

# Deliverable

Upload these 5 files to moodle

1. zombie1.rb (slide 21)
2. zombie\_spec1.rb (slide 21)
3. zombie2.rb (slide 39)
4. zombie\_spec2.rb (slide 39)
5. zombie3.rb (slide 41)
6. Read through “slides at home” [by your next midterm]. Information covered in the slides might show up on your next midterm



These slides are taken from  
[codeschool.com](http://codeschool.com)



# 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

LEVEL 1 - INTRODUCTION



# 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 ← in your project directory  
create spec/spec_helper.rb  
create .rspec
```

more about configuration in level 2

LEVEL 1 - INTRODUCTION



## Using rspec without rails

1. Create a directory in rails\_project directory and call it lab4-1
2. cd lab4-1
3. Create a directory called "lib"
4. Continue with slide
5. \*sudo \* gem install rspec

## 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
```

LEVEL 1 - INTRODUCTION

1. Create “lib” directory inside spec
2. Create file zombie\_spec.rb in spec/lib
3. Type in code from slide



# 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

LEVEL 1 - INTRODUCTION



# 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:
  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 

LEVEL 1 - INTRODUCTION



1. Run rspec on `zombie_spec.rb`
2. Should be “pending”

## Describe Class

spec/lib/zombie\_spec.rb

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

```
$ 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

LEVEL 1 - INTRODUCTION



1. Run rspec on zombie\_spec.rb
2. See it fail

## Creating the class

spec/lib/zombie\_spec.rb

```
require "spec_helper"
require "zombie"

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

lib/zombie.rb

```
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



1. Create zombie.rb in lab4-1/lib
2. Type in the code in the top right corner into zombie.rb
3. Add require 'zombie' into zombie\_spec.rb
4. Run rspec

# Expectations

spec/lib/zombie\_spec.rb

Test::Unit

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

expectation

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

LEVEL 1 - INTRODUCTION



# Test properly fails

spec/lib/zombie\_spec.rb

```
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
```



1. Type code into appropriate file
2. Run rspec
3. See if fail

# Make it pass

spec/lib/zombie\_spec.rb

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

lib/zombie.rb

```
class Zombie
  attr_accessor :name

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

```
$ rspec spec/lib/zombie_spec.rb
 Passed! 
Finished in 0.00047 seconds
1 example, 0 failures
```

LEVEL 1 - INTRODUCTION

1. Add lines into zombie.rb
2. Run rspec
3. See it pass



# Another expectation

spec/lib/zombie\_spec.rb

```
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
```



1. Add another expectation to zombie\_spec.rb
2. Run rspec
3. See it fail
4. Result will say something similar to  
Zombie  
is named Ash  
has no brains (FAILED -1)

# Make it pass

spec/lib/zombie\_spec.rb

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

lib/zombie.rb

```
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 examples, 0 failures
```

Passed! 



LEVEL 1 - INTRODUCTION

1. Add brains to zombie.rb
2. In attr\_accessor
3. In initialize
4. Run rspec
5. See it pass

## Other matchers

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

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

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

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

```
zombie.alive.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
```

LEVEL 1 - INTRODUCTION



- Read through, no need to type it in

# Testing a predicate

/lib/zombie.rb

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

predicate

/spec/lib/zombie\_spec.rb

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

could read better

LEVEL 1 - INTRODUCTION

1. Add code to appropriate files
2. Run rspec
3. See it pass



## Predicate 'be'

```
/spec/lib/zombie_spec.rb
```

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

zombie.hungry?.should be\_true

zombie.should be\_hungry

predicate matcher

LEVEL 1 - INTRODUCTION



1. Change appropriate line to  
zombie.hungry?.should  
be\_true
2. Run rspec
3. See it pass
4. Change appropriate line to zombie.should  
be\_hungry
5. Run rspec
6. See it pass

# Test passes

/spec/lib/zombie\_spec.rb

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

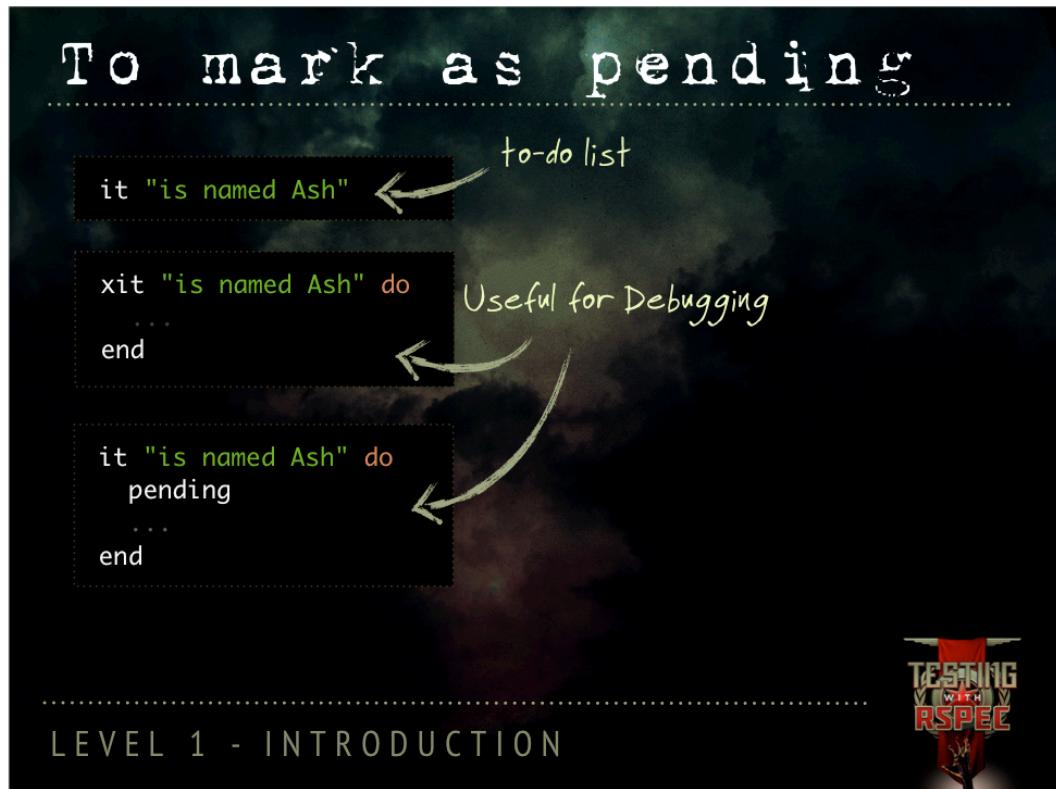
```
$ rspec spec/lib/zombie_spec.rb .
Finished in 0.00045 seconds
1 example, 0 failures
```



LEVEL 1 - INTRODUCTION

1. Run rspec
2. See it pass





1. Create a new expectation for example:
  - `it "is dead"`
1. Run rspec
2. See what happens
3. use `xit` to mark the expectation `it "is named Ash"` as pending
4. Run rspec
5. See what happens
6. Use keyword `pending` to mark `it "has no brains"` as pending
7. Run rspec
8. See what happens
9. Undo 3
10. Undo 6
11. Run rspec
12. Should have 4 examples, 0 failures, 1 pending
13. Submit `zombie.rb` (as `zombie1.rb`) and `zombie_spec.rb` (as `zombie_spec1.rb`) to moodle



# Configuration & Matchers

• LEVEL 2 •

## Using rspec with rails

1. Run “rails new lab4-2” in rails\_project directory
2. cd lab4-2

Move on to the next slide ....nothing on here

# 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

LEVEL 2 – CONFIGURATION & MATCHERS



1. Add appropriate text to Gemfile
2. In group:test do at end of file add:
  - gem 'mocha', :require => false
3. Run bundle install
4. Run rails generate rspec:install

# Configuration

spec/spec\_helper.rb

```
Dir[Rails.root.join("spec/support/**/*.rb")].each { |f| require f}  
...  
    requires all helper files within spec/support  
  
RSpec.configure do |config|  
  config.mock_with :mocha  
  ...  
end  
    allows you to change the default mocking framework
```

LEVEL 2 – CONFIGURATION & MATCHERS



1. Open spec/spec\_helper.rb
2. Uncomment  
config.mock\_with :mocha
3. Add require 'mocha/setup' to  
top of spec\_helper.rb ...after  
the other require(s)
4. RUN
  1. rails generate model  
zombie
  2. rails generate model  
tweet
  3. rails generate model  
weapon
  4. rake db:migrate
  5. rake db:test:load

# 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*

LEVEL 2 – CONFIGURATION & MATCHERS



1. Set color and format for `zombie_spec.rb`
2. Result should say 1 pending since we haven't written any expectation or test in the file
3. For more information on format see :  
<https://www.relishapp.com/rspec/rspec-core/v/2-6/docs/command-line/format-option>

## 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

LEVEL 2 – CONFIGURATION & MATCHERS



- Simply read through, no need to run anything here

## Model spec

spec/models/zombie\_spec.rb

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

app/models/zombie.rb

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

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

*predicate matcher*



1. Add code from slide to appropriate files
2. Run rspec on zombie\_spec.rb
3. See it fail
4. Add attr\_accessor :name to zombie.rb
5. Run rspec on zombie\_spec.rb
6. See it pass

## 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")
    A(zombie.name).should match(/Ash Clone \d/)
  end
end
```

takes a regular expression

- Add code to appropriate file
- Run rspec on zombie\_spec.rb
- See it pass

LEVEL 2 - CONFIGURATION & MATCHERS



## Matchers: include

spec/models/zombie\_spec.rb

```
describe Zombie do
  it 'include tweets' do
    tweet1 = Tweet.new(status: 'Uuuuunhhhhh')
    tweet2 = Tweet.new(status: 'Arrrrrgggg')
    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?

LEVEL 2 – CONFIGURATION & MATCHERS



1. Add code to appropriate file
2. Run rspec on zombie\_spec.rb
3. See it fail
4. go into tweet.rb
5. add
6. attr\_accessor :status
7. belongs\_to :zombie
8. Run rspec on zombie\_spec.rb
9. See it fail
10. Go into zombie.rb
11. Add :tweets after :name in attr\_accessor
12. run rspec on zombie\_spec.rb
13. See if pass

## Matchers: have

spec/models/zombie\_spec.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

```
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

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

LEVEL 2 – CONFIGURATION & MATCHERS



1. Go into weapon.rb and add  
attr\_accessor :name  
belongs\_to :zombie
2. Go into zombie.rb and add  
has\_many :weapons  
def initialize(attributes = nil)  
super(attributes)  
weapons << Weapon.new  
weapons << Weapon.new  
end
3. Add \*first code\* from slide to appropriate file
4. Run rspec on zombie\_spec.rb
5. See it fail (4 examples, 1 failure)
6. Change code to \*second code\* from slide
7. Run rspec on zombie\_spec.rb
8. See it pass

## Matchers: change

spec/models/zombie\_spec.rb

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

you can chain them ↗

.from(1).to(5)

LEVEL 2 – CONFIGURATION & MATCHERS



1. Add code (that uses by) to appropriate file
2. Run rspec on zombie\_spec.rb
3. See it pass (5 examples, 0 failures)
4. Change code to it 'changes the number of Zombies'  
do

```
zombie = Zombie.new(name: "Ash")
expect { zombie.save }.to change { Zombie.count }.from(0).to(1)
end
```
5. Run rspec on zombie\_spec.rb
6. See it pass (5 examples, 0 failures)

## raise\_error

spec/models/zombie\_spec.rb

```
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

LEVEL 2 - CONFIGURATION & MATCHERS



1. Add to appropriate file
2. Run rspec on zombie\_spec.rb
3. See it pass (6 examples, 0 failures)

## More matchers

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

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

```
exist
```

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

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

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

LEVEL 2 – CONFIGURATION & MATCHERS

- Read through, no need to type it into a file



## 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)
```

LEVEL 2 – CONFIGURATION & MATCHERS

- Read through, no need to type into file



# D R Y   S p e c s

• LEVEL 3 •

↑  
Don't Repeat Yourself

## Implicit Subject

spec/models/zombie\_spec.rb

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

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

*only works using describe with a class*

LEVEL 3 – DRY SPECS

1. Type in file
2. Run rspec
3. See it pass (7 examples, 0 failures)
4. Change code to use subject
5. Run rspec
6. See it pass again



# Implicit Receiver

spec/models/zombie\_spec.rb

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



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

LEVEL 3 – DRY SPECS



1. Change code to use
  1. Should  
respond\_to(:name)
2. Run rspec
3. See it pass again (7 examples, 0 failures)

## 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
```

LEVEL 3 - DRY SPECS

1. Change code to use  
it {should respond\_to(:name)}
2. Run rspec
3. See it pass again
4. Submit zombie.rb (as  
zombie2.rb) and zombie\_spec.rb  
(as zombie\_spec2.rb) on moodle

## its

spec/lib/zombie\_spec.rb

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

```
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
```

LEVEL 3 - DRY SPECS



### Example of using its

1. Delete all the expectations from zombie\_spec.rb
2. type in code from slide
3. Change initialize in zombie.rb to:

```
def initialize(attributes = nil)
  super(attributes)
  @name = 'Ash'
end
```

4. The only expectation in zombie\_spec.rb should be:

```
its(:name) { should == 'Ash' }
```

5. Run rspec on zombie\_spec.rb
6. See it pass

1. Change your zombie\_spec.rb to be:

```
require 'spec_helper'
```

```
describe Zombie do
  its(:name) { should == 'Ash' }
  its(:brains) { should be_nil }
  its('tweets.size') { should == 2 }
end
```

2. Make all 3 examples to pass (right now only 1 is passing ... the first one which we worked on in the previous slide)

3. Submit to moodle (as zombie3.rb) your updated zombie.rb that makes all three expectations pass