Further Testing Techniques

Test Driven Development (TDD)

- new code (both in app and possible in step defs)
- TDD: write tests before the new code itself
- AKA: write tests for code you wish you had
- Improves modularity
- Steps:
  - 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

Testing Today

- Before: debugging focus
  - Developers finish code, some ad-hoc tests
  - "toss over the wall to QA"
  - QA staff manually poke at software
- Today: 'maintainability and validation focus'
  - Testing is part of every Agile iteration
  - Developers test their own code
  - Testing tools and processes highly automated
  - QA/testing group improves testability and tools
  - Still some manual testing to... but for different reasons

Unit Tests should be FIRST

- Fast: run (subset of) tests quickly (since you'll be running them all the time)
- Independent: no test depends on others; can run any subset in any order
- Repeatable: run *N* times, get same result (to help isolate bugs and enable automation)
- Self-checking: test can automatically detect if passed (no human checking of output)

Test Cases: Arrange, Act, Assert

- Arrange preconditions
  - Q: What about non-leaf methods
  - What about methods that depend on external state or even an external service
  - What about database data used by the test
  - What about 'world state' (eg: logged in?)
- Act on the system under tests (SUT)
  - Q: what about testing controller actions?
- Assert postcondition(s)
  - Model tests are 'easy'
  - Have to isolate MVC even though a real request usually touches all three ## Expectations / Assertions
- `expect(x).to eq('<value>')`
  - `eq` could be any of RSpec's matchers
  - Can use `not_to` for negation
  - Can append `_<method:bool>` to check
- Can also expect expression eg: `expect { <expression> }.to raise_error`
  - `expect { @review.destroy }.to change { Review.count }.by -1`
- Can set up preconditions with `before(:each)` to run before all code blocks within the `describe` block

- Specs should test just one behavior

Isolating Code: Doubles & Seams Intro

- `rspec-rails` gem can simulate `get`, `post`, and `put` requests for testing controllers
  - Has `response` object which says what controller is about to do when action finishes
  - Has matchers to test rails behaviors (`render_template`, `expect(assigns[:results]).to be_a_kind_of Enumerable`)
  - Supports creating doubles
- Can use method stubs

```
require 'rails_helper'

describe MoviesController do
    describe 'searching TMDb' do
        it 'calls the model method that preforms TMDb search' do
            # Set up a spy on the 'find_in_tmdb' method on Movie.
            expect(Movie).to receive(:find_in_tmdb).with('hardware')
            # Make request to trigger the expectation.
            get "movies/search_tmdb?search_terms=hardware"
        end
        it 'selects the Search Results template for rendering' do
            # Mock the find_in_tmdb method.
            allow(Movie).to receive(:find_in_tmdb)
            get "movies/search_tmdb?search_terms=hardware"
            # Expect the correct render template.
            expect(response).to render_template('search')
        end
    end
end

def search
    params = params.permit('search')
    Movie.find_in_tmdb(params['search'])
end
```

- `assigns[]` defines a set of instance variables set by a controller
  - eg usage: `expect(assigns[:movie]).to eq(<val>)`

Stunt Doubles

- `m = double('Movie')`
- Can stub methods on doubles
- `allow(m).to receive(:title).and_return('Snowden')`
  - Can also use `m = double('movie', :title => 'snowden')`
- Can stub responses on specific call numbers

```
it 'makes search results available to template' do
    fake_results = [double('Movie'), double('Movie')]
    allow(Movie).to
        receive(:find_in_tmdb).and_return(fake_results)
    post :search_tmdb, {:search_terms => 'hardware'}
    expect assigns[:movies].to eq(fake_results)
end
```