TESTING WITH RSPEC

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

```
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

```
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
```