.html.haml	(use Cucumber!)

Example: calling TMDb

➤New RottenPotatoes feature: add movie using info from TMDb (vs. typing in)

➤How should user story steps behave?

When I fill in "Search Terms" with "Inception"

And I press "Search TMDb"

Then I should be on the RottenPotatoes homepage

...

Recall Rails Cookery #2:
adding new feature ==
new route+new controller method+(new view?)

The Code You Wish You Had

What should the *controller method* do that receives the search form?

1. it should call a method that will search TMDb for specified movie
2. if match found: it should select (new) “Search Results” view to display match
3. If no match found: it should redirect to RP home page with message

http://pastebin.com/kJxjwSF6

EXAMPLE (SEE PASTEBIN)

```
require 'spec_helper'

describe MoviesController do
  describe 'searching TMDb' do
    it 'should call the model method that performs TMDb
search'
    it 'should select the Search Results template for
rendering'
    it 'should make the TMDb search results available to that
template'
  end
end
```

The TDD Cycle: Red—Green—Refactor

Example: calling TMDb

➤New RottenPotatoes feature: add movie using info from TMDb (vs. typing in)

➤How should user story steps behave?

When I fill in "Search Terms" with "Inception"

And I press "Search TMDb"

Then I should be on the RottenPotatoes homepage

...

Recall Rails Cookery #2:
adding new feature ==
new route+new controller method+(new view?)

The Code You Wish You Had

What should the *controller method* do that receives the search form?

1. it should call a method that will search TMDb for specified movie
2. if match found: it should select (new) “Search Results” view to display match
3. If no match found: it should redirect to RP home page with message

http://pastebin.com/kJxjwSF6

EXAMPLE (SEE PASTEBIN)

```
require 'spec_helper'

describe MoviesController do
  describe 'searching TMDb' do
    it 'should call the model method that performs TMDb
search'
    it 'should select the Search Results template for
rendering'
    it 'should make the TMDb search results available to that
template'
  end
end
```

Test-First development

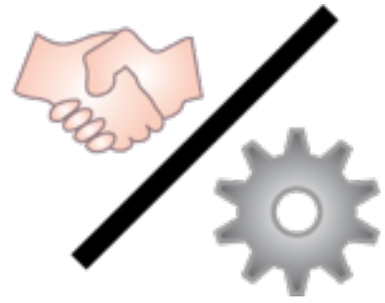
- Think about one thing the code *should* do
- Capture that thought in a test, which fails
- Write the simplest possible code that lets the test pass
- Refactor: DRY out commonality w/other tests
- Continue with next thing code should do

Red – Green – Refactor

Aim for "always have working code"

TDD for the Controller action: Setup

➤Add a route to **`config/routes.rb`**



```
# Route that posts 'Search TMDb' form

post '/movies/search_tmdb'
```

➤Convention over configuration will map this to MoviesController#search_tmdb

➤Create an empty view:

```
touch app/views/movies/search_tmdb.html.haml
```

➤Replace fake “hardwired” method in **`movies_controller.rb`** with empty method:

```
def search_tmdb

end
```

What model method?

➤Calling TMDb is responsibility of the model... but no model method exists to do this yet!

➤No problem...we'll use a seam to test the *code we wish we had ("CWWWH")*, Movie.find_in_tmdb

➤Game plan:

- ➤Simulate POSTing search form to controller action.
- ➤Check that controller action *tries to call* Movie.find_in_tmdb with data from submitted form.
- ➤The test will fail (red), because the (empty) controller method *doesn't* call find_in_tmdb.
- ➤Fix controller action to make green.

http://pastebin.com/zKnwphQZ

MOVIES_CONTROLLER_SPEC.RB

```ruby
require 'spec_helper'

describe MoviesController do
  describe 'searching TMDb' do
    it 'should call the model method that performs TMDb search' do
      Movie.should_receive(:find_in_tmdb).with('hardware')
      post :search_tmdb, {:search_terms => 'hardware'}
    end
  end
end
```

NOTE: should_receive is OLD! http://www.teaisaweso.me/blog/2013/05/27/rspecs-new-message-expectation-syntax/

MOVIES_CONTROLLER_SPEC.RB

```
require 'spec_helper'

describe MoviesController do
  describe 'searching TMDb' do
    it 'should call the model method that performs TMDb search' do
      expect(Movie).to receive(:find_in_tmdb).with('hardware')
      //Old Movie.should_receive(:find_in_tmdb).with(‘hardware’)
      post :search_tmdb, {:search_terms => 'hardware'}
    end
  end
end
```

NOTE: should_receive is OLD! http://www.teaisaweso.me/blog/2013/05/27/rspecs-new-message-expectation-syntax/

Seams

➤A place where you can change your app's *behavior* without editing the *code*. (Michael Feathers, *Working Effectively With Legacy Code*)

➤Useful for testing: ***isolate*** behavior of some code from that of other code it depends on.

➤should_receive (and expect) uses Ruby's open classes to create a seam for isolating controller action from behavior of (possibly buggy or missing) Movie.find_in_tmdb

➤Rspec *resets* all mocks & stubs after *each example* (keep tests Independent)

How to make this spec green?

- Expectation says controller action should call Movie.find_in_tmdb
- So, let's call it!

http://pastebin.com/DxzFURiu

```
def search_tmdb
  Movie.find_in_tmdb(params[:search_terms])
end
```

The spec has *driven* the creation of the controller method to pass the test.

- But shouldn't find_in_tmdb *return* something?

Test techniques we know

obj.should_receive(a).with(b)

expect(obj).to receive(a).with(b)

More Controller Specs and Refactoring

Where we are & where we're going: "outside in" development

- Focus: write *expectations* that drive development of controller method
  - Discovered: must *collaborate* w/model method
  - Use outside-in recursively: *stub* model method in this test, write it later- NOTE: NOW CALLED A DOUBLE
    - Key idea: *break dependency* between method under test & its collaborators
- Key concept: *seam*—where you can affect app behavior without editing code

The Code You Wish You Had

What should the *controller method* do that receives the search form?

1. it should call a method that will search TMDb for specified movie
2. if match found: it should select (new) “Search Results” view to display match
3. If no match found: it should redirect to RP home page with message

"it should select Search Results view to display match"

- Really 2 specs:
  - It should decide to render Search Results
    - more important when different views could be rendered depending on outcome
  - It should make list of matches available to that view
- New *expectation* construct:
  - obj.should *match-condition*
  - expect(obj).to *match-condition*
  - Many built-in matchers, or define your own

Should & Should-not

- Matcher applies test to receiver of *should*

count.should == 5	Syntactic sugar for count.should.==(5)
5.should(be.<(7))	be creates a lambda that tests the predicate expression
5.should be < 7	Syntactic sugar allowed
5.should be_odd	Use method_missing to call odd? on 5
result.should include(elt)	calls Enumerable#include?
result.should match(/regex/)	
should_not also available	

```
result.should render_template('search_tmdb')
```

Good examples of NEW SYNTAX: http://rspec.info/blog/2012/06/rspecs-new-expectation-syntax/ — USE EXPECT!

Checking for rendering

- After post :search_tmdb, response() method returns controller's *response object*
  - render_template matcher can check what view the controller tried to render

- Note that this view has to exist!

http://pastebin.com/C2x13z8M

  - post :search_tmdb will try to do the whole MVC flow, including rendering the view
  - hence, controller specs can be viewed as *functional testing*

AN EXAMPLE

More "new" stuff: https://semaphoreci.com/community/tutorials/mocking-with-rspec-doubles-and-expectations

```
1.  require 'spec_helper'
2.
3.  describe MoviesController do
4.    describe 'searching TMDb' do
5.      it 'should call the model method that performs TMDb search' do
6.        Movie.should_receive(:find_in_tmdb).with('hardware')
7.        post :search_tmdb, {:search_terms => 'hardware'}
8.      end
9.      it 'should select the Search Results template for rendering' do
10.       Movie.stub(:find_in_tmdb)
11.       post :search_tmdb, {:search_terms => 'hardware'}
12.       response.should render_template('search_tmdb')
13.     end
14.   end
15. end
```

Test techniques we know

obj.should_receive(a).with(b)

expect(obj).to receive(a).with(b)

obj.should *match-condition*

expect(obj).to match-condition

Rails-specific extensions to RSpec:

response()

render_template()