Nesting examples

spec/models/zombie_spec.rb

```
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!

```
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

```
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

```
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' }
```

LEVEL 3 – DRY SPECS

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

```
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

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

newer syntax

```
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

```ruby
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

```
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

```
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

LEVEL 4 – HOOKS & TAGS

Hooks

spec/models/zombie_spec.rb

```
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

```
before(:each)
```

run once before all

```
before(:all)
```

run after each

```
after(:each)
```

run after all

```
after(:all)
```

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
    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

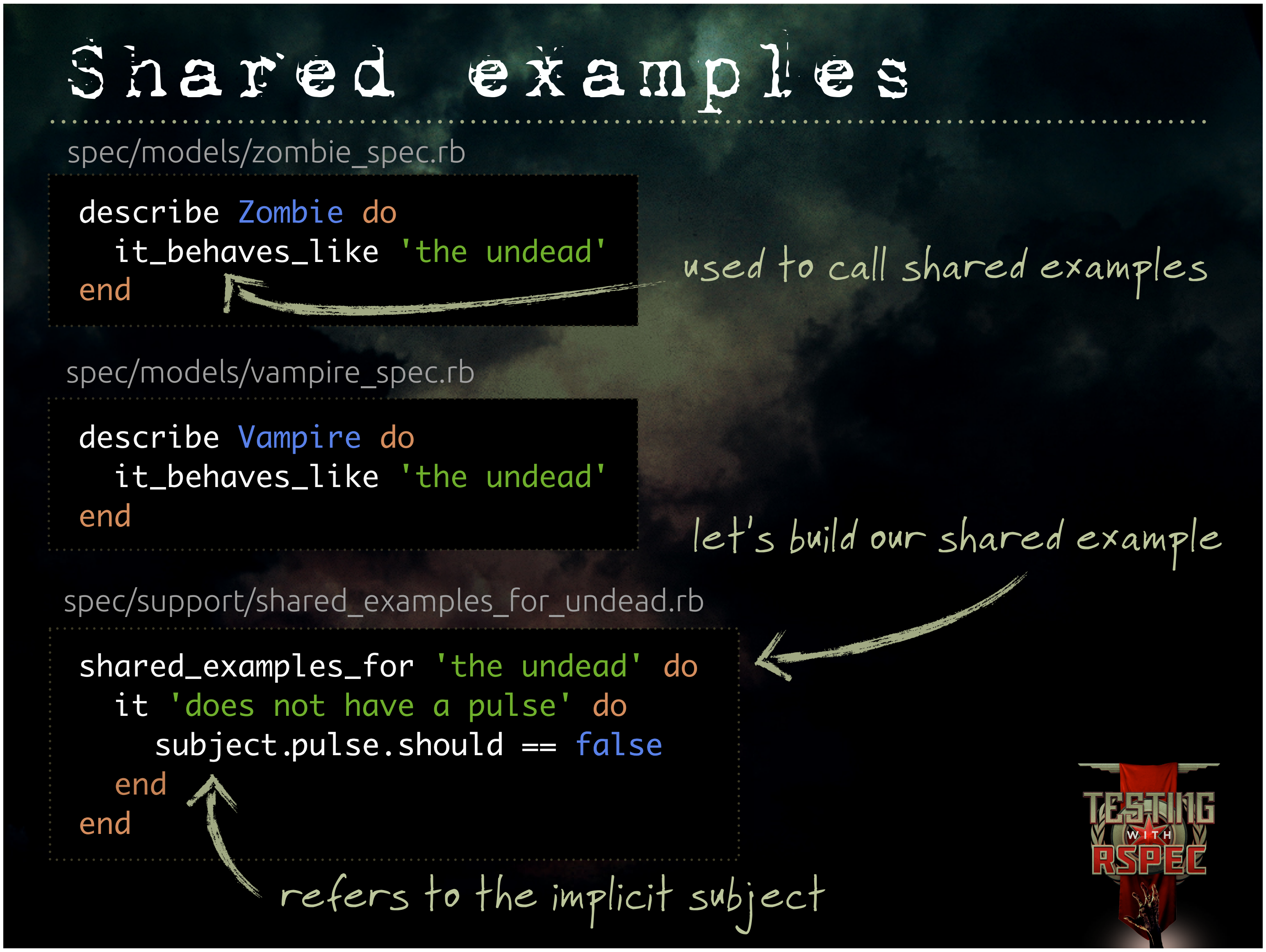
spec/models/zombie_spec.rb

```
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

```
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

TESTING WITH RSPEC

Shared examples

spec/models/zombie_spec.rb

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

spec/support/shared_examples_for_undead.rb

```
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

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

spec/support/shared_examples_for_undead.rb

```
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

```
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
end
```

spec/spec_helper.rb

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

runs everything if none match

Metadata and filters

spec/models/zombie_spec.rb

```
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

```
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
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_with_everything_filtered = true
  ...
end
```

Metadata and filters

spec/models/zombie_spec.rb

```
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
```

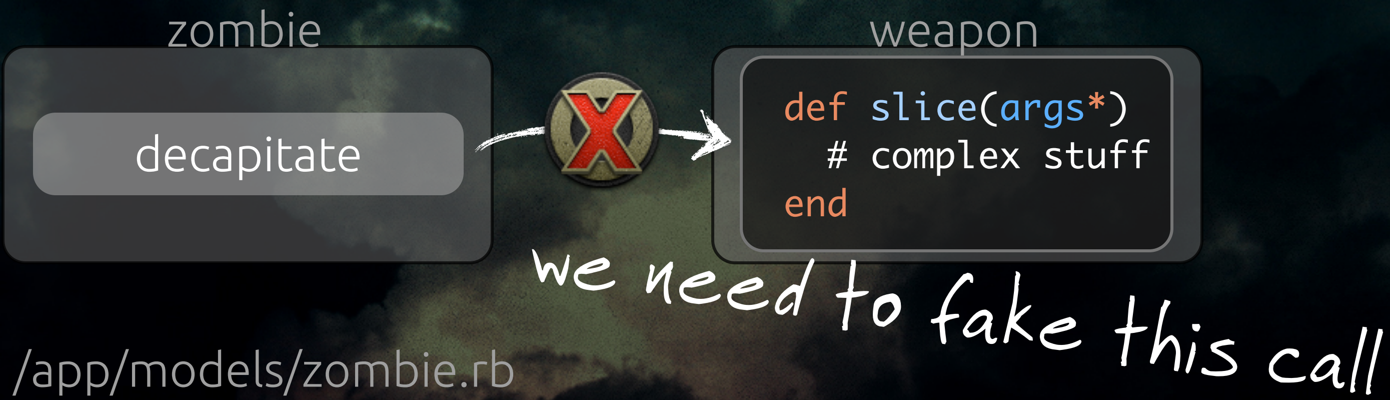
```
$ rspec --tag slow spec/lib/zombie_spec.rb
Zombie
  with a veggie preference
    still craves brains
    prefers vegan brains
Finished in 0.00125 seconds
2 examples, 0 failures
```

still filtered to :focus

Mocking & Stubbing

• LEVEL 5 •

Why stub is needed



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

/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

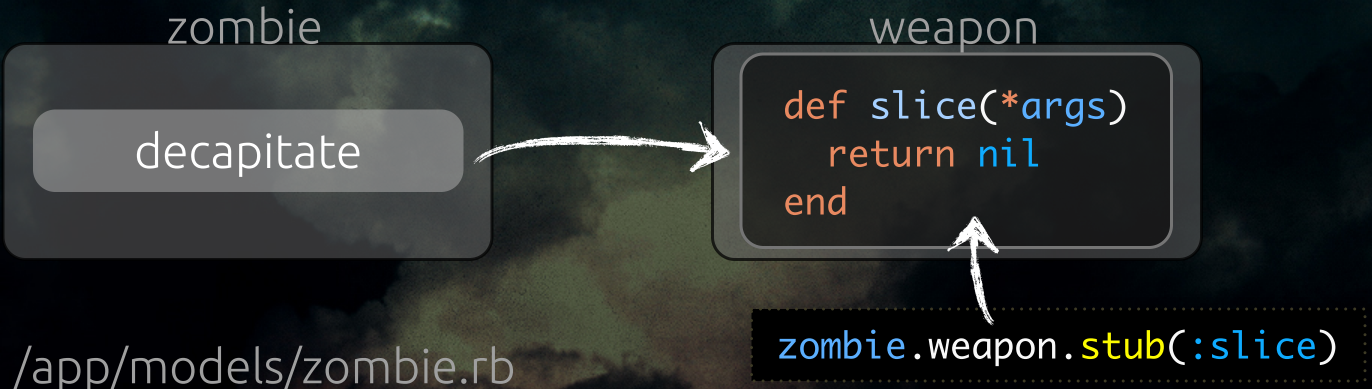
Stub

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

Mock

A stub with an expectations that the method gets called.

Stubbing



```
def slice(*args)
  return nil
end
```

```
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

```
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
```

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

we need to test that slice is called

LEVEL 5 - MOCKING & STUBBING

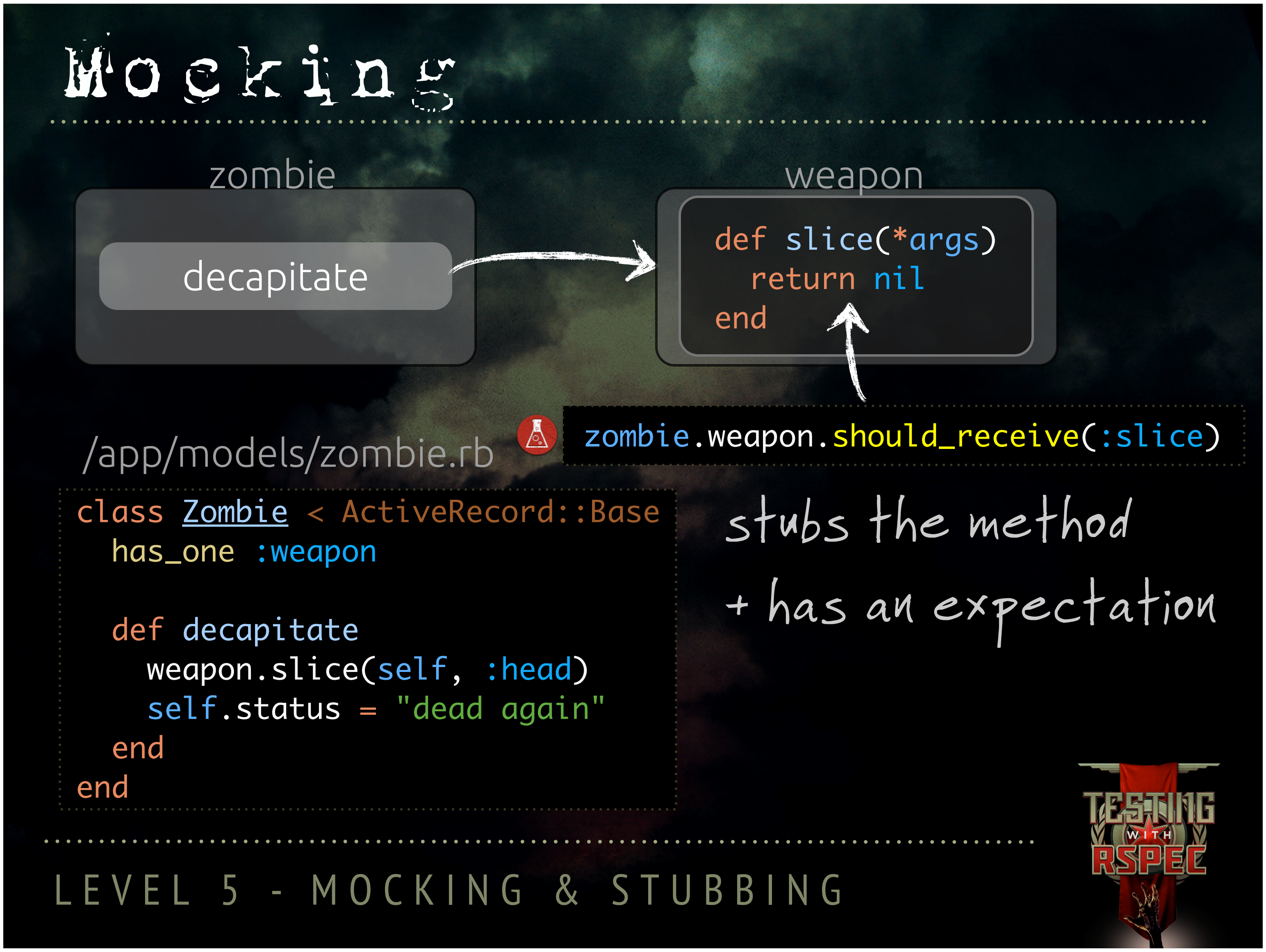
Missing example

/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

zombie

decapitate

weapon

```
def slice(*args)
  return nil
end
```

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

/app/models/zombie.rb

```
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

LEVEL 5 - MOCKING & STUBBING

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
```