TESTING WITH RSPEC

Introduction

• LEVEL 1 •

RSpec

- Popular Ruby testing framework
- Thriving community
- Less Ruby-like, natural syntax
- Well-formatted output

David Chelimsky

LEVEL 1 - INTRODUCTION

Behavior-Driven Dev

- Describe your application's behavior
  - Can be done with other frameworks
  - BDD is not required
  - But encouraged by the RSpec syntax
  - And RSpec makes it elegant and readable

Installing RSpec

```
$ gem install rspec
...
Successfully installed rspec-core
Successfully installed rspec-expectations
Successfully installed rspec-mocks
Successfully installed rspec
4 gems installed
```

```
$ rspec --init
  create   spec/spec_helper.rb
  create   .rspec
```

in your project directory

more about configuration in level 2

Describe block

Our project source will live in /lib/zombie.rb

but lets write a test first

spec/lib/zombie_spec.rb <name_of_spec>_spec.rb

```
require "spec_helper"
describe "A Zombie" do
  # your examples (tests) go here
end
```


Describe it

spec/lib/zombie_spec.rb

```
require "spec_helper"
describe "A Zombie" do
  it "is named Ash"
end
```

name of the example

examples are declared using the "it" method

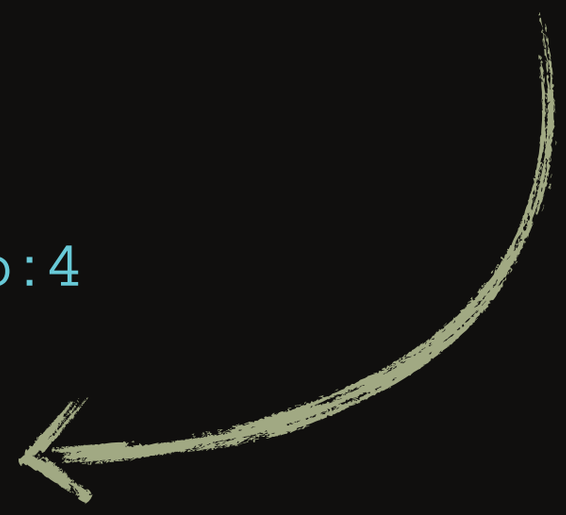
Pending

spec/lib/zombie_spec.rb

```
require "spec_helper"
describe "A Zombie" do
  it "is named Ash"
end
```

```
$ rspec spec/lib/zombie_spec.rb
*
Pending:
  A Zombie is named "Ash"
    # Not yet implemented
      # ./spec/lib/zombie_spec.rb:4
Finished in 0.00028 seconds
1 example, 0 failures, 1 pending
```

Pending



Describe Class

spec/lib/zombie_spec.rb

```ruby
require "spec_helper"
describe Zombie do
  it "is named Ash"
end
```

class we want to test

```
$ rspec spec/lib/zombie_spec.rb
zombie_spec.rb:16:in `<top (required)>':
uninitialized constant Zombie (NameError)
```

Failing

we don't have a Zombie class yet

Creating the class

spec/lib/zombie_spec.rb

```ruby
require "spec_helper"
require "zombie"

describe Zombie do
  it "is named Ash"
end
```

lib/zombie.rb

```ruby
class Zombie

end
```

```
$ rspec spec/lib/zombie_spec.rb
*
Pending:
  Zombie is named "Ash"
    # Not yet implemented
      # ./spec/lib/zombie_spec.rb:17
Finished in 0.00028 seconds
1 example, 0 failures, 1 pending
```

Pending

Expectations

spec/lib/zombie_spec.rb

```
describe Zombie do
  it "is named Ash" do
    zombie = Zombie.new
    zombie.name.should == 'Ash'
  end
end
```

Test::Unit

```
assert_equal 'Ash', zombie.name
```

expectation

- This is how you ‘assert’ in RSpec
- Assertions are called ‘expectations’
- They read more like plain English

Test properly fails

spec/lib/zombie_spec.rb

```ruby
describe Zombie do
  it "is named Ash" do
    zombie = Zombie.new
    zombie.name.should == 'Ash'
  end
end
```

```
$ rspec spec/lib/zombie_spec.rb

1) Zombie is named "Ash"
     Failure/Error: zombie.name.should == 'Ash'
     NoMethodError:
       undefined method `name' for #<Zombie>

Finished in 0.00125 seconds
1 example, 1 failure
```

Make it pass

spec/lib/zombie_spec.rb

```ruby
describe Zombie do
  it "is named Ash" do
    zombie = Zombie.new
    zombie.name.should == 'Ash'
  end
end
```

lib/zombie.rb

```ruby
class Zombie
  attr_accessor :name

  def initialize
    @name = 'Ash'
  end
end
```

```
$ rspec spec/lib/zombie_spec.rb
.

Finished in 0.00047 seconds
1 example, 0 failures
```

Passed!

Another expectation

spec/lib/zombie_spec.rb

```ruby
describe Zombie do

  ...
  it "has no brains" do
    zombie = Zombie.new
    zombie.brains.should < 1
  end
end
```

matcher

modifier

```
$ rspec spec/lib/zombie_spec.rb

1) Zombie is named "Ash"
     Failure/Error: zombie.brains.should < 1
     NoMethodError:
       undefined method `brains' for #<Zombie>

Finished in 0.00125 seconds
1 example, 1 failure
```

Make it pass

spec/lib/zombie_spec.rb

```ruby
describe Zombie do

  ...
  it "has no brains" do
    zombie = Zombie.new
    zombie.brains.should < 1
  end
end
```

lib/zombie.rb

```ruby
class Zombie
  attr_accessor :name, :brains

  def initialize
    @name = 'Ash'
    @brains = 0
  end
end
```

```
$ rspec spec/lib/zombie_spec.rb
..

Finished in 0.00045 seconds
2 example, 0 failures
```

Passed!

Other matchers

```
zombie.name.should == 'Ash'
```

```
zombie.alive.should == false
```

```
zombie.alive.should be_false
```

```
zombie.rotting.should == true
```

```
zombie.rotting.should be_true
```

```
zombie.height.should > 5
```

```
zombie.brains.should be < 1
```

```
zombie.height.should >= 5
```

```
zombie.height.should < 5
```

```
zombie.height.should_not == 5
```

Testing a predicate

/lib/zombie.rb

```ruby
class Zombie
  def hungry?
    true
  end
end
```

predicate

/spec/lib/zombie_spec.rb

```ruby
describe Zombie do
  it 'is hungry' do
    zombie = Zombie.new
    zombie.hungry?.should == true
  end
end
```

could read better

Predicate 'be'

/spec/lib/zombie_spec.rb

```ruby
describe Zombie do
  it 'is hungry' do
    zombie = Zombie.new
    zombie.hungry?.should == true
  end
end
```

```ruby
zombie.hungry?.should be_true
```

```ruby
zombie.should be_hungry
```

predicate matcher

Test passes

/spec/lib/zombie_spec.rb

```
describe Zombie do
  it 'is hungry' do
    zombie = Zombie.new
    zombie.should be_hungry
  end
end
```

```
$ rspec spec/lib/zombie_spec.rb
 .

 Finished in 0.00045 seconds
 1 example, 0 failures
```

Still passing!

To mark as pending

to-do list

```
it "is named Ash"
```

```
xit "is named Ash" do
  ...
end
```

Useful for Debugging

```
it "is named Ash" do
  pending
  ...
end
```


Configuration & Matchers

• LEVEL 2 •

Installing RSpec

```
$ gem install rspec
Fetching: rspec-core.gem (100%)
Fetching: rspec-expectations.gem (100%)
Fetching: rspec-mocks.gem (100%)
Fetching: rspec.gem (100%)
Successfully installed rspec-core
Successfully installed rspec-expectations
Successfully installed rspec-mocks
Successfully installed rspec
4 gems installed

$ rspec --init
  create   spec/spec_helper.rb
  create   .rspec
```

in your project directory

Inside Rails

```
group :development, :test do
  gem 'rspec-rails'
end
```

in your Gemfile

```
$ bundle install
...
Installing rspec-core
Installing rspec-expectations
Installing rspec-mocks
Installing rspec
Installing rspec-rails

$ rails generate rspec:install
  create .rspec
  create spec/spec_helper.rb
```

not just rspec core

different with Rails

Configuration

spec/spec_helper.rb

```
Dir[Rails.root.join("spec/support/**/*.rb")].each {|f| require f}
...
RSpec.configure do |config|
  config.mock_with :mocha
  ...
end
```

requires all helper files within spec/support

allows you to change the default mocking framework

LEVEL 2 — CONFIGURATION & MATCHERS

Output

```
$ rspec --color spec/models/zombie_spec.rb
.
Finished in 0.00176 seconds
1 examples, 0 failures
```

```
$ rspec --color --format documentation spec/models/zombie_spec.rb

Zombie
  is invalid without a name
.
Finished in 0.00052 seconds
1 examples, 0 failures
```

.rspec

```
--color
--format documentation
```

makes it a default

Running specs

```
$ rspec
```

running all the '_spec.rb' files within /spec

```
$ rspec spec/models/
```

running a specific directory

```
$ rspec spec/models/zombie_spec.rb
```

running a specific test

```
$ rspec spec/models/zombie_spec.rb:4
```

running a specific line

Model spec

spec/models/zombie_spec.rb

```ruby
require 'spec_helper'
describe Zombie do
  it 'is invalid without a name' do
    zombie = Zombie.new
    zombie.should_not be_valid
  end
end
```

predicate matcher

app/models/zombie.rb

```ruby
class Zombie < ActiveRecord::Base
  validates :name, presence: true
end
```

```
$ rspec spec/models/zombie_spec.rb
.
Finished in 0.00176 seconds
1 examples, 0 failures
```

Matchers: match

spec/models/zombie_spec.rb

```
describe Zombie do
  it "has a name that matches 'Ash Clone'" do
    zombie = Zombie.new(name: "Ash Clone 1")
    zombie.name.should match(/Ash Clone \d/)
  end
end
```

takes a regular expression

Matchers: include

spec/models/zombie_spec.rb

```
describe Zombie do
  it 'include tweets' do
    tweet1 = Tweet.new(status: 'Uuuuunhhhhh')
    tweet2 = Tweet.new(status: 'Arrrrgggg')
    zombie = Zombie.new(name: 'Ash', tweets: [tweet1, tweet2])
    zombie.tweets.should include(tweet1)
    zombie.tweets.should include(tweet2)
  end
end
```

matching on an array

is this Tweet inside tweets?

Matchers: have

spec/models/zombie_spec.rb

```rb
describe Zombie do
  it 'starts with two weapons' do
    zombie = Zombie.new(name: 'Ash')
    zombie.weapons.count.should == 2
  end
end
```

reads much better

```rb
describe Zombie do
  it 'starts with two weapons' do
    zombie = Zombie.new(name: 'Ash')
    zombie.should have(2).weapons
  end
end
```

these will work too

```rb
have(n)
have_at_least(n)
have_at_most(n)
```


Matchers: change

spec/models/zombie_spec.rb

```ruby
describe Zombie do
  it 'changes the number of Zombies' do
    zombie = Zombie.new(name: 'Ash')
    expect { zombie.save }.to change { Zombie.count }.by(1)
  end
end
```

runs before and after expect

results are compared

give these a shot

```ruby
by(n)
from(n)
to(n)
```

you can chain them

```ruby
.from(1).to(5)
```


raise_error

spec/models/zombie_spec.rb

```ruby
describe Zombie do
  it 'raises an error if saved without a name' do
    zombie = Zombie.new
    expect { zombie.save! }.to raise_error(
      ActiveRecord::RecordInvalid
    )
  end
end
```

optionally pass in an exception

```
to
not_to
to_not
```

these modifiers also work

More matchers

```
respond_to(:<method_name>)
```

```
be_within(<range>).of(<expected>)
```

```
exist
```

```
satisfy { <block> }
```

```
be_kind_of(<class>)
```

```
be_an_instance_of(<class>)
```

More matchers

```
@zombie.should respond_to(:hungry?)
```

```
@width.should be_within(0.1).of(33.3)
```

```
@zombie.should exist
```

```
@zombie.should satisfy { |zombie| zombie.hungry? }
```

```
@hungry_zombie.should be_kind_of(Zombie)
```

HungryZombie < Zombie

```
@status.should be_an_instance_of(String)
```

DRY Specs

• LEVEL 3 •

Don't Repeat Yourself

Implicit Subject

spec/models/zombie_spec.rb

```ruby
describe Zombie do
  it 'responds to name' do
    zombie = Zombie.new
    zombie.should respond_to(:name)
  end
end
```

```ruby
describe Zombie do
  it 'responds to name' do
    subject.should respond_to(:name)
  end
end
```

subject = Zombie.new

only works using describe with a class

Implicit Receiver

spec/models/zombie_spec.rb

```ruby
describe Zombie do
  it 'responds to name' do
    subject.should respond_to(:name)
  end
end
```

```ruby
describe Zombie do
  it 'responds to name' do
    should respond_to(:name)
  end
end
```


it without name

```
describe Zombie do
  it 'responds to name' { should respond_to(:name) }
end
```

```
describe Zombie do
  it { should respond_to(:name) }
end
```

example name created automatically

```
$ rspec spec/lib/zombie_spec.rb

Zombie
  should respond to #name
Finished in 0.00125 seconds
1 examples, 0 failures
```

its

spec/lib/zombie_spec.rb

```ruby
describe Zombie do
  it { subject.name.should == 'Ash' }
end
```

```ruby
describe Zombie do
  its(:name) { should == 'Ash' }
end
```

```
$ rspec spec/lib/zombie_spec.rb

Zombie
  should == "Ash"
Finished in 0.00057 seconds
1 examples, 0 failures
```

its examples

```ruby
describe Zombie do
  its(:name) { should == 'Ash' }
  its(:weapons) { should include(weapon) }
  its(:brain) { should be_nil }
  its('tweets.size') { should == 2 }
end
```


Nesting examples

spec/models/zombie_spec.rb

```ruby
describe Zombie do
  it 'craves brains when hungry'
  it 'with a veggie preference still craves brains when hungry'
  it 'with a veggie preference prefers vegan brains when hungry'
end
```

Duplication!

```ruby
describe Zombie do
  it 'craves brains when hungry'
  describe 'with a veggie preference' do
    it 'still craves brains when hungry'
    it 'prefers vegan brains when hungry'
  end
end
```


Nesting examples

```
describe Zombie do
  it 'craves brains when hungry'
  describe 'with a veggie preference' do
    it 'still craves brains when hungry'
    it 'prefers vegan brains when hungry'
  end
end
```

Duplication!

```
describe Zombie do
  describe 'when hungry' do
    it 'craves brains'
    describe 'with a veggie preference' do
      it 'still craves brains'
      it 'prefers vegan brains'
    end
  end
end
```

context

```ruby
describe Zombie do
  describe 'when hungry' do
    it 'craves brains'
    describe 'with a veggie preference' do
      it 'still craves brains'
      it 'prefers vegan brains'
    end
  end
end
```

context instead of describe

```ruby
describe Zombie do
  context 'when hungry' do
    it 'craves brains'
    context 'with a veggie preference' do
      it 'still craves brains'
      it 'prefers vegan brains'
    end
  end
end
```

subject in context

spec/models/zombie_spec.rb

```
...
context 'with a veggie preference' do
  subject { Zombie.new(vegetarian: true) }

  it 'craves vegan brains' do
    craving.should == 'vegan brains'
  end
end
```

```
its(:craving) { should == 'vegan brains' }
```


Using subject

spec/models/zombie_spec.rb

```
context 'with a veggie preference' do
  subject { Zombie.new(vegetarian: true, weapons: [axe]) }
  let(:axe) { Weapon.new(name: 'axe') }

  its(:weapons) { should include(axe) }

  it 'can use its axe' do
    subject.swing(axe).should == true
  end
end
```

unclear what "subject" is,
especially with many tests

Naming the subject

spec/models/zombie_spec.rb

```ruby
context 'with a veggie preference' do
  let(:zombie) { Zombie.new(vegetarian: true, weapons: [axe]) }
  let(:axe) { Weapon.new(name: 'axe') }
  subject { zombie }

  its(:weapons) { should include(axe) }

  it 'can use its axe' do
    zombie.swing(axe).should == true
  end
end
```


New subject syntax

spec/models/zombie_spec.rb

```rb
context 'with a veggie preference' do
  let(:zombie) { Zombie.new(vegetarian: true, weapons: [axe]) }
  let(:axe) { Weapon.new(name: 'axe') }
  subject { zombie }
  ...
```

newer syntax

```rb
context 'with a veggie preference' do
  subject(:zombie) { Zombie.new(vegetarian: true, weapons: [axe]) })
  let(:axe) { Weapon.new(name: 'axe') }
```

step by step subject

```
describe Zombie do

  let(:zombie) { Zombie.create }

  subject { zombie }

  its(:name) { should be_nil? }

  ...
end
```

3. zombie gets created
2. needs to know subject
1. example begins to run

this is an example of Lazy Evaluation

```
it "creates a zombie" { Zombie.count == 1 }
```

wouldn't work!

Let every time

```
describe Zombie do

  let!(:zombie) { Zombie.create }

  subject { zombie }

  its(:name) { should be_nil? }

  ...
end
```

Will create zombie
before every example

Let's refactor

```
describe Zombie do
  it 'has no name' do
    @zombie = Zombie.create
    @zombie.name.should be_nil?
  end

  it 'craves brains' do
    @zombie = Zombie.create
    @zombie.should be_craving_brains
  end

  it 'should not be hungry after eating brains' do
    @zombie = Zombie.create
    @zombie.hungry.should be_true
    @zombie.eat(:brains)
    @zombie.hungry.should be_false
  end
end
```

Let's refactor

```ruby
describe Zombie do
  let(:zombie) { Zombie.create }
  subject { zombie }

  it 'has no name' do
    zombie.name.should be_nil?
  end

  it 'craves brains' do
    zombie.should be_craving_brains
  end

  it 'should not be hungry after eating brains' do
    zombie.hungry.should be_true
    zombie.eat(:brains)
    zombie.hungry.should be_false
  end
end
```

Let's refactor

```ruby
describe Zombie do
  let(:zombie) { Zombie.create }
  subject { zombie }

  its(:name) { should be_nil? }

  it { should be_craving_brains }

  it 'should not be hungry after eating brains' do
    zombie.hungry.should be_true
    zombie.eat(:brains)
    zombie.hungry.should be_false
  end
end
```

Let's refactor

```ruby
describe Zombie do
  let(:zombie) { Zombie.create }
  subject { zombie }

  its(:name) { should be_nil? }

  it { should be_craving_brains }

  it 'should not be hungry after eating brains' do
    expect { zombie.eat(:brains) }.to change {
     zombie.hungry
    }.from(true).to(false)
  end
end
```

Hooks & Tags

• LEVEL 4 •

Hooks

spec/models/zombie_spec.rb

```ruby
describe Zombie do
  let(:zombie) { Zombie.new }

  it 'is hungry' do
    zombie.hungry!
    zombie.should be_hungry
  end

  it 'craves brains' do
    zombie.hungry!
    zombie.should be_craving_brains
  end
end
```

don't repeat yourself

Hooks

spec/models/zombie_spec.rb

```ruby
describe Zombie do
  let(:zombie) { Zombie.new }
  before { zombie.hungry! }

  it 'is hungry' do
    zombie.should be_hungry
  end

  it 'craves brains' do
    zombie.should be_craving_brains
  end
end
```

run before each example

```ruby
before(:each)
```

run once before all

```ruby
before(:all)
```

run after each

```ruby
after(:each)
```

run after all

```ruby
after(:all)
```

Hooks in context

spec/models/zombie_spec.rb

```
describe Zombie do
  let(:zombie) { Zombie.new }
  before { zombie.hungry! }
  ...
  it 'craves brains' do
    zombie.should be_craving_brains
  end
  context 'with a veggie preference' do
    it 'still craves brains' do
      zombie.hungry!
      zombie.vegetarian = true
      ...
    end
    it 'craves vegan brains' do
      zombie.hungry!
      zombie.vegetarian = true
      ...
    end
  end
```

Hooks in context

spec/models/zombie_spec.rb

```
describe Zombie do
  let(:zombie) { Zombie.new }
  before { zombie.hungry! }
  ...
  it 'craves brains' do
    zombie.should be_craving_brains
  end
  context 'with a veggie preference' do
    it 'still craves brains' do
      zombie.vegetarian = true
      ...
    end
    it 'craves vegan brains' do
      zombie.vegetarian = true
      ...
    end
  end
...
```

available within contexts

Hooks in context

spec/models/zombie_spec.rb

```ruby
describe Zombie do
  let(:zombie) { Zombie.new }
  before { zombie.hungry! }
  ...
  it 'craves brains' do
    zombie.should be_craving_brains
  end
  context 'with a veggie preference' do
    before { zombie.vegetarian = true }
    it 'still craves brains' do
      ...
    end
    it 'craves vegan brains' do
      ...
    end
  end
...
```

available to all examples within this context

Shared examples

spec/models/zombie_spec.rb

```ruby
describe Zombie do
  it 'should not have a pulse' do
    zombie = Zombie.new
    zombie.pulse.should == false
  end
end
```

Duplication!

spec/models/vampire_spec.rb

```ruby
describe Vampire do
  it 'should not have a pulse' do
    vampire = Vampire.new
    vampire.pulse.should == false
  end
end
```


Shared examples

spec/models/zombie_spec.rb

```ruby
describe Zombie do
  it_behaves_like 'the undead'
end
```

used to call shared examples

spec/models/vampire_spec.rb

```ruby
describe Vampire do
  it_behaves_like 'the undead'
end
```

let's build our shared example

spec/support/shared_examples_for_undead.rb

```ruby
shared_examples_for 'the undead' do
  it 'does not have a pulse' do
    subject.pulse.should == false
  end
end
```

refers to the implicit subject

Shared examples

spec/models/zombie_spec.rb

```ruby
describe Zombie do
  it_behaves_like 'the undead' do
    let(:undead) { Zombie.new }
  end
end
```

spec/support/shared_examples_for_undead.rb

```ruby
shared_examples_for 'the undead' do
  it 'does not have a pulse' do
    undead.pulse.should == false
  end
end
```

we can access the 'undead' we defined in 'let'

Shared examples

spec/models/zombie_spec.rb

```ruby
describe Zombie do
  it_behaves_like 'the undead', Zombie.new
end
```

spec/support/shared_examples_for_undead.rb

```ruby
shared_examples_for 'the undead' do |undead|
  it 'does not have a pulse' do
    undead.pulse.should == false
  end
end
```


Metadata and filters

spec/models/zombie_spec.rb

```ruby
describe Zombie do
  context 'when hungry' do
    it 'wants brains'
    context 'with a veggie preference', focus: true do
      it 'still craves brains'
      it 'prefers vegan brains', vegan: true
    end
  end
end
```

run only :focus examples

```
$ rspec --tag focus spec/lib/zombie_spec.rb
Zombie
  with a veggie preference
    still craves brains
    prefers vegan brains
Finished in 0.00125 seconds
2 examples, 0 failures
```

only :focus examples ran

Metadata and filters

spec/models/zombie_spec.rb

```ruby
describe Zombie do
  context 'when hungry' do
    it 'wants brains'
    context 'with a veggie preference', focus: true do
      it 'still craves brains'
      it 'prefers vegan brains', vegan: true
    end
  end
```

spec/spec_helper.rb

```ruby
RSpec.configure do |config|
  config.filter_run focus: true
  config.run_all_when_everything_filtered = true
  ...
end
```

runs everything if none match

Metadata and filters

spec/models/zombie_spec.rb

```ruby
describe Zombie do
  context 'when hungry' do
    it 'wants brains'
    context 'with a veggie preference', focus: true do
      it 'still craves brains'
      it 'prefers vegan brains', vegan: true
    end
  end
end
```

```
$ rspec spec/lib/zombie_spec.rb
Zombie
  with a veggie preference
    still craves brains
    prefers vegan brains
Finished in 0.00125 seconds
2 examples, 0 failures
```

still filtered to :focus

Metadata and filters

spec/models/zombie_spec.rb

```ruby
describe Zombie do
  context 'when hungry' do
    it 'wants brains'
    context 'with a veggie preference', slow: true do
      it 'still craves brains'
      it 'prefers vegan brains'
    end
  end
end
```

skip slow examples

```
$ rspec --tag ~slow spec/lib/zombie_spec.rb
Zombie
  wants brains

Finished in 0.00125 seconds
1 examples, 0 failures
```

slow examples didn't run

Metadata and filters

spec/models/zombie_spec.rb

```ruby
describe Zombie do
  context 'when hungry' do
    it 'wants brains'
    context 'with a veggie preference', slow: true do
      it 'still craves brains'
      it 'prefers vegan brains'
    end
  end
```

spec/spec_helper.rb

skip slow examples in default runs

```ruby
RSpec.configure do |config|
  config.filter_run_excluding slow: true
  config.run_all_when_everything_filtered = true
  ...
end
```

Metadata and filters

spec/models/zombie_spec.rb

```ruby
describe Zombie do
  context 'when hungry' do
    it 'wants brains'
    context 'with a veggie preference', slow: true do
      it 'still craves brains'
      it 'prefers vegan brains'
    end
  end
end
```

```
$ rspec --tag slow spec/lib/zombie_spec.rb
Zombie
  with a veggie preference
    still craves brains
    prefers vegan brains
Finished in 0.00125 seconds
2 examples, 0 failures
```

still filtered to :focus

Mocking & Stubbing

• LEVEL 5 •

Why stub is needed

zombie

decapitate

weapon

```ruby
def slice(args*)
  # complex stuff
end
```

we need to fake this call

/app/models/zombie.rb

```ruby
class Zombie < ActiveRecord::Base
  has_one :weapon

  def decapitate
    weapon.slice(self, :head)
    self.status = "dead again"
  end
end
```


Stubs & Mocks

Stub

For replacing a method with code that returns a specified result.

Mock

A stub with an expectations that the method gets called.

Stubbing

zombie.weapon.stub(:slice)

/app/models/zombie.rb

```
class Zombie < ActiveRecord::Base
  has_one :weapon

  def decapitate
    weapon.slice(self, :head)
    self.status = "dead again"
  end
end
```


Example with stub

/spec/models/zombie_spec.rb

```ruby
describe Zombie do
  let(:zombie) { Zombie.create }

  context "#decapitate" do
    it "sets status to dead again" do
      zombie.weapon.stub(:slice)
      zombie.decapitate
      zombie.status.should == "dead again"
    end
  end
end
```

```ruby
def decapitate
  weapon.slice(self, :head)
  self.status = "dead again"
end
```

we need to test that slice is called

Missing example

/spec/models/zombie_spec.rb

```
describe Zombie do
  let(:zombie) { Zombie.create }

  context "#decapitate" do
    it "calls weapon.slice" do
      zombie.decapitate
    end
    it "sets status to dead again" do
      zombie.weapon.stub(:slice)
      zombie.decapitate
      zombie.status.should == "dead again"
    end
  end
end
```

Mocking

/app/models/zombie.rb

```
zombie.weapon.should_receive(:slice)
```

```
class Zombie < ActiveRecord::Base
  has_one :weapon

  def decapitate
    weapon.slice(self, :head)
    self.status = "dead again"
  end
end
```

stubs the method

+ has an expectation

Complete Spec

/spec/models/zombie_spec.rb

```
describe Zombie do
  let(:zombie) { Zombie.create }

  context "#decapitate" do
    it "calls weapon.slice" do
      zombie.weapon.should_receive(:slice)
      zombie.decapitate
    end
    it "sets status to dead again" do
      zombie.weapon.stub(:slice)
      zombie.decapitate
      zombie.status.should == "dead again"
    end
  end
end
```

Mocking with param

/app/models/zombie.rb

```
class Zombie < ActiveRecord::Base
  def geolocate
    Zoogle.graveyard_locator(self.graveyard)
  end
end
```

/spec/models/zombie_spec.rb

```
it "calls Zoogle.graveyard_locator" do
  Zoogle.should_receive(:graveyard_locator).with(zombie.graveyard)
  zombie.geolocate
end
```

stubs the method + expectation with correct param

Mock and return

/app/models/zombie.rb

```ruby
def geolocate
  loc = Zoogle.graveyard_locator(self.graveyard)
  "#{loc[:latitude]}, #{loc[:longitude]}"
end
```

/spec/models/zombie_spec.rb

```ruby
it "calls Zoogle.graveyard_locator" do
  Zoogle.should_receive(:graveyard_locator).with(zombie.graveyard)
    .and_return({latitude: 2, longitude: 3})
  zombie.geolocate
end
```

stubs the method + expectation with correct param + return value

Stub and return

/app/models/zombie.rb

```ruby
def geolocate
  loc = Zoogle.graveyard_locator(self.graveyard)
  "#{loc[:latitude]}, #{loc[:longitude]}"
end
```

/spec/models/zombie_spec.rb

```ruby
it "returns properly formatted lat, long" do
  Zoogle.stub(:graveyard_locator).with(zombie.graveyard)
      .and_return({latitude: 2, longitude: 3})
  zombie.geolocate.should == "2, 3"
end
```

Stub object

/app/models/zombie.rb

```
def geolocate_with_object
  loc = Zoogle.graveyard_locator(self.graveyard)
  "#{loc.latitude}, #{loc.longitude}"
end
```

should return

object

```
def latitude
  return 2
end
def longitude
  return 3
end
```

/spec/models/zombie_spec.rb

```
it "returns properly formatted lat, long" do
  loc = stub(latitude: 2, longitude: 3)
  Zoogle.stub(:graveyard_locator).returns(loc)
  zombie.geolocate_with_object.should == "2, 3"
end
```

Stubs in the wild

app/mailers/zombie_mailer.rb

```ruby
class ZombieMailer < ActionMailer::Base
  def welcome(zombie)
    mail(from: 'admin@codeschool.com', to: zombie.email,
         subject: 'Welcome Zombie!')
  end
end
```

Not good, calling ActiveRecord

spec/mailers/zombie_mailer_spec.rb

```ruby
describe ZombieMailer do
  context '#welcome' do
    let(:zombie) { Zombie.create(email: 'ash@zombiemail.com') }
    subject { ZombieMailer.welcome(zombie) }

    its(:from) { should include('admin@codeschool.com') }
    its(:to) { should include(zombie.email) }
    its(:subject) { should == 'Welcome Zombie!' }
  end
end
```


Stubs in the wild

```
let(:zombie) { Zombie.create(email: 'ash@zombiemail.com') }
```

Lets create a fake object

```
let(:zombie) { stub(email: 'ash@zombiemail.com') }
```


Stubs in the wild

app/mailers/zombie_mailer.rb

```
class ZombieMailer < ActionMailer::Base
  def welcome(zombie)
    mail(from: 'admin@codeschool.com', to: zombie.email,
         subject: 'Welcome Zombie!')
  end
end
```

spec/mailers/zombie_mailer_spec.rb

```
describe ZombieMailer do
  context '#welcome' do
    let(:zombie) { stub(email: 'ash@zombiemail.com') }
    subject { ZombieMailer.welcome(zombie) }

    its(:from) { should include('admin@codeschool.com') }
    its(:to) { should include(zombie.email) }
    its(:subject) { should == 'Welcome Zombie!' }
  end
end
```

More stub options

```
target.should_receive(:function).once
                                .twice
                                .exactly(3).times
                                .at_least(2).times
                                .at_most(3).times
                                .any_number_of_times
```

```
target.should_receive(:function).with(no_args())
                                .with(any_args())
                                .with("B", anything())
                                .with(3, kind_of(Numeric))
                                .with(/zombie ash/)
```


Custom Matchers

• LEVEL 6 •

Refactoring

app/models/zombie.rb

```ruby
class Zombie < ActiveRecord::Base
  validates :name, presence: true
end
```

we might do this a lot!

spec/models/zombie_spec.rb

```ruby
describe Zombie do
  it 'validates presence of name' do
    zombie = Zombie.new(name: nil)
    zombie.valid?
    zombie.errors.should have_key(:name)
  end
end
```

like 'include', but for hashes

LEVEL 6 – CUSTOM MATCHERS

Refactoring

app/models/zombie.rb

```ruby
class Zombie < ActiveRecord::Base
  validates :name, presence: true
end
```

spec/models/zombie_spec.rb

```ruby
describe Zombie do
  it 'validates presence of name' do
    zombie = Zombie.new(name: nil)
    zombie.should validate_presence_of_name
  end
end
```

so let's create this custom matcher

Custom matchers

spec/support/validate_presence_of_name.rb

```
module ValidatePresenceOfName
  class Matcher
    def matches?(model)
      model.valid?
      model.errors.has_key?(:name)
    end
  end
  def validate_presence_of_name
    Matcher.new
  end
end

RSpec.configure do |config|
  config.include ValidatePresenceOfName, type: :model
end
```

same as our example

matcher available only to model specs

Custom matchers

spec/models/zombie_spec.rb

```
describe Zombie do
  it 'validates presence of name' do
    zombie = Zombie.new(name: nil)
    zombie.should validate_presence_of_name
  end
end
```

```
$ rspec spec/models/zombie_spec.rb
Zombie
  validates presence of name

Finished in 0.02884 seconds
1 example, 0 failures
```

good, but we can't use this matcher for anything else, yet...

Modular matchers

spec/models/zombie_spec.rb

```
describe Zombie do
  it 'validates presence of name' do
    zombie = Zombie.new(name: nil)
    zombie.should validate_presence_of(:name)
  end
end
```

it doesn't accept a parameter

```
$ rspec spec/models/zombie_spec.rb
Zombie
  validates presence of name

Failures:
 1) Zombie has errors on name
  Failure/Error: zombie.should validate_presence_of(:name)
  NoMethodError:
   undefined method `validate_presence_of' for #<RSpec::Core>
   # ./spec/models/zombie.rb:5:in `block (2 levels) in <top (required)>'

1 example, 1 failure
```

Modular matchers

spec/support/validate_presence_of_name.rb

```
module ValidatePresenceOfName
  class Matcher
    def matches?(model)
      model.valid?
      model.errors.has_key?(:name)
    end
  end

  def validate_presence_of_name
    Matcher.new
  end
end
```

time to make this reusable!

Modular matchers

spec/support/validate_presence_of.rb

```ruby
module ValidatePresenceOf
  class Matcher
    def initialize(attribute)
      @attribute = attribute
    end

    def matches?(model)
      model.valid?
      model.errors.has_key?(@attribute)
    end
  end

  def validate_presence_of(attribute)
    Matcher.new(attribute)
  end
end
```

2) store the attribute

3) use your attribute

1) accept an attribute

now let's see if it works as we expect...

Modular matchers

spec/models/zombie_spec.rb

```
describe Zombie do
  it 'validates presence of name' do
    zombie = Zombie.new(name: nil)
    zombie.should validate_presence_of(:name)
  end
end
```

```
$ rspec spec/models/zombie_spec.rb
Zombie
  validates presence of name

Finished in 0.02884 seconds
1 example, 0 failures
```

it does, we're green!

LEVEL 6 — CUSTOM MATCHERS

Matcher failures

spec/models/zombie_spec.rb

```ruby
describe Zombie do
  it 'validates presence of name' do
    zombie = Zombie.new(name: nil)
    zombie.should validate_presence_of(:name)
  end
end
```

But what happens if the validation fails?

app/models/zombie.rb

```ruby
class Zombie < ActiveRecord::Base
  # validates :name, presence: true
end
```

Matcher failures

spec/models/zombie_spec.rb

```
describe Zombie do
  it 'validates presence of name' do
    zombie = Zombie.new(name: nil)
    zombie.should validate_presence_of(:name)
  end
end
```

```
$ rspec spec/models/zombie_spec.rb
Zombie
  validates presence of name

Failures:

 1) Zombie validates presence of name
  Failure/Error: zombie.should validate_presence_of(:name)
   NoMethodError:  undefined method 'failure_message'

Finished in 0.02884 seconds
1 example, 1 failures
```

asked your matcher for a failure message

we haven't defined one yet!

Matcher failures

spec/support/validate_presence_of.rb

```ruby
module ValidatePresenceOf
  class Matcher
    def matches?(model)
      @model = model
      @model.valid?
      @model.errors.has_key?(@attribute)
    end

    def failure_message
      "#{@model.class} failed to validate :#{@attribute} presence."
    end

    def negative_failure_message
      "#{@model.class} validated :#{@attribute} presence."
    end
  end
end
```

saved for use in messages

Matcher failures

```
$ rspec spec/models/zombie_spec.rb
Zombie
  validates presence of name

Failures:

 1) Zombie validates presence of name
  Failure/Error: zombie.should validate_presence_of(:name)
    Expected Zombie to error on :name presence.

Finished in 0.02884 seconds
1 example, 1 failures
```

message from your matcher

Chained methods

app/models/zombie.rb

```ruby
class Zombie < ActiveRecord::Base
  validates :name, presence: { message: 'been eaten' }
end
```

spec/models/zombie_spec.rb

```ruby
describe Zombie do
  it 'validates presence of name' do
    zombie = Zombie.new(name: nil)
    zombie.should validate_presence_of(:name).
      with_message('been eaten')
  end
end
```

this error message should be returned on failure

Chained methods

spec/support/validate_presence_of.rb

```
class Matcher
  def initialize(attribute)
    @attribute = attribute
    @message = "can't be blank"
  end

  def matches?(model)
    @model = model
    @model.valid?
    errors = @model.errors[@attribute]
    errors.any? { |error| error == @message }
  end

  def with_message(message)
    @message = message
    self
  end
end
```

default failure message

collect errors and find a match

override failure message & return self

Chained methods

spec/models/zombie_spec.rb

```
describe Zombie do
  it 'validates presence of name' do
    zombie = Zombie.new(name: nil)
    zombie.should validate_presence_of(:name).
      with_message('been eaten')
  end
end
```

```
$ rspec spec/models/zombie_spec.rb
Zombie
  validates presence of name

Finished in 0.02884 seconds
1 example, 0 failures
```

Chained methods

```
it { should validate_presence_of(:name).with_message('oh noes') }
it { should ensure_inclusion_of(:age).in_range(18..25) }
it { should have_many(:weapons).dependent(:destroy) }
it { should have_many(:weapons).class_name(OneHandedWeapon) }
```

single line chaining

```
it 'has many Tweets' do
  should have_many(:tweets).
    dependent(:destroy).
    class_name(Tweet)
end
```

multiple line chaining