a tale of
testing frameworks

Platform 45

RSpec

(BDD)

minitest

(TDD)

RSpec

RSpec is a powerful testing DSL for writing executable specifications that describe the behaviour of the system under test.

Minitest is a suite of simple testing tools that provide support for xUnit and spec style tests, and mocking and benchmarking.

minitest

RSpec

Behaviour Driven Development

rspec/rspec

- rspec/rspec-core (cli & DSL)
- rspec/rspec-expectations (matchers)
- rspec/rspec-mocks (mocks & stubs)

minitest

- minitest/autorun (test runner)
- minitest/unit (test system)
- minitest/spec (spec system)
- minitest/mock (mocks & stubs)
- minitest/benchmark (assert performance)
- minitest/pride (coloured test reporter)

xUnit

Style

Testing

minitest

RSpec

```ruby
RSpec.describe Foo do
  describe "#bar" do
    context "when called with parameters x or in state y" do
      it "behaves a certain way, plays well with others, then " \
         "returns a value x or causes a side effect y" do
        foo = Foo.new
        expect(foo.work).to eq(Bar.new(1, 2))
      end
    end
  end
end
```

RSpec

```
RSpec.describe Order do
  describe "#total" do
    it "sums the prices of the items in its line items" do
      order = Order.new
      order.add_entry(LineItem.new(item: Item.new(
        price: Money.new(1.11, :USD)
      )))
      order.add_entry(LineItem.new(item: Item.new(
        price: Money.new(2.22, :USD),
        quantity: 2
      )))
      expect(order.total).to eq(Money.new(5.55, :USD))
    end
  end
end
```

RSpec

```
RSpec.describe #=> class TestClass
  describe
    context
      it
```

RSpec

```
RSpec.describe #=> class TestClass
  describe     #=> class SubTestClass < TestClass
    context
      it
```

RSpec

```
RSpec.describe #=> class TestClass
  describe     #=> class SubTestClass < TestClass
    context    #=> class SubSubTestClass < SubTestClass
      it
```

RSpec

```
RSpec.describe #=> class TestClass
  describe     #=> class SubTestClass < TestClass
    context    #=> class SubSubTestClass < SubTestClass
      it       #=>   def test_behaves_a_certain_way…
               #=>     # work...
               #=>   end
               #=> end
```

RSpec

```
RSpec.describe #=> class TestClass
  before {}    #=>   def setup
               #=>   end
  describe     #=> class SubTestClass < TestClass
    context    #=> class SubSubTestClass < SubTestClass
      it       #=>   def test_behaves_a_certain_way…
               #=>     # work...
               #=>   end
               #=> end
```

RSpec

```
RSpec.describe #=> class TestClass
  before {}    #=>   def setup
               #=>   end
  describe     #=> class SubTestClass < TestClass
    context    #=> class SubSubTestClass < SubTestClass
    before {}  #=>   def setup
               #=>     super
               #=>   end
      it       #=>   def test_behaves_a_certain_way...
               #=>     # work...
               #=>   end
               #=> end
```

```ruby
class Meme
  def i_can_has_cheezburger?
    "OHAI!"
  end
end

require "minitest/autorun"

class MemeTest < Minitest::Test
  def test_that_kitty_can_eat
    meme = Meme.new
    assert_equal "OHAI!", meme.i_can_has_cheezburger?
  end
end
```

minitest

```ruby
class Meme
  def i_can_has_cheezburger?
    "OHAI!"
  end

  def doge
    "WOW"
  end
end

require "minitest/autorun"

class TestMeme < Minitest::Test
  def setup
    @meme = Meme.new
  end

  def test_that_kitty_can_eat
    assert_equal "OHAI!", @meme.i_can_has_cheezburger?
  end

  def test_such_is_wow
    assert_equal "WOW", @meme.doge
  end
end
```

minitest

Why the switch?

1. A good fit for our clients developers

No DSL experience (non-rubyists and non-railsers)

Team was familiar with xUnit style testing

Lower barrier to entry with simpler framework

Why the switch?

2. Curiosity about the other side

We aren’t using RSpec tests as specification at P45

Legacy projects had stupid long spec files

Deeply nested context/describes difficult to follow

Why the switch?

3. Some moar!

Just test your code, it doesn't matter what with

Minitest is just ruby, so ruby all the things

Minitest is faster (RSpec matchers create objects)

Why the switch?

4. Can even have moar! Rawr!

RSpec matchers are confusing: ==, eq, eql, equal

Let is clever but a little too clever for me

Example Code

https://github.com/koblenski/sample_app_rails_4

http://interblah.net/how-rspec-works

Contrasts between RSpec & *minitest*

contrast
noun |ˈkɒntrɑːst| [mass noun]

the state of being strikingly different from something else in juxtaposition or close association: *the day began cold and blustery,* ***in contrast to*** *almost two weeks of uninterrupted sunshine* | [count noun] *: a* ***contrast between*** *rural and urban trends* | *Kos is an island of contrasts.*

Contrast I

Tooling

RSpec

CLI runner and reporter

```
# run all the tests
$ rspec spec

# run a single test file
$ rspec spec/controllers/sessions_controller_spec.rb

# run a single test case
$ rspec spec/controllers/sessions_controller_spec.rb:45

# run a subset of tests
$ rspec spec/services
```

RSpec

CLI runner and reporter

```
# --example
$ rspec spec --example some_test_name_here

# --format
$ rspec spec --format=doc
$ rspec spec --format=dots
$ rspec spec --format=progress

# --tag
it "ordinary example", slow: true do; end

$ rspec spec --tag slow

it "ordinary example", speed: 'slow' do; end

$ rspec spec --tag speed:slow
```

```
# --order / --seed
$ rspec spec --order rand:1234
$ rspec spec --seed 1234

# --dry-run
$ rspec spec --dry-run

# --fail-fast
$ rspec spec --fail-fast

# --init
# --color
# --pattern
# --require
# --warnings
```

RSpec

CLI runner and reporter

```
# in .rspec

--color
--format=doc
```

Minitest
Ruby and Regex

```
# You need rake, but you probably already use it

# run all the tests
$ ruby -Ilib:test test
$ rake test

# run a single test file
$ ruby -Ilib:test test/controllers/sessions_controller_test.rb
$ rake test TEST=test/controllers/sessions_controller_test.rb

# run a single test case
# https://github.com/qrush/m
$ m test/controllers/sessions_controller_test.rb:45
# => -n "/^(test_\\#valid\\?)$/"

# run subset of the tests
$ rake test TESTOPTS="--name=/regression/"
```

Minitest
Ruby and Regex

```
# Some reporters from https://github.com/kern/minitest-reporters

Minitest::Reporters::DefaultReporter  # => Redgreen-capable version of standard Minitest reporter
Minitest::Reporters::SpecReporter     # => Turn-like output that reads like a spec
Minitest::Reporters::ProgressReporter # => Fuubar-like output with a progress bar
Minitest::Reporters::RubyMateReporter # => Simple reporter designed for RubyMate
Minitest::Reporters::RubyMineReporter # => Reporter designed for RubyMine IDE and TeamCity CI server
Minitest::Reporters::JUnitReporter    # => JUnit test reporter designed for JetBrains TeamCity


# in test/test_helper.rb

require 'minitest/reporters'

# Configure sweet minitest output
Minitest::Reporters.use!(
  Minitest::Reporters::DefaultReporter.new,
  ENV,
  Minitest.backtrace_filter
)
```

Minitest
Ruby and Regex

```
# run subdirectories

$ rake test:serializers
$ rake test:queries


# run custom defined groups

$ rake test:features
$ rake test:unit
```

```ruby
# run a subset of tests

namespace :test do
  sections = %i(controllers features helpers interactions jobs mailers
                models policies queries serializers services uploaders)
  sections.each do |section|
    desc "Runs tests in test/#{section} folder"
    Rails::TestTask.new(section) do |t|
      t.pattern = "test/#{section}/**/*_test.rb"
    end
  end

  unit_test_sections = sections - [:features]
  desc "Runs unit tests (everything except test/features folder)"
  Rails::TestTask.new(:unit) do |t|
    t.pattern = "test/{#{unit_test_sections.join(',')}}/**/*_test.rb"
  end
end
```

Minitest
Ruby and Regex

```
% ruby -Ilib:test test/foos/foo_test.rb --help
minitest options:
    -h, --help                       Display this help.
    -s, --seed SEED                  Sets random seed
    -v, --verbose                    Verbose. Show progress processing files.
    -n, --name PATTERN               Filter run on /pattern/ or string.

Known extensions: pride, autotest
    -p, --pride                      Pride. Show your testing pride!
    -a, --autotest                   Connect to autotest server.
```

Contrast II

Shared Tests

RSpec
Shared examples

```ruby
# in spec/support/shared/authenticated_routes.rb
RSpec.shared_examples "an authenticated route" do |http_method, path, params_hash = nil|
  context "when not authenticated" do
    it "redirects to the login page" do
      if params_hash
        expect( public_send(http_method, *[path, params_hash] )).to redirect_to(new_session_path)
      else
        expect( public_send(http_method, path) ).to redirect_to(new_session_path)
      end
    end
  end
end
```

RSpec
Shared examples

```ruby
# in spec/controllers/card_readers_controller_spec.rb
require 'rails_helper'

describe CardReadersController do
  it_behaves_like "an authenticated route", :get, :index
  it_behaves_like "an authenticated route", :get, :show
end
```

Minitest
Ruby modules

```ruby
# in test/support/shared/controller_auth_test.rb
module Shared
  module ControllerAuthTest
    def shared_authentication_test(name, &block)
      define_method "test_#{name}_is_authenticated" do
        request.headers['Authorization'] = nil
        instance_exec(&block)
        assert_response :unauthorized
      end
    end
  end
end
```

Minitest
Ruby modules

```ruby
# in test/controllers/api/v1/users_controller_test.rb
require 'test_helper'

class Api::V1::UsersControllerTest < ActionController::TestCase
  shared_authentication_test("create") { post :create }
  shared_authentication_test("destroy") { delete :destroy, id: 1 }
end
```

Contrast III

Configuration

RSpec
DSL include

```ruby
# in spec/spec_helper.rb
RSpec.configure do |config|
  config.include FactoryGirl::Syntax::Methods               # include in base test class
  config.include Devise::TestHelpers, type: :controller   # include in controller test class
  config.include Features::SessionHelpers, type: :feature # include in feature test class
end
```

Minitest
Ruby include

```ruby
# in test/test_helper.rb
# include in base test class
class ActiveSupport::TestCase
  include DescribedClass
  include FactoryGirl::Syntax::Methods
end

# include in controller test class
class ActionController::TestCase
  include ResponseParser
  extend Shared::ControllerAuthTest
end

# include in feature test class
class Capybara::Rails::TestCase
  include Features::AuthHelper
end
```

Contrast IV

Matchers

RSpec
Matchers

```ruby
# in spec/models/pay_fast_itn.rb
require 'spec_helper'

describe PayFastITN, type: :model do
  let(:itn_complete) { PayFastITN.new(some_data) }

  it "should generate a signature matching PayFast's returned signature" do
    expect(itn_complete.signature).to eq 'fbdeb4e5d67fc9ac07b96a8217285777'
    #                                  eql 'fbdeb4e5d67fc9ac07b96a8217285777'
    #                                  equal 'fbdeb4e5d67fc9ac07b96a8217285777'
    #                                  == 'fbdeb4e5d67fc9ac07b96a8217285777'

    # Bonus Q: which one?
    # expect(true).to eq true
    # expect(true).to be true
    # expect(true).to eq truthy
    # expect(true).to be truthy
  end
end
```

Minitest
Assertions

```ruby
# in test/models/user_test.rb
require 'test_helper'

class UserTest < ActiveSupport::TestCase
  test "#fullname joins the first & last names" do
    user = FG.build(:user, first_name: 'That', last_name: ‘Guy')
    assert "That Guy” == user.fullname
    # or
    assert_equal "That Guy", user.fullname
  end
end
```

Contrast V

Contexts

RSpec
Context blocks

describe == context

```ruby
# in spec/controllers/storefront/gifts_controller_spec.rb
require 'spec_helper'

describe Storefront::GiftsController do
  let(:user) { create(:valid_user) }

  describe "POST create" do
    context "with valid params” # ...
    context "with invalid params" do
      before { sign_in user }

      it "redirects to accounts page & sets error flash" do
        post :create, { purchase_gift: { credit_attributes: {} } }
        expect(response).to redirect_to(account_path)
        expect(flash[:error]).to_not be_blank
      end
    end
  end

  describe "PUT update" do
    context "with valid params” # ...
    context "with invalid params" # ...
  end
end
```

RSpec
Context blocks

describe == context

```
# in spec/controllers/storefront/gifts_controller_spec.rb
require 'spec_helper'

describe Storefront::GiftsController do
  let(:user) { create(:valid_user) }

  context "POST create" do
    context "with valid params” # ...
    context "with invalid params" do
      before { sign_in user }

      it "redirects to accounts page and sets error message" do
        post :create, { purchase_gift: { credit_attributes: {} } }
        expect(response).to redirect_to(account_path)
        expect(flash[:error]).to_not be_blank
      end
    end
  end

  context "PUT update" do
    context "with valid params” # ...
    context "with invalid params" # ...
  end
end
```

Minitest

It's called a "directory" (you probably haven't heard of it)

```ruby
# in test/controllers/users_controller/show_test.rb
require 'test_helper'

class Api::V1::UsersController::ShowTest < ActionController::TestCase
  include AuthHelper::LoginAdmin

  test "GET show returns the serialized user" do
    user = FG.create(:user)
    get :show, { id: user.id }
    assert_response :success
    assert_equal json, UserSerializer.new(user).as_json
  end
end
```

Minitest

It's called a "directory" (you probably haven't heard of it)

```ruby
# in test/controllers/users_controller/show_test.rb
require 'test_helper'

class Api::V1::UsersController::ShowTest < ActionController::TestCase
  include AuthHelper::LoginAdmin

  test "GET show returns the serialized user" do
    user = FG.create(:user)
    get :show, { id: user.id }
    assert_response :success
    assert_equal json, UserSerializer.new(user).as_json
  end
end
```

Minitest

It's called a "directory" (you probably haven't heard of it)

```ruby
# in test/controllers/users_controller/destroy_test.rb
require 'test_helper'

class Api::V1::UsersController::DestroyTest < ActionController::TestCase
  include AuthHelper::LoginAdmin

  test "DELETE destroy returns no content" do
    user = FG.create(:user)
    get :show, { id: user.id }
    assert_response :no_content
    assert response.body.empty?
  end
end
```

Minitest

It's called a "directory" (you probably haven't heard of it)

```ruby
# in test/controllers/users_controller/destroy_test.rb
require 'test_helper'

class Api::V1::UsersController::DestroyTest < ActionController::TestCase
  include AuthHelper::LoginAdmin

  test "DELETE destroy returns no content" do
    user = FG.create(:user)
    get :show, { id: user.id }
    assert_response :no_content
    assert response.body.empty?
  end
end
```

Minitest

It's called a "directory" (you probably haven't heard of it)

```ruby
# in test/controllers/users_controller/destroy_test.rb
require 'test_helper'

class Api::V1::UsersController < ActionController::TestCase
  include AuthHelper::LoginAdmin

  test "DELETE destroy returns no content" do
    user = FG.create(:user)
    get :show, { id: user.id }
    assert_response :no_content
    assert response.body.empty?
  end
end
```

Contrast VI

Mocks & Stubs

Stubs

“Stubs provide canned answers to method calls made during the test”.

Mocks

“Mocks are pre-programmed with expectations which form a specification of the calls they are expected to receive. They can throw an exception if they receive a call they don't expect and are checked during verification to ensure they got all the calls they were expecting.”

www.martinfowler.com/bliki/TestDouble.html

Stubs

"We replace a real object with a test-specific object that feeds the desired indirect inputs into the SUT."

Mocks

"We replace an object the SUT depends on with a test-specific object that verifies it is being used correctly by the SUT."

Spies

“We use a Test Double to capture the indirect output calls made to another component by the SUT for later verification by the test.”

SUT is an abbreviation for “system under test”

http://blog.firsthand.ca/2012/03/testing-terminology-spies-stubs-and.html

the basic idea behind stubs

I. Setup stub objects for collaborators that you want to isolate the object under test from

II. Inject stubs into object under test to feed it indirect input

III. Run the code to be tested

IV. Yipee side effect free test!!!

the basic idea behind mocks

I. Setup expectations/assertions about the collaborators using mock objects

II. Inject mocks into object under test

III. Run the code to be tested

IV. Verify expectations/assertions from step I

minitest mocks are cool

```ruby
class MemeAsker
  def initialize(meme)
    @meme = meme
  end

  def ask(question)
    method = question.tr(" ", "_") + "?"
    @meme.__send__(method)
  end
end

require "minitest/autorun"

describe MemeAsker do
  describe "#ask" do
    context "when passed an unpunctuated question" do
      it "invokes the appropriate predicate method on the meme" do
        @meme = Minitest::Mock.new
        @meme.expect :will_it_blend?, :return_value

        @meme_asker = MemeAsker.new @meme
        @meme_asker.ask "will it blend"

        @meme.verify
      end
    end
  end
end
```

what a spy would look like

```
class MemeAsker
  def initialize(meme)
    @meme = meme
  end

  def ask(question)
    method = question.tr(" ", "_") + "?"
    @meme.__send__(method)
  end
end

require "minitest/autorun"

describe MemeAsker do
  describe "#ask" do
    context "when passed an unpunctuated question" do
      it "invokes the appropriate predicate method on the meme" do
        @meme = Minitest::Spy.new


        @meme_asker = MemeAsker.new @meme
        @meme_asker.ask "will it blend”

        @meme.expect :will_it_blend?, :return_value
      end
    end
  end
end
```

RSpec mocks & Mocha

RSpec
doubles/stubs

```ruby
# Class under test
class CreateScheduleOccurrencesJob < BaseJob
  def perform(schedule, occurrences, generator_service: OccurrenceGeneratorService)
    generator_service.new(schedule, occurrences: occurrences).call
  end
end

# in spec/jobs/create_schedule_occurrences_job_spec.rb
require 'rails_helper'

RSpec.describe CreateScheduleOccurrencesJob do
  describe "#perform" do
    it "delegates work to a service object" do
      schedule = create(:schedule)
      occurrences = []

      generator_mock_instance = double('service mock instance')
      expect(generator_mock_instance).to receive(:call).and_return(nil)

      generator_mock = double('generator service mock')
      expect(generator_mock).to receive(:new).and_return(generator_mock_instance)

      described_class.perform_now(schedule, occurrences, generator_service: generator_mock)
    end
  end
end
```

RSpec
doubles/stubs

```ruby
# Class under test
class CreateScheduleOccurrencesJob < BaseJob
  def perform(schedule, occurrences, generator_service: OccurrenceGeneratorService)
    generator_service.new(schedule, occurrences: occurrences).call
  end
end

# in spec/jobs/create_schedule_occurrences_job_spec.rb
require 'rails_helper'

RSpec.describe CreateScheduleOccurrencesJob do
  describe "#perform" do
    it "delegates work to a service object" do
      schedule = create(:schedule)
      occurrences = []

      generator_mock_instance = double('service mock instance', call: nil)
      generator_mock = double('generator service mock', new: generator_mock_instance)

      described_class.perform_now(schedule, occurrences, generator_service: generator_mock)
    end
  end
end
```

(shorthand)

Minitest
mocks/stubs

```ruby
# Class under test
class CreateScheduleOccurrencesJob < BaseJob
  def perform(schedule, occurrences, generator_service: OccurrenceGeneratorService)
    generator_service.new(schedule, occurrences: occurrences).call
  end
end


# in test/jobs/create_schedule_occurrences_job_test.rb
require 'test_helper'

class CreateScheduleOccurrencesJobTest < ActiveSupport::TestCase
  test "#perform delegates work to a service object" do
    schedule = create(:schedule)
    occurrences = []

    generator_mock_instance = mock('service mock instance')
    generator_mock_instance.expects(:call).returns(nil)

    generator_mock = mock('generator service mock')
    generator_mock.expects(:new).returns(generator_mock_instance)

    described_class.perform_now(schedule, occurrences, generator_service: generator_mock)
  end
end
```

Minitest
mocks/stubs

```ruby
# Class under test
class CreateScheduleOccurrencesJob < BaseJob
  def perform(schedule, occurrences, generator_service: OccurrenceGeneratorService)
    generator_service.new(schedule, occurrences: occurrences).call
  end
end


# in test/jobs/create_schedule_occurrences_job_test.rb
require 'test_helper'

class CreateScheduleOccurrencesJobTest < ActiveSupport::TestCase
  test "#perform delegates work to a service object" do
    schedule = create(:schedule)
    occurrences = []

    generator_mock_instance = mock('service mock instance', call: nil)
    generator_mock = mock('generator service mock', new: generator_mock_instance)

    described_class.perform_now(schedule, occurrences, generator_service: generator_mock)
  end
end
```

(shorthand)

Contrast VII

Feature Tests

RSpec

Capybara and RSpec expectations

```ruby
# in Gemfile
group :test do
  gem 'capybara'
  gem "capybara-webkit"
end
```

```ruby
# in spec/spec_helper.rb
require ‘capybara/rspec'

Capybara.javascript_driver = :webkit

RSpec.configure do |config|
  config.include FeatureHelpers, type: :feature
end
```

Minitest
Capybara with assertions

```ruby
# in Gemfile
group :test do
  gem 'minitest-rails-capybara', '~> 2.1.1'
  # Execute javascript in feature specs
  gem 'poltergeist', '~> 1.6.0'
end
```

```ruby
# in test/test_helper.rb
require 'minitest/rails/capybara'
require 'capybara/poltergeist'

Capybara.javascript_driver = :poltergeist

class Capybara::Rails::TestCase
  include Features::AuthHelpers
  include Features::JsHelpers
  include Features::AssertionHelpers
end
```

RSpec

Capybara and RSpec expectations

```ruby
# in spec/features/user_logs_in.rb
require 'rails_helper'

feature 'User logs in' do
  scenario 'with the correct login details' do
    visit new_session_path

    fill_in 'login_username', with: 'test'
    fill_in 'login_password', with: 'password'
    click_button 'Log In'

    expect(current_path).to eq home_path
  end
end
```

Minitest

Capybara with assertions

```ruby
# in test/features/login_authentication.rb
require "test_helper"

feature "Login authentication" do
  context "with an admin role" do
    scenario "a user can login" do
      user = FG.create(:user, roles: ['admin'])

      visit manage_root_path
      assert page.has_content? "Login"

      fill_in 'Email', with: user.email
      fill_in 'Password', with: user.password
      click_button 'Login'

      assert page.has_content? "Signed in successfully."
    end
  end
end
```

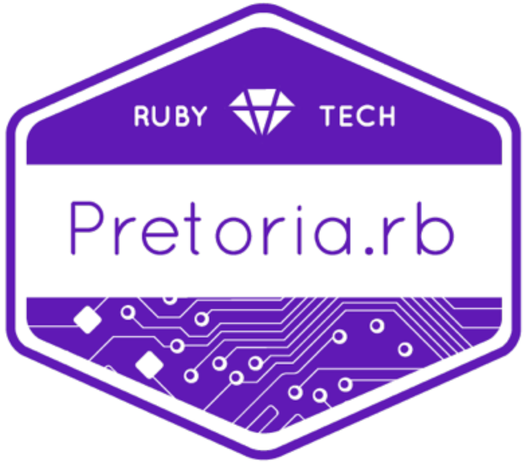
Recap

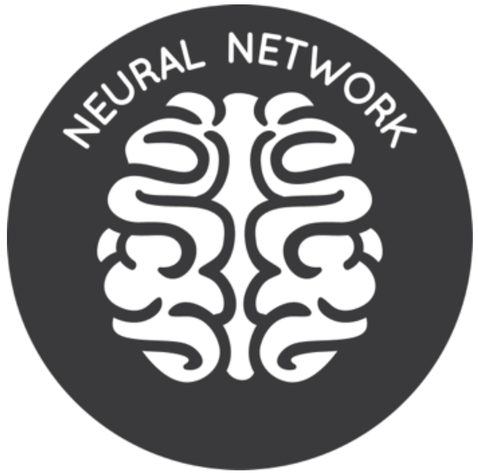
I. Tooling
II. Shared Tests
III. Configuration
IV. Matchers vs Assertions
V. Contexts vs Directories
VI. Mocks and Stubs
VII. Feature tests

fin

Simon van Dyk

twitter @siefi
github @sighmin
email simon.vandyk@gmail.com






thanks for
joining us
jozi ruby