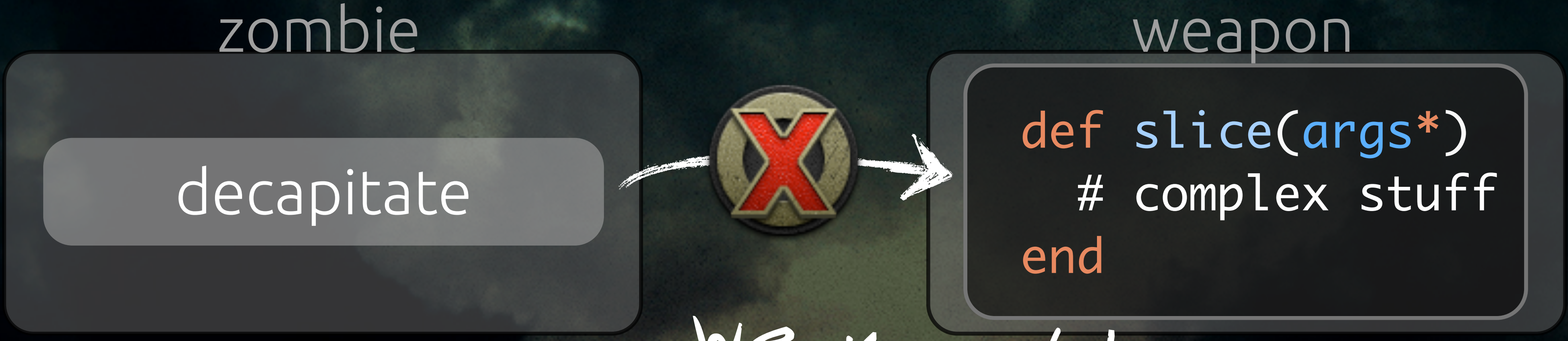
Testing with RSpec

Mocking & Stubbing

• LEVEL 5 •

Why stub is needed



```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

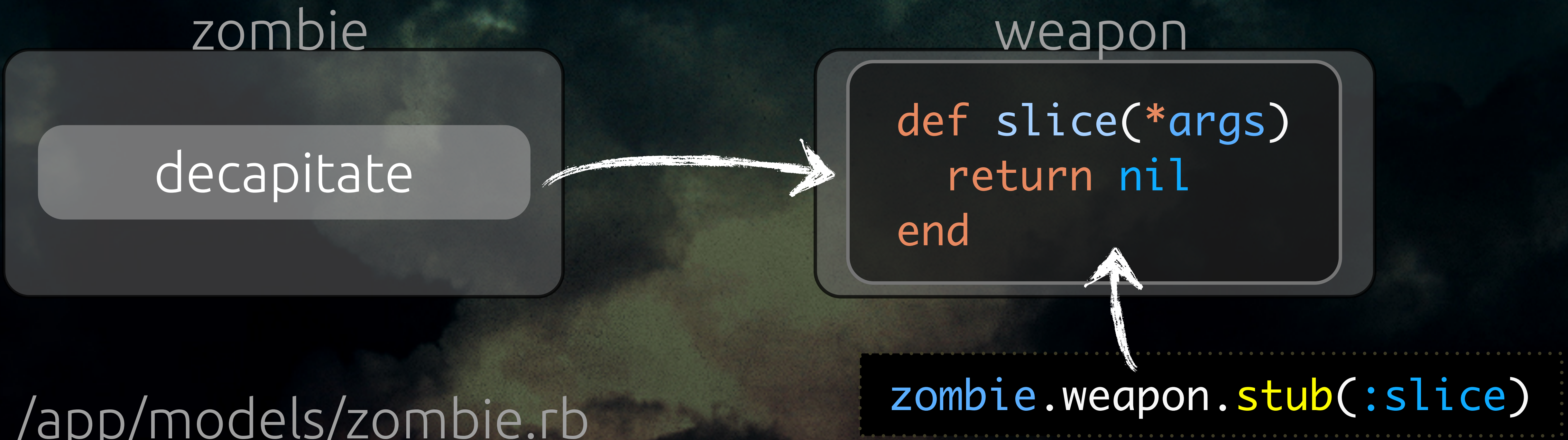
Stub

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

Mock

A stub with an expectations that the method gets called.

Stubbing



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

```ruby
zombie.weapon.stub(:slice)
```

/app/models/zombie.rb

```ruby
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

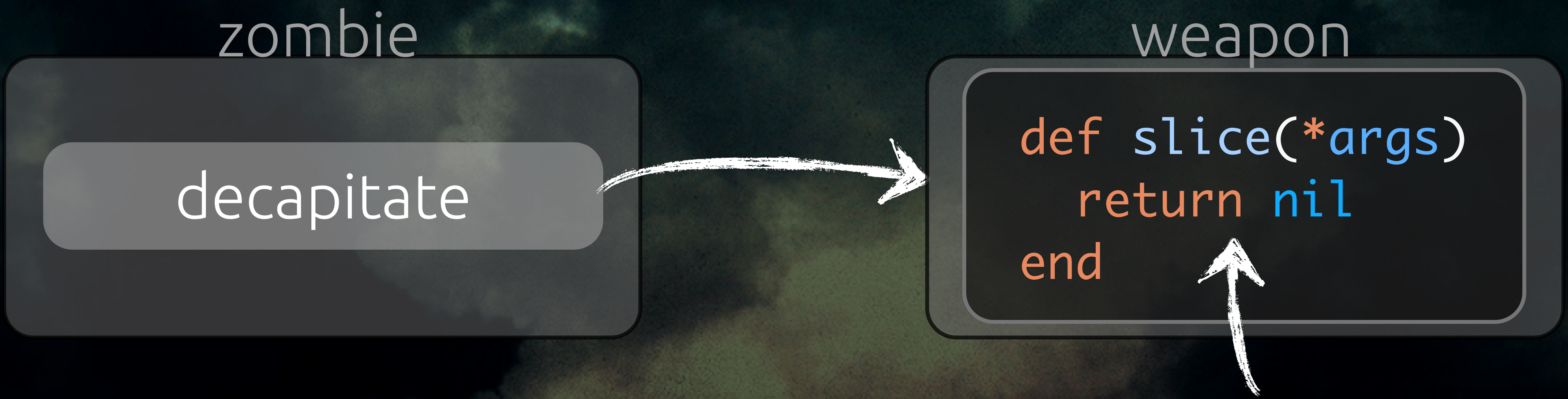
/spec/models/zombie_spec.rb

```
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



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

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

/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 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

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

/spec/models/zombie_spec.rb

```ruby
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

LEVEL 5 - MOCKING & STUBBING

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
```

LEVEL 5 - MOCKING & STUBBING

Stub object

/app/models/zombie.rb

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

should return

object

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

/spec/models/zombie_spec.rb

```ruby
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

```ruby
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

```ruby
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/)
```