
WHAT FACTORIES ARE HIDING FROM YOU
ERIC ROBERTS – @eroberts

WHAT IS A FACTORY?

Test factories are helpers for creating data that can be used in tests.

```ruby
FactoryGirl.define do
  factory :user do
    name "Eric Roberts"
    email "eric@boltmade.com"
  end
end

user = FactoryGirl.create(:user)
#=> #<User id: 1, name: "Eric Roberts", email: "eric@boltmade.com">

user.name
#=> "Eric Roberts"

user.email
#=> "eric@boltmade.com"
```

SO, WHAT'S WRONG WITH FACTORIES?

THEY'RE SLOW!

```
FactoryGirl.define do
  factory :user do
    name "Eric Roberts"
    email "eric@boltmade.com"
  end
end
```

```
FactoryGirl.create(:user)
```

```
INSERT INTO "users" ("name", "email", "created_at", "updated_at")
VALUES (?, ?, ?, ?)  [
    ["name", "Eric"],
    ["email", "eric@boltmade.com"],
    ["created_at", "2015-04-21 01:08:49.409179"],
    ["updated_at", "2015-04-21 01:08:49.409179"]
]
```

```ruby
FactoryGirl.define do
  factory :user do
    association :company

    name "Eric Roberts"
    email "eric@boltmade.com"
  end

  factory :company do
    name "Boltmade"
  end
end
```

```
FactoryGirl.create(:user)
```

```
INSERT INTO "companies" ("name", "created_at", "updated_at")
VALUES (?, ?, ?)  [
    ["name", "Boltmade"],
    ["created_at", "2015-04-21 01:22:24.307976"],
    ["updated_at", "2015-04-21 01:22:24.307976"]
]

INSERT INTO "users" ("name", "email", "company_id", "created_at", "updated_at")
VALUES (?, ?, ?, ?, ?)  [
    ["name", "Eric"],
    ["email", "eric@boltmade.com"],
    ["company_id", 1],
    ["created_at", "2015-04-21 01:22:24.311557"],
    ["updated_at", "2015-04-21 01:22:24.311557"]
]
```

SAVING 0.1 SECONDS PER TEST ON 1200 TESTS SAVES YOU A TOTAL OF 2 MINUTES EVERY TIME THE TESTS RUN.

DON'T YOU KNOW ABOUT `build_stubbed`?

WHAT ELSE IS WRONG WITH THEM?

THEY MAKE BAD DESIGN EASY

WHAT IS THE POINT OF TDD?

FEWER BUGS

SELF-TESTING CODE

REFACTOR WITH CONFIDENCE

BETTER DESIGN

LET'S TAKE A LOOK AT SOME CODE

```ruby
class Estimator < ActiveRecord::Base
  has_many :customers

  def projection
    customers.inject([0,0]) do |(min, max), customer|
      min += customer.revenue * customer.rate.min
      max += customer.revenue * customer.rate.max
      [min, max]
    end
  end
end

class Customer < ActiveRecord::Base
  belongs_to :rate
  belongs_to :estimator

  # revenue method added by ActiveRecord
end

class Rate < ActiveRecord::Base
  def min
    self[:min] / 100.to_f
  end

  def max
    self[:max] / 100.to_f
  end
end
```

```ruby
FactoryGirl.define do
  factory :estimator do
  end

  factory :customer do
    association :rate
    association :estimator
    revenue 100
  end

  factory :rate do
    min 80
    max 90
  end
end
```

```ruby
RSpec.describe Estimator do
  subject { customer.estimator }
  let(:customer) { create :customer }

  describe "#projection" do
    it "should return the sum of the estimated min and max projections" do
      expect(subject.projection).to eq [
        customer.revenue * customer.rate.min,
        customer.revenue * customer.rate.max
      ]
    end
  end
end
```

```
.

Finished in 0.0482 seconds (files took 0.38182 seconds to load)
1 example, 0 failures
```

REQUIREMENT: DON'T USE FACTORIES.

```ruby
RSpec.describe Estimator do
  subject { customer.estimator }
  let(:customer) { create :customer }

  describe "#projection" do
    it "should return the sum of the estimated min and max projections" do
      expect(subject.projection).to eq [
        customer.revenue * customer.rate.min,
        customer.revenue * customer.rate.max
      ]
    end
  end
end
```

```ruby
RSpec.describe Estimator do
  subject { Estimator.new }

  describe "#projection" do
    let(:customer) { double }

    before do
      allow(subject).to receive(:customers).and_return([customer])
    end

    [...]
  end
end
```

```
F

Failures:

  1) Estimator#projection should return the sum of the estimated min and max projections
     Failure/Error: expect(subject.projection).to eq [
       Double received unexpected message :revenue with (no args)
     # ./lib/estimator.rb:8:in `block in projection'
     # ./lib/estimator.rb:7:in `each'
     # ./lib/estimator.rb:7:in `inject'
     # ./lib/estimator.rb:7:in `projection'
     # ./spec/estimator_spec.rb:17:in `block (3 levels) in <top (required)>'

Finished in 0.01076 seconds (files took 0.38914 seconds to load)
1 example, 1 failure
```

```ruby
RSpec.describe Estimator do
  subject { Estimator.new }

  describe "#projection" do
    let(:customer) { double }

    before do
      allow(subject).to receive(:customers).and_return([customer])
      allow(customer).to receive(:revenue).and_return(100)
    end

    [...]
  end
end
```

```ruby
RSpec.describe Estimator do
  subject { Estimator.new }

  describe "#projection" do
    let(:customer) { double }
    let(:rate) { double }

    before do
      allow(subject).to receive(:customers).and_return([customer])
      allow(customer).to receive(:revenue).and_return(100)
      allow(customer).to receive(:rate).and_return(rate)
    end

    [...]
  end
end
```

```ruby
RSpec.describe Estimator do
  subject { Estimator.new }

  describe "#projection" do
    let(:customer) { double }
    let(:rate) { double }

    before do
      allow(subject).to receive(:customers).and_return([customer])
      allow(customer).to receive(:revenue).and_return(100)
      allow(customer).to receive(:rate).and_return(rate)
      allow(rate).to receive(:min).and_return(80)
      allow(rate).to receive(:max).and_return(90)
    end

    [...]
  end
end
```

```ruby
RSpec.describe Estimator do
  subject { Estimator.new }

  describe "#projection" do
    let(:customer) { double }
    let(:rate) { double }

    before do
      allow(subject).to receive(:customers).and_return([customer])
      allow(customer).to receive(:revenue).and_return(100)
      allow(customer).to receive(:rate).and_return(rate)
      allow(rate).to receive(:min).and_return(80)
      allow(rate).to receive(:max).and_return(90)
    end

    it "should return the sum of the estimated min and max projections" do
      expect(subject.projection).to eq [
        customer.revenue * customer.rate.min,
        customer.revenue * customer.rate.max
      ]
    end
  end
end
```

```ruby
class Estimator < ActiveRecord::Base
  has_many :customers

  def projection
    customers.inject([0,0]) do |(min, max), customer|
      min += customer.revenue * customer.rate.min
      max += customer.revenue * customer.rate.max
      [min, max]
    end
  end
end
```

```
.

Finished in 0.0171 seconds (files took 0.40143 seconds to load)
1 example, 0 failures
```

THAT'S 3x FASTER!

TESTING WITH ALL THESE STUBS IS PAINFUL

THAT'S GOOD.

To paraphrase Justin Searls in The Failures of "Intro to TDD":

Your tools and practices should encourage you to do the right thing at each step in your workflow.

```ruby
RSpec.describe Estimator do
  subject { Estimator.new }

  describe "#projection" do
    let(:customer) { double }
    let(:rate) { double }

    before do
      allow(subject).to receive(:customers).and_return([customer])
      allow(customer).to receive(:revenue).and_return(100)
      allow(customer).to receive(:rate).and_return(rate)
      allow(rate).to receive(:min).and_return(80)
      allow(rate).to receive(:max).and_return(90)
    end

    it "should return the sum of the estimated min and max projections" do
      expect(subject.projection).to eq [
        customer.revenue * customer.rate.min,
        customer.revenue * customer.rate.max
      ]
    end
  end
end
```

```ruby
class Estimator < ActiveRecord::Base
  has_many :customers

  def projection
    customers.inject([0,0]) do |(min, max), customer|
      min += customer.revenue * customer.rate.min
      max += customer.revenue * customer.rate.max
      [min, max]
    end
  end
end
```

ESTIMATOR KNOWS WAY TOO MUCH ABOUT CUSTOMER

WHAT DO WE WANT FROM CUSTOMER?

#PROJECTION

```ruby
RSpec.describe Customer do
  subject { Customer.new(revenue: 100) }
  let(:rate) { double }
  let(:min) { 80 }
  let(:max) { 90 }

  before do
    allow(subject).to receive(:rate).and_return(rate)
    allow(rate).to receive(:min).and_return(min)
    allow(rate).to receive(:max).and_return(max)
  end

  it "should return the min and max projection" do
    expect(subject.projection).to eq [
      subject.revenue * rate.min,
      subject.revenue * rate.max
    ]
  end
end
```

```ruby
class Customer < ActiveRecord::Base
  belongs_to :rate
  belongs_to :estimator

  def projection
    [
      revenue * rate.min,
      revenue * rate.max
    ]
  end
end
```

LET'S REVISIT THE ESTIMATOR TESTS

```ruby
RSpec.describe Estimator do
  subject { Estimator.new }

  describe "#projection" do
    let(:customer) { double }
    let(:rate) { double }

    before do
      allow(subject).to receive(:customers).and_return([customer])
      allow(customer).to receive(:revenue).and_return(100)
      allow(customer).to receive(:rate).and_return(rate)
      allow(rate).to receive(:min).and_return(80)
      allow(rate).to receive(:max).and_return(90)
    end

    it "should return the sum of the estimated min and max projections" do
      expect(subject.projection).to eq [
        customer.revenue * customer.rate.min,
        customer.revenue * customer.rate.max
      ]
    end
  end
end
```

```ruby
RSpec.describe Estimator do
  subject { Estimator.new }

  describe "#projection" do
    let(:customer) { double }
    let(:rate) { double }
    let(:projection) { [80,90] }

    before do
      allow(subject).to receive(:customers).and_return([customer])
      allow(customer).to receive(:projection).and_return(projection)
    end

    it "should return the sum of the estimated min and max projections" do
      expect(subject.projection).to eq [
        projection.min,
        projection.max
      ]
    end
  end
end
```

```ruby
class Estimator < ActiveRecord::Base
  has_many :customers

  def projection
    customers.inject([0,0]) do |(min, max), customer|
      min += customer.projection.min
      max += customer.projection.min
      [min, max]
    end
  end
end
```

```
..

Finished in 0.01977 seconds (files took 0.39439 seconds to load)
2 examples, 0 failures
```

```ruby
RSpec.describe Customer do
  subject { Customer.new }

  describe "#projection" do
    let(:rate) { double }

    before do
      allow(subject).to receive(:rate).and_return(rate)
      allow(rate).to receive(:min).and_return(min)
      allow(rate).to receive(:max).and_return(max)
    end

    it "should return the min and max projection" do
      expect(subject.projection).to eq [
        customer.revenue * rate.min,
        customer.revenue * rate.max
      ]
    end
  end
end
```

CUSTOMER KNOWS TOO MUCH ABOUT RATE

WHAT DO WE WANT TO DO WITH RATE?

```
[
  rate.min * revenue,
  rate.max * revenue
]
#=> [80, 90]
```

```
rate * revenue
#=> [80, 90]
```

```ruby
RSpec.describe Rate do
  subject { Rate.new(min: min, max: max) }
  let(:min) { 80 }
  let(:max) { 90 }

  describe "#*" do
    let(:multiplier) { 100 }

    it "should return the two possible rates" do
      expect(subject * multiplier).to eq [
        min * multiplier,
        max * multiplier
      ]
    end
  end
end
```

```ruby
class Rate < ActiveRecord::Base
  def min
    self[:min] / 100.to_f
  end

  def max
    self[:max] / 100.to_f
  end

  def * other
    [
      min * other,
      max * other
    ]
  end
end
```

```ruby
RSpec.describe Customer do
  subject { Customer.new(revenue: 100) }
  let(:rate) { double }
  let(:min) { 80 }
  let(:max) { 90 }

  before do
    allow(subject).to receive(:rate).and_return(rate)
    allow(rate).to receive(:min).and_return(min)
    allow(rate).to receive(:max).and_return(max)
  end

  it "should return the min and max projection" do
    expect(subject.projection).to eq [
      subject.revenue * rate.min,
      subject.revenue * rate.max
    ]
  end
end
```

```ruby
RSpec.describe Customer do
  subject { Customer.new(revenue: revenue) }
  let(:revenue) { 100 }

  describe "#projection" do
    let(:rate) { double }

    before do
      allow(subject).to receive(:rate).and_return(rate)
    end

    it "should send the * message to rate" do
      expect(rate).to receive(:*).with(revenue)
      subject.projection
    end
  end
end
```

```ruby
class Customer < ActiveRecord::Base
  belongs_to :rate
  belongs_to :estimator

  def projection
    rate * revenue
  end
end
```

LET'S REVISIT ESTIMATOR

```ruby
class Estimator < ActiveRecord::Base
  has_many :customers

  def projection
    customers.inject([0,0]) do |(min, max), customer|
      min += customer.projection[0]
      max += customer.projection[1]
      [min, max]
    end
  end
end
```

ESTIMATOR KNOWS TOO MUCH ABOUT RATE

WHAT I'D LIKE TO DO...

```ruby
class Estimator < ActiveRecord::Base
  has_many :customers

  def projection
    customers.inject([0,0]) do |sum, customer|
      sum + customer.projection
    end
  end
end
```

```ruby
RSpec.describe MinMax do
  subject { MinMax.new(min, max) }
  let(:min) { 80 }
  let(:max) { 90 }

  describe "#+" do
    let(:other) { [min, max] }

    it "should return a new object that responds to min and max" do
      new_min_max = subject + other
      expect(new_min_max.min).to eq subject.min + other.min
      expect(new_min_max.max).to eq subject.max + other.max
    end
  end
end
```

```ruby
class MinMax
  attr_reader :min, :max

  def initialize(min, max)
    @min, @max = min, max
  end

  def + other
    self.class.new(min + other.min, max + other.max)
  end

  def self.zero
    new(0, 0)
  end
end
```

```
MinMax.new(1, 10)
#=> #<MinMax:0x007ff19604e2e8 @min=1, @max=10>

MinMax.new(2, 10)
#=> #<MinMax:0x007ff193045650 @min=2, @max=10>

MinMax.new(1, 10) + MinMax.new(2, 10)
#=> #<MinMax:0x007ff1950009b8 @min=3, @max=20>
```

```ruby
class Estimator < ActiveRecord::Base
  has_many :customers

  def projection
    customers.inject(MinMax.zero) do |minmax, customer|
      minmax + customer.projection
    end
  end
end
```

```
....

Finished in 0.0188 seconds (files took 0.40418 seconds to load)
4 examples, 0 failures
```

THE FINAL CODE

```ruby
class Estimator < ActiveRecord::Base
  has_many :customers

  def projection
    customers.inject(MinMax.zero) do |minmax, customer|
      minmax + customer.projection
    end
  end
end
```

```ruby
class Estimator < ActiveRecord::Base
  has_many :customers

  def projection
    customers.map(&:projection).inject(MinMax.zero, :+)
  end
end
```

```ruby
class Customer < ActiveRecord::Base
  belongs_to :rate
  belongs_to :estimator

  def projection
    rate * revenue
  end
end
```

```ruby
class Rate < ActiveRecord::Base
  def min
    self[:min] / 100.to_f
  end

  def max
    self[:max] / 100.to_f
  end

  def * other
    [
      min * other,
      max * other
    ]
  end
end
```

```ruby
class MinMax
  attr_reader :min, :max

  def initialize(min, max)
    @min, @max = min, max
  end

  def + other
    new_min = min + other.min
    new_max = max + other.max
    self.class.new(new_min, new_max)
  end

  def self.zero
    new(0, 0)
  end
end
```

BEFORE

```
35.4: flog total
 5.9: flog/method average

20.7: Estimator#projection                    lib/estimator.rb:6
 4.1: Rate#min                              lib/rate.rb:4
```

AFTER

```
46.1: flog total
 3.8: flog/method average

 9.2: MinMax#+                                  lib/min_max.rb:8
 7.5: Estimator#projection                         lib/estimator.rb:7
 4.8: Rate#*                                    lib/rate.rb:12
 4.4: main#none
 4.1: Rate#min                                  lib/rate.rb:4
```

SO REMEMBER...


FACTORIES ARE BAD

USE ONLY ORGANIC, LOCALLY SOURCED DATA INSTEAD

That's all, folks!

ERIC ROBERTS – @eroberts