RSpec on Rails

(Engineering Software as a Service §8.2)

Armando Fox

RSpec, a Domain-Specific Language for testing

- RSpec tests (*specs)* inhabit **`spec`** directory

  `rails generate rspec:install` creates structure

- Unit tests (model, helpers)
- Functional tests (controllers)
- Integration tests (views)?

`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

The method that contacts TMDb to search for a movie should be:

- ☐ A class method of the Movie model
- ☐ An instance method of the Movie model
- ☐ A controller method
- ☐ A helper method

The TDD Cycle: Red–Green–Refactor

(Engineering Software as a Service §8.3)

Armando Fox

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"

How to test something “in isolation” if it has *dependencies* that would affect test?

The Code You Wish You Had

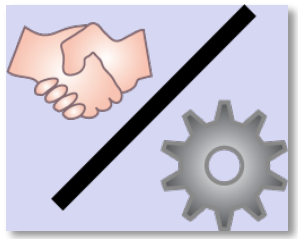
What should the *controller method* do that receives the search form?

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

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

Which is FALSE about `should_receive`?

- ☐ It provides a stand-in for a real method that doesn't exist yet
- ☐ It would override the real method, even if it did exist
- ☐ It can be issued either before or after the code that should make the call
- ☐ It exploits Ruby's open classes and metaprogramming to create a seam

Seams

(Engineering Software as a Service §8.3)

Armando Fox

Seams

- A place where you can change app's *behavior* without changing *source 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` 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

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)
```

Optional!

Eventually we will have to write a real `find_in_tmdb`. When that happens, we should:

- ☐ Replace the call to `should_receive` in our test with a call to the real `find_in_tmdb`
- ☐ Ensure the API to the real `find_in_tmdb` matches the fake one used by `should_receive`
- ☐ Keep the `should_receive` seam in the spec, but if necessary, change the spec to match the API of the real `find_in_tmdb`
- ☐ Remove this spec (test case) altogether since it isn't really testing anything anymore

Expectations

(Engineering Software as a Service §8.4)

Armando Fox

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
- 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:

1. It should decide to render Search Results
   - more important when different views could be rendered depending on outcome
2. It should make list of matches available to that view

- New *expectation* construct:

  `obj.should` *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 `#include?`, which usually gets handled by Enumerable
`republican.should cooperate_with(democrat)`	calls programmer's custom matcher `#cooperate_with` (and probably fails)

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

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

http://pastebin.com/C2x13z8M

- Note that this view has to exist!
  - `post :search_tmdb` will try to do the whole MVC flow, including rendering the view
  - hence, controller specs can be viewed as *functional testing*

Test techniques we know

```
obj.should_receive(a).with(b)
```

```
obj.should match-condition
```

Rails-specific extensions to RSpec:

```
response()
render_template()
```

Which of these, if any, is *not* a valid use of `should` or `should_not`?

- ☐ `result.should_not be_empty`
- ☐ `5.should be <=> result`
- ☐ `result.should_not match /^D'oh!$/`
- ☐ All of the above are valid uses

Mocks and Stubs

(Engineering Software as a Service §8.4)

Armando Fox

It should make search results available to template

- Another rspec-rails addition: `assigns()`
  - pass symbol that names controller instance variable
  - returns value that controller assigned to variable

- D'oh! our current code *doesn't set any instance variables:*
- TCWWWH: list of matches in `@movies`

http://pastebin.com/DxzFURiu

http://pastebin.com/4W08wL0X

Two new seam concepts

- `stub`
  - similar to `should_receive`, but not expectation
  - `and_return` optionally controls return value
- `mock`: "stunt double" object, often used for behavior verification (did method get called)
  - stub individual methods on it:

```
m=mock('movie1',:title=>'Rambo')
```

each seam enables just enough functionality for some *specific* behavior under test

RSpec Cookery #1

- Each spec should test *just one behavior*
- Use seams as needed to isolate that behavior
- Determine what type of expectation will check the behavior
- Write the test and make sure it fails *for the right reason*
- Add code until test is green
- Look for opportunities to refactor/beautify

Test techniques we know

```
obj.should_receive(a).with(b).and_return(c)
obj.stub(a).and_return(b)

d = mock('impostor')
```

Optional!

```
obj.should match-condition
```

Rails-specific extensions to RSpec:

```
assigns(:instance_var)
response()
render_template()
```

`should_receive` combines _____
and _____,
whereas `stub` is only _____.

- ☐ A mock and an expectation; a mock
- ☐ A mock and an expectation; an expectation
- ☐ A seam and an expectation; an expectation
- ☐ A seam and an expectation; a seam

Fixtures and Factories

(Engineering Software as a Service §8.5)

Armando Fox

When you need the real thing

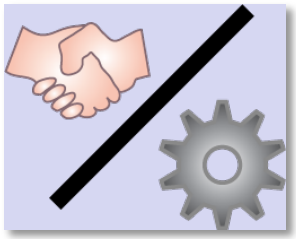
http://pastebin.com/N3s1A193

Where to get a real object:

- Fixture: statically preload some known data into database tables
- Factory: create only what you need per-test

Fixtures

- database wiped & reloaded before *each spec*
  - add `fixtures :movies` at beginning of `describe`
  - **`spec/fixtures/movies.yml`** are `Movies` and will be added to **`movies`** table



- Pros/uses
  - truly static data, e.g. configuration info that never changes
  - easy to see all test data in one place
- Cons/reasons not to use
  - may introduce dependency on fixture data

Factories

- Set up "helpers" to quickly create objects with default attributes, as needed per-test
- Example: FactoryGirl gem http://pastebin.com/bzvKG0VB
  - or just add your own code in **`spec/support/`**
- Pros/uses:
  - Keep tests Independent: unaffected by presence of objects they don't care about
- Cons/reasons not to use:
  - Complex relationships may be hard to set up (but may indicate too-tight coupling in code!)

Pitfall: *mock trainwreck*

- Goal: test searching for movie by its director or by awards it received

```
m.award.type.should == 'Oscar'
m.director.name.split(/ +/).last.
  should == 'Aronovsky'
```

- Mock setup:

```
a = mock('Award', :type => 'Oscar')
d = mock('Director',
    :name => 'Darren Aronovsky'
m = mock('Movie', :award => a,
    :director => d)
```

Which of the following kinds of data, if any, should *not* be set up as fixtures?

- ☐ Movies and their ratings
- ☐ The TMDb API key
- ☐ The application's time zone settings
- ☐ Fixtures would be fine for all of these

TDD for the Model & Stubbing the Internet

(Engineering Software as a Service §8.6–8.7)

Armando Fox

Explicit vs. implicit requirements

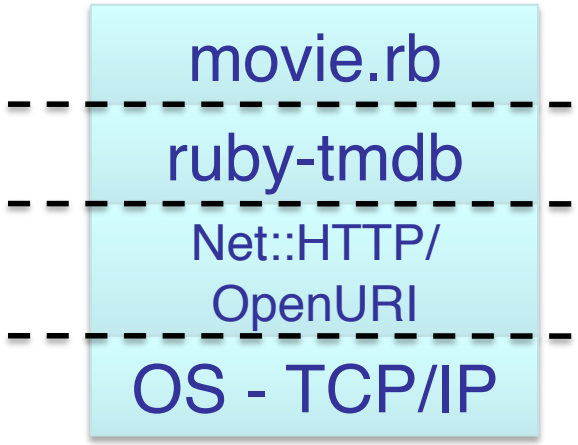
- `find_in_tmdb` should call TmdbRuby gem with title keywords
  - If we had no gem: It should directly submit a RESTful URI to remote TMDb site
- What if TmdbRuby gem signals error?
  - API key is invalid
  - API key is not provided
- Use *context & describe* to divide up tests

http://pastebin.com/ELQfC8Je

Review

- Implicit requirements derived from explicit
  - by reading docs/specs
  - as byproduct of designing classes
- We used 2 different stubbing approaches
  - case 1: we *know* TMDb gem will *immediately* throw error; test that we catch & convert it
  - case 2: need to *prevent* gem from contacting TMDb at all
- `context` & `describe` *group* similar tests
  - in book: using `before(:each)` to setup common preconditions that apply to whole group of tests

Where to stub in Service Oriented Architecture?




```
Movie.stub(:find_in_tmdb).
  and_return(...)
TmdbMovie.stub(:find).with(...).
  and_return(...)
Net::HTTP.stub(:get).
  with(...).
  and_return(...)
```

Rule of thumb:

- for unit testing, stub nearby, for maximum *isolation* of class under test (Fast, Independent)
- for integration testing, stub *far away*, to test as many interfaces as possible

Parse contrived request
Return canned value(s)
(using `FakeWeb` *gem, for example)*

Test techniques we know

```
obj.should_receive(a).with(b).and_return(c)
                    .with(hash_including 'k'=>'v')
obj.stub(a).and_raise(SomeClass::SomeError)

d = mock('impostor')

{ action }.should raise_error(Some::Error)
describe, context
```

Rails-specific extensions to RSpec:

```
assigns(:instance_var)
response()
render_template()
```

Which statement(s) are TRUE about Implicit requirements?

- ☐ They are often, but not always, derived from explicit requirements
- ☐ They apply only to unit & functional tests, not integration tests
- ☐ Testing them is lower priority than testing explicit requirements, since they don't come from the customer
- ☐ All of the above are true

Coverage, Unit vs. Integration Tests

(Engineering Software as a Service §8.8)

Armando Fox

How much testing is enough?

- Bad: “Until time to ship”
- A bit better: (Lines of test) / (Lines of code)
  - 1.2–1.5 not unreasonable
  - often *much higher* for production systems
- Better question: “How thorough is my testing?”
  - Formal methods
  - Coverage measurement
  - We focus on the latter, though the former is gaining steady traction

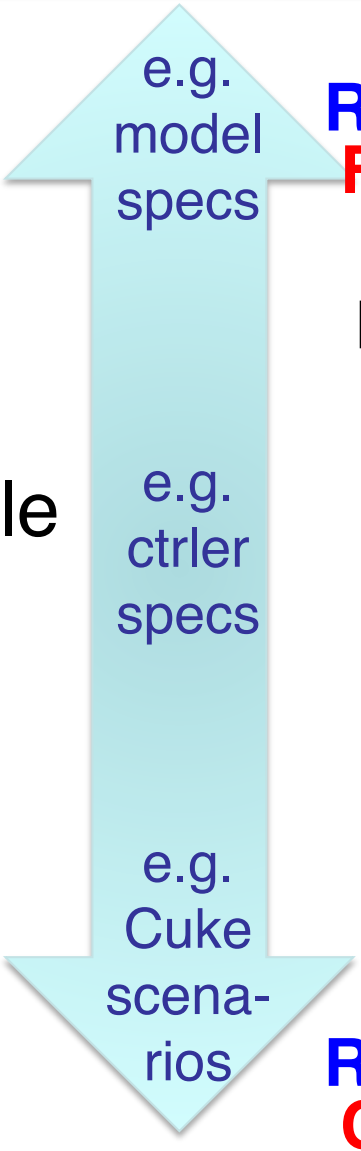
Measuring Coverage—Basics

```
class MyClass
 def foo(x,y,z)
  if x
    if (y && z) then bar(0) end
   else
    bar(1)
   end
 end
 def bar(x) ; @w = x ; end
end
```

- S0: every method called
- S1: every method *from every call site*
- C0: every statement
  - Ruby SimpleCov gem
- C1: every branch in both directions
- C1+decision coverage: every *subexpression* in conditional
- C2: every path (difficult, and disagreement on how valuable)

What kinds of tests?

- Unit (one method/ class)
- Functional or module (a few methods/ classes)
- Integration/system



Runs fast **High coverage**
Fine resolution
Many mocks;
Doesn't test interfaces

Few mocks;
tests interfaces
Runs slow **Low coverage**
Coarse resolution

Going to extremes

- ✗ "I kicked the tires, it works"
- ✗ "Don't ship until 100% covered & green"
- ☑ use coverage to identify untested or undertested parts of code
- ✗ "Focus on unit tests, they're more thorough"
- ✗ "Focus on integration tests, they're more realistic"
- ☑ each finds bugs the other misses

Which of these is POOR advice for TDD?

- ☐ Mock & stub early & often in unit tests
- ☐ Aim for high unit test coverage
- ☐ Sometimes it's OK to use stubs & mocks in integration tests
- ☐ Unit tests give you higher confidence of system correctness than integration tests

Other Testing Concepts; Testing vs. Debugging

(Engineering Software as a Service §8.9, 8.12)

Armando Fox

Other testing terms you may hear

- Mutation testing: if introduce deliberate error in code, does some test break?
- Fuzz testing: 10,000 monkeys throw random input at your code
  - Find ~20% MS bugs, crash ~25% Unix utilities
  - *Tests app the way it wasn't meant to be used*
- DU-coverage: is every pair <define x/use x> executed?
- Black-box vs. white-box/glass-box

TDD vs. Conventional debugging

Conventional	TDD
Write 10s of lines, run, hit bug: break out debugger	Write a few lines, with test first; know immediately if broken
Insert printf's to print variables while running repeatedly	Test short pieces of code using expectations
Stop in debugger, tweak/set variables to control code path	Use mocks and stubs to control code path
Dammit, I thought for sure I fixed it, now have to do this all again	Re-run test automatically

- Lesson 1: TDD uses same skills & techniques as conventional debugging—but more productive (FIRST)

- Lesson 2: writing tests *before* code takes *more time* up-front, but often *less time* overall

TDD Summary

- Red – Green – Refactor, and always have working code
- Test *one* behavior at a time, using seams
- Use `it` "placeholders" or `pending` to note tests you know you'll need
- Read & understand coverage reports
- "Defense in depth": don't rely too heavily on any *one* kind of test

Which non-obvious statement about testing is FALSE?

- ☐ Even 100% test coverage is not a guarantee of being bug-free
- ☐ If you can stimulate a bug-causing condition in a debugger, you can capture it in a test
- ☐ Testing eliminates the need to use a debugger
- ☐ When you change your code, you need to change your tests as well

Plan-And-Document Perspective on Software Testing

(Engineering Software as a Service §8.10)

David Patterson

P&D Testing?

- BDD/TDD writes tests before code
  - When do P&D developers write tests?
- BDD/TDD starts from user stories
  - Where do P&D developers start?
- BDD/TDD developers write tests & code
  - Does P&D use same or different people for testing and coding?
- What does the Testing Plan and Testing Documentation look like?

P&D Project Manager

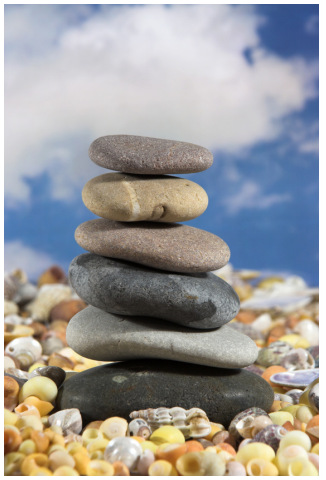
- P&D depends on Project Managers
- Document project management plan
- Creates *Software Requirements Specification* (SRS)
  - Can be 100s of pages
  - IEEE standard to follow
- Must document Test Plan
  - IEEE standard to follow

P&D Approach to Testing

- Manager divides SRS into programming units
- Developers code units
- Developers perform unit testing
- Separate Quality Assurance (QA) team does higher level tests:
  - Module, Integration, System, Acceptance

3 QA Integration Options

1. Top-down integration
   - Starts top of dependency graph
   - High-level functions (UI) work soon
   - Many stubs to get app to “work”
2. Bottom-up integration
   - Start bottom of dependency graph
   - No stubs, integrate everything in a module
   - Can’t see app working until all code written & integrated

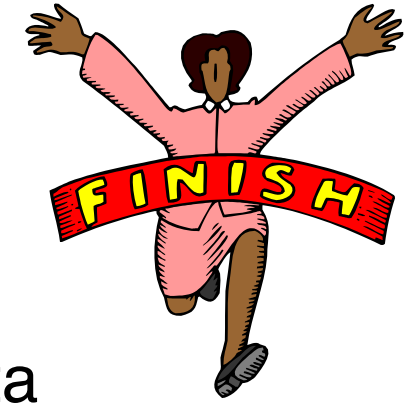


3 Integration Options

3. Sandwich integration
   - Best of both worlds?
   - Reduce stubs by integrating some units bottom up
   - Try to get UI operational by integrating some units top down

QA Team Testing

- Next QA Team does System Test
  - Full app should work
  - Test non-functional requirements (performance) + functional requirements (features in SRS)
- When P&D System Testing Done?
  - Organization policy
    - Eg, test coverage level (all statements)
    - Eg, all inputs tested with good and bad data

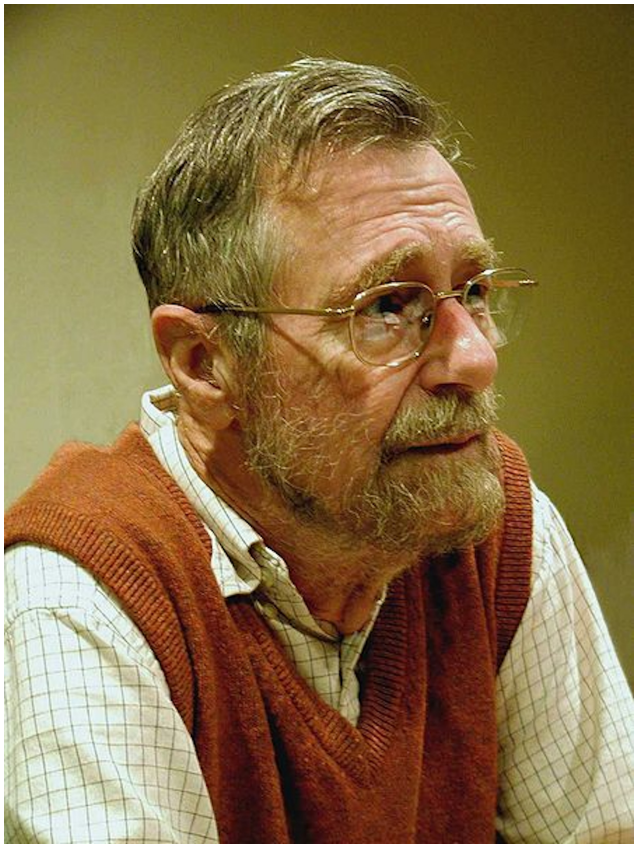


- Final step: Customer or User Acceptance tests (UAT)—validation vs. verification

Limits of Testing

- Program testing can be used to show the presence of bugs, but never to show their absence!
  - Edsger W. Dijkstra

(received the 1972 Turing Award for fundamental contributions to developing programming languages)



Formal Methods

- Start with formal specification & prove program behavior follows spec.

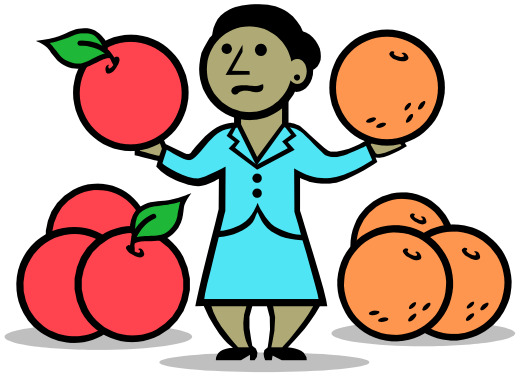
1. Human does proof
2. Computer via automatic theorem proving
   - Uses inference + logical axioms to produce proofs from scratch

3. Computer via model checking
   - Verifies selected properties by exhaustive search of all possible states that a system could enter during execution

Formal Methods

- Computationally expensive, so use
  - Small, fixed function
  - Expensive to repair, very hard to test
  - Eg. Network protocols, safety critical SW
- Biggest: OS kernel
  10K LOC @ $500/LOC
  - NASA SW $80/LOC
- This course: rapidly changing SW (SaaS), easy to repair, easy to test
  => no formal methods

SW Testing: P&D vs. Agile (Fig. 8.26)



Tasks	*In Plan and Document*	*In Agile*
Test Plan and Documentation	Software Test Documentation such as IEEE Standard 829-2008	User stories
Order of Coding and Testing	1. Code units 2. Unit test 3. Module test 4. Integration test 5. System test 6. Acceptance test	1. Acceptance test 2. Integration test 3. Module test 4. Unit test 5. Code units
Testers	Developers for unit tests; QA testers for module, integration, system, and acceptance tests	Developers
When Testing Stops	Company policy (e.g., statement coverage, happy and sad user inputs)	All tests pass (green)

Which statement regarding testing is FALSE?

1. Formal methods are expensive but worthwhile to verify important applications
2. P&D developers code before they write tests while its vice versa for Agile developers
3. Agile developers perform module, integration, system, & acceptance tests; P&D developers don't
4. Sandwich integration in P&D aims to reduce effort making stubs while trying to get general functionality early