Hey, I'm @searls!
justin@testdouble.com
Hey, I'm @searls!

justin@testdouble.com
Hey, I'm @サールズ!
justin@testdouble.com
発音が難しいので、日本語でジュースと申します。

justin@testdouble.com
発音が難しいので、日本語でジュースと申します。リンゴ
justin@testdouble.com
I come from @testdouble. We are software consultants. Say hello@testdouble.com.
About me
1. I talk **fast** when I'm **nervous**
1. I talk fast when I'm nervous 🐰
1. I talk fast when I'm nervous!
2. I am always nervous 🐰
1. I talk **fast** when I'm **nervous** 🐰
2. I am **always** nervous 😳
皆さんへ、ごめんない
皆さんへ、ごめんない頑張って下さい！
I was very nervous about screen size
Wait!
The secret to Ruby 3×3!
is a massively successful language!
Early success
Early success 🎉
Early success
Early success
Early success: Making it easy to make new things
Early success:
Making it easy to make new things 😄
Later success
Later success
Later success
Later success
Later success: Making it easy to maintain old things
Later success:
Making it easy to maintain old things 😕
Today's Question:
Can we make it easier to maintain old Ruby?
Today's Question:
Can we make it easier to maintain old Ruby?
Today, let's refactor some legacy code
Today, let's refactor some legacy code
Refactor - verb
Refactor - *verb*
To change the design of code without changing its observable behavior.
Refactor - verb

To change in advance of a new feature or bug fix, making the job easier.
Today, let's refactor some legacy code.
Today, let's refactor some legacy code.
Legacy code has many definitions
Legacy Code - noun
Legacy Code - noun

Old code.
Legacy Code - *noun*

Code without tests.
Legacy Code - *noun*

Code that we don't like.
Today, my definition is:
Legacy Code - noun
Code we don't understand well enough to change confidently.
Today, let's refactor some legacy code
Refactoring is hard
Refactoring legacy code is very hard
Easy to accidentally break functionality
Legacy refactors often feel unsafe.
Legacy refactors are hard to sell
Business Priority
Business Priority

Bug Fixes

Testing

New Features

Refactoring

Cost/Risk

No selling needed
Easy to sell

Business Priority

Cost/Risk

Bug Fixes

New Features

Testing

Refactoring
Bug Fixes | New Features
---|---
Testing | Refactoring

Can often sell
Very hard to sell
Refactors are hard
Refactors are hard
Refactors are hard
Refactors are hard
As complexity goes up
As complexity goes up

Greater importance
As complexity goes up, less certain.
As complexity goes up

More costly
Make Refactors Great Again
Make Refactors Great for the 1st Time
Selling refactoring to businesspeople
Selling refactoring to businesspeople
Selling refactoring to businesspeople
1. Scare them!
1. Scare them!
"If we don't refactor, then ___________________________!
______________________________!"
1. Scare them!

"If we don't refactor, then someday we'll need to rewrite everything!"
1. Scare them!

"If we don't refactor, then someday we'll need to rewrite everything!"

Far in the future
1. Scare them!

"If we don't refactor, then your maintenance costs will be much higher!"
1. Scare them!
"If we don't refactor, then your maintenance costs will be much higher!"

Hard to quantify
2. Absorb the cost
2. Absorb the cost

New Feature Activities
2. Absorb the cost

Planning

New Feature Activities
2. Absorb the cost

Planning  Development

New Feature Activities
2. Absorb the cost

Planning  Development  Testing

New Feature Activities
2. Absorb the cost

- Development
- Planning
- Testing
- New Feature Activities
2. Absorb the cost

- Refactoring
- Development
- Planning
- Testing
- New Feature Activities
2. Absorb the cost

Refactoring
Planning
Testing
Development

Requires extreme discipline
2. Absorb the cost

Refactoring
Planning
Testing
Development

Collapses under pressure
3. Take hostages
3. Take hostages

Feature #1
3. Take hostages

<table>
<thead>
<tr>
<th>Feature #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature #2</td>
</tr>
</tbody>
</table>
3. Take hostages

<table>
<thead>
<tr>
<th>Feature #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature #2</td>
</tr>
<tr>
<td>Feature #3</td>
</tr>
</tbody>
</table>
3. Take hostages

<table>
<thead>
<tr>
<th>Feature #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature #2</td>
</tr>
<tr>
<td>Feature #3</td>
</tr>
<tr>
<td>Feature #4</td>
</tr>
</tbody>
</table>
3. Take hostages

<table>
<thead>
<tr>
<th>Feature #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical Debt</td>
</tr>
<tr>
<td>Feature #2</td>
</tr>
<tr>
<td>Feature #3</td>
</tr>
</tbody>
</table>
3. Take hostages

<table>
<thead>
<tr>
<th>Feature #1</th>
<th>Technical Debt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature #2</td>
<td>Technical Debt</td>
</tr>
</tbody>
</table>
3. Take hostages

<table>
<thead>
<tr>
<th>Feature #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical Debt</td>
</tr>
<tr>
<td>Feature #2</td>
</tr>
<tr>
<td>Technical Debt</td>
</tr>
</tbody>
</table>
3. Take hostages

<table>
<thead>
<tr>
<th>Feature #1</th>
<th>Technical Debt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature #2</td>
<td>Technical Debt</td>
</tr>
</tbody>
</table>

Erodes trust in the team
Refactoring is hard to sell
Cost/Risk

Refactoring
Too much pressure!
Too much pressure!
Too much pressure!
Too much pressure!
Refactors are scary!
You should buy my book!
THE FRIGHTENED PROGRAMMER

JUSTIN SEARLS
1. Refactoring Patterns
1. Refactoring Patterns
1. Refactoring Patterns

- Extract method
1. Refactoring Patterns

- Extract method
- Pull up / push down
1. Refactoring Patterns

- Extract method
- Pull up / push down
- Split loop
1. Refactoring Patterns

- Extract method
- Pull up / push down
- Split loop

Safer with good tools
1. Refactoring Patterns

- Extract method
- Pull up / push down
- Split loop
1. Refactoring Patterns

- Extract method
- Pull up / push down
- Split loop

Not very expressive
2. Characterization Testing
2. Characterization Testing
2. Characterization Testing
2. Characterization Testing
2. Characterization Testing
2. Characterization Testing
2. Characterization Testing
2. Characterization Testing
2. Characterization Testing
2. Characterization Testing
2. Characterization Testing
2. Characterization Testing
2. Characterization Testing

No wrong answers!
2. Characterization Testing
2. Characterization Testing
2. Characterization Testing
2. Characterization Testing
2. Characterization Testing
2. Characterization Testing
2. Characterization Testing
2. Characterization Testing
2. Characterization Testing
2. Characterization Testing

That's a lot of testing!
2. Characterization Testing
2. Characterization Testing

It's hard to let go of characterization tests
2. Characterization Testing

Tempting to quit halfway through
3. A/B Testing / Experiments
3. A/B Testing / Experiments
3. A/B Testing / Experiments
3. A/B Testing / Experiments

Old code
3. A/B Testing / Experiments

Old code

New code
3. A/B Testing / Experiments

```python
if rand < 0.2
```

Old code

New code
3. A/B Testing / Experiments

if rand < 0.2

Old code

true

New code
3. A/B Testing / Experiments

```
if rand < 0.2
```

false  →  Old code

true  →  New code
3. A/B Testing / Experiments

Scientist!

A Ruby library for carefully refactoring critical paths. **build passing**

How do I science?
3. A/B Testing / Experiments

if rand < 0.2

false

true

Old code

New code
3. A/B Testing / Experiments

Rewriting in big steps is confusing & error-prone
3. A/B Testing / Experiments

```
if rand < 0.2
false
true
```

Old code

New code

Heavy monitoring & analysis required
3. A/B Testing / Experiments

if rand < 0.2

false  New code
true   Old code

Experimenting on humans is risky
Characterization Testing
A/B Experiments
😊 Development
😊 Development
 salarié
 🤕 Testing
Development

Testing

Staging
😊 Development
😢 Testing
🕳️ Staging
🕳️ Production
Development
Testing
Staging
Production
WORKING EFFECTIVELY WITH LEGACY CODE

Michael C. Feathers
Development
Testing
Staging
Development
Testing
Staging
Production
Development

Testing

Staging

Production
"Oh no, I have to give a talk on this"
"Instead of slides, I'll write a gem!"
TDD
TDD
(Talk-Driven Development)
suture
github.com/testdouble/suture
A Ruby gem that helps you refactor your legacy code — Edit

- 212 commits
- 1 branch
- 9 releases
- 2 contributors

Branch: master
New pull request

<table>
<thead>
<tr>
<th>File</th>
<th>Latest commit</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin</td>
<td>initial commit:</td>
<td>21 days ago</td>
</tr>
<tr>
<td>example/rails_app</td>
<td>setup</td>
<td>a day ago</td>
</tr>
<tr>
<td>lib</td>
<td>1.0.0</td>
<td>a day ago</td>
</tr>
<tr>
<td>safe</td>
<td>pretty print</td>
<td>a day ago</td>
</tr>
<tr>
<td></td>
<td>comparator settings well</td>
<td></td>
</tr>
<tr>
<td>test</td>
<td>remove redundant test setup</td>
<td>a day ago</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.codeclimate.yml</td>
<td>codeclimateing</td>
<td>16 days ago</td>
</tr>
<tr>
<td>.gitignore</td>
<td>Going to start locking again</td>
<td>2 days ago</td>
</tr>
<tr>
<td>.projections.json</td>
<td>vim-projectionist settings</td>
<td>19 days ago</td>
</tr>
<tr>
<td>.rubocop.yml</td>
<td>set up codeclimate</td>
<td>16 days ago</td>
</tr>
<tr>
<td>.travis.yml</td>
<td>keep building that chicken</td>
<td>a day ago</td>
</tr>
<tr>
<td>CHANGELOG.md</td>
<td>Changelog for 1.0.0</td>
<td>a day ago</td>
</tr>
</tbody>
</table>
$ gem install suture
Refactors as Surgeries 😷
Refactors as Surgeries 😷
Refactors as Surgeries 😷

Serve a common purpose
Refactors as Surgeries 😷

Serve a common purpose 😊
Refactors as Surgeries 😷

Require careful planning 🗺️
Refactors as Surgeries 😷

Flexible tools
Refactors as Surgeries 😷

Flexible tools

Flexible tools
Refactors as Surgeries 😷

Flexible tools

Flexible tools
Refactors as Surgeries 😷

Follow a process
Refactors as Surgeries 😷

Follow a process
Refactors as Surgeries 😷

Follow a process
Refactors as Surgeries 😷

Multiple Observations

Microscope emoji
Refactors as Surgeries 😷

Multiple Observations 🕵️‍♀️🔍🔬
Refactors as Surgeries 😷

Multiple Observations 🔬🔍🌡
<table>
<thead>
<tr>
<th>Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Plan

Cut
<table>
<thead>
<tr>
<th>Plan</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cut</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Record</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plan</td>
<td>Validate</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>----------</td>
<td></td>
</tr>
<tr>
<td>Cut</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Record</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plan</td>
<td>Validate</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
<td></td>
</tr>
<tr>
<td>Cut</td>
<td>Refactor</td>
<td></td>
</tr>
<tr>
<td>Record</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plan</td>
<td>Validate</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
<td></td>
</tr>
<tr>
<td>Cut</td>
<td>Refactor</td>
<td></td>
</tr>
<tr>
<td>Record</td>
<td>Verify</td>
<td></td>
</tr>
<tr>
<td>Plan</td>
<td>Validate</td>
<td>Compare</td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>Cut</td>
<td>Refactor</td>
<td></td>
</tr>
<tr>
<td>Record</td>
<td>Verify</td>
<td></td>
</tr>
<tr>
<td>Plan</td>
<td>Validate</td>
<td>Compare</td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>Cut</td>
<td>Refactor</td>
<td>Fallback</td>
</tr>
<tr>
<td>Record</td>
<td>Verify</td>
<td>Delete</td>
</tr>
</tbody>
</table>
Plan
Two Bug Fixes:
#1

Calculator service doesn't add negative numbers correctly.
Pure function
class Controller
  def show
    calc = Calculator.new
    @result = calc.add(params[:left], params[:right])
  end
end
class Controller
  def show
    calc = Calculator.new
    @result = calc.add(params[:left], params[:right])
  end
end
class Controller
  def show
    calc = Calculator.new
    @result = calc.add(
      params[:left],
      params[:right]
    )
  end
end
class Controller
  def show
    calc = Calculator.new
    @result = calc.add(params[:left], params[:right])
  end
end
class Calculator
  def add(left, right)
    right.times do
      left += 1
    end
    left
  end
end
class Calculator
  def add(left, right)
    right.times do
      left += 1
    end
    left
  end
end
class Calculator
  def add(left, right)
    right.times do
      left += 1
    end
    left
  end
end
class Calculator
  def add(left, right)
    right.times do
      left += 1
    end
    left
  end
end
class Calculator
  def add(left, right)
    right.times do
      left += 1
    end
    left
  end
end
class Calculator
  def add(left, right)
    right.times do
      left += 1
    end
    left
  end
end
class Calculator
  def add(left, right)
    right.times do
      left += 1
    end
    left
  end
end
class Controller
  def show
    calc = Calculator.new
    @result = calc.add(
      params[:left],
      params[:right]
    )
  end
end
class Controller
  def show
    calc = Calculator.new
    @result = calc.add(params[:left], params[:right])
  end
end

We will create our "seam" here
Calculator tally service doesn't handle odd numbers correctly.
Mutation
class Controller
  def index
    calc = Calculator.new
    params[:nums].each { |n|
      calc.tally(n)
    }
    @result = calc.total
  end
end
class Controller
  def index
    calc = Calculator.new
    params[:nums].each { |n|
      calc.tally(n)
    }
    @result = calc.total
  end
end
class Controller
  def index
    calc = Calculator.new
    params[:nums].each { |n|
      calc.tally(n)
    }
    @result = calc.total
  end
end
class Controller
  def index
    calc = Calculator.new
    params[:nums].each { |n|
      calc.tally(n)
    }
    @result = calc.total
  end
end
class Controller
  def index
    calc = Calculator.new
    params[:nums].each do |n|
      calc.tally(n)
    end
    @result = calc.total
  end
end
class Calculator
  attr_reader :total
  def tally(n)
    @total ||= 0
    n.downto(0) do |i|
      if i * 2 == n
        @total += i * 2
      end
    end
    return
  end
end
class Calculator
  attr_reader :total
  def tally(n)
    @total ||= 0
    n.downto(0) do |i|
      if i * 2 == n
        @total += i * 2
      end
    end
    return
  end
end
class Calculator
  attr_reader :total
  def tally(n)
    @total ||= 0
    n.downto(0) do |i|
      if i * 2 == n
        @total += i * 2
      end
    end
    return
  end
end
class Calculator
  attr_reader :total
  def tally(n)
    @total ||= 0
    n.downto(0) do |i|
      if i * 2 == n
        @total += i * 2
      end
    end
    return
  end
end
class Calculator
  attr_reader :total
  def tally(n)
    @total ||= 0
    n.downto(0) do |i|
      if i * 2 == n
        @total += i * 2
      end
    end
    return
  end
end
class Calculator
  attr_reader :total
  def tally(n)
    @total ||= 0
    n.downto(0) do |i|
      if i * 2 == n
        @total += i * 2
      end
    end
    return
  end
end
class Calculator
  attr_reader :total
  def tally(n)
    @total ||= 0
    n.downto(0) do |i|
      if i * 2 == n
        @total += i * 2
      end
    end
    return
  end
end
class Controller
  def index
    calc = Calculator.new
    params[:nums].each { |n|
      calc.tally(n)
    }
    @result = calc.total
  end
end
class Controller
  def index
    calc = Calculator.new
    params[:nums].each { |n|
      calc.tally(n)
    }
    @result = calc.total
  end
end
class Controller
  def index
    calc = Calculator.new
    params[:nums].each {
      |n|
      calc.tally(n)
    }
    @result = calc.total
  end
end

This seam is more complex
<table>
<thead>
<tr>
<th>Plan</th>
<th>Validate</th>
<th>Compare</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cut</td>
<td>Refactor</td>
<td>Fallback</td>
</tr>
<tr>
<td>Record</td>
<td>Verify</td>
<td>Delete</td>
</tr>
</tbody>
</table>
Cut
Pure function
class Controller
  def show
    calc = Calculator.new
    @result = calc.add(params[:left], params[:right])
  end
end
class Controller

  def show
  
    calc = Calculator.new
    @result = calc.add(
      params[:left],
      params[:right]
    )
  
  end
end
calc = Calculator.new
@result = calc.add(
  params[:left],
  params[:right]
)
calc = Calculator.new
@result = Suture.create :add,
calc = Calculator.new
@result = Suture.create :add,

args: [  
  params[:left],
  params[:right]
]

calc = Calculator.new
@result = Suture.create :add,
old: calc.method(:add),
args: [
  params[:left],
  params[:right]
]
calc = Calculator.new
@result = Suture.create :add,
  old: calc.method(:add),
  args: [
    params[:left],
    params[:right]
  ]

:old must respond_to?(:call)
calc = Calculator.new
@result = Suture.create :add,
         old: calc.method(:add),
         args: [
               params[[:left]],
               params[[:right]]
             ]
calc = Calculator.new
@result = Suture.create :add, old: calc.method(:add), args: [params[:left], params[:right]]

Initially a no-op; verify it still works
Mutation
class Controller
  def index
    calc = Calculator.new
    params[:nums].each { |n|
      calc.tally(n)
    }
    @result = calc.total
  end
end
```ruby
def index
  calc = Calculator.new
  params[:nums].each { |n|
    calc.tally(n)
  }
  @result = calc.total
end
```
calc = Calculator.new

params[:nums].each { |n|
    calc.tally(n)
}

@result = calc.total
calc = Calculator.new
params[:nums].each { |n|
  Suture.create :tally,
}
@result = calc.total
calc = Calculator.new
params[:nums].each { |n|
  Suture.create :tally,
  old: ->(my_calc, m) {
    my_calc.tally(m)
    my_calc.total
  },
  args: [calc, n]
}
@result = calc.total
calc = Calculator.new
params[:nums].each { |n|
    Suture.create :tally,
    old: ->(my_calc, m) {
        my_calc.tally(m)
        my_calc.total
    },
    args: [calc, n]
}

@result = calc.total

Wait, calc isn't an arg!

args: [calc, n]
How to design a seam
Pure functions are easy
Pure functions are easy

`Calculator#add(a,b)`
Pure functions are easy

`Calculator#add(a, b)`

`(2, 8) →`
Pure functions are easy

$\text{Calculator}\#\text{add}(a,b)$

$(2,8) \rightarrow 10$
Pure functions are easy

Calculator#add(a,b)

(2,8) → 10
(2,8)
Pure functions are easy

Calculator\#add(a, b)

(2, 8) \rightarrow 10

(2, 8) \rightarrow 10
Pure functions are easy

`Calculator#add(a,b)`

(2,8) ➔ 10 ➔ (2,8) ➔ 10

✅ Repeatable input & output
Mutation is hard
Mutation is hard

Calculator#tally(n)
Mutation is hard

Calculator#tally(n)

(4) →
Mutation is hard

Calculator\#tally(n)

(4) \rightarrow 4
Mutation is hard

Calculator\#tally(n)

(4) \rightarrow 4

(4) \rightarrow
Mutation is hard

Calculator\#tally(n)

(4) → (4) → 4 → 8
Mutation is hard

Calculator\#tally(n)

(4) → 4
(4) → 8

@total=
Mutation is hard

Calculator\#tally(n)

(4) $\rightarrow$ 4
(4) $\rightarrow$ 8
Mutation is hard

Calculator\#tally(n)

\( (\text{calc@0}, 4) \rightarrow 4 \)

\( (4) \rightarrow 8 \)
Mutation is hard

\[ \text{Calculator}\#tally(n) \]

\[ (\text{calc}@0,4) \rightarrow 4 \]

\[ (\text{calc}@0,4) \rightarrow 8 \]
Mutation is hard

Calculator\#tally(n)

\((\text{calc@0,4}) \rightarrow 4 \rightarrow 4\)
Mutation is hard

Calculator\#tally(n)

(calc₀, 4) → 4

Repeatable input & output
calc = Calculator.new
params[:nums].each { |n|
  Suture.create :tally,
    old: ->(my_calc, m) {
      my_calc.tally(m)
      my_calc.total
    },
    args: [calc, n]
}
@result = calc.total
calc = Calculator.new
params[:nums].each{|n|
  Suture.create :tally,
  old: ->(my_calc, m) {
    my_calc.tally(m)
    my_calc.total
  },
  args: [calc, n]
}
@result = calc.total

Broaden the seam

args: [calc, n]
calc = Calculator.new
params[:nums].each { |n|
    Suture.create :tally,
        old: ->(my_calc, m) {
            my_calc.tally(m)
            my_calc.total
        },
        args: [calc, n]
}

@result = calc.total
calc = Calculator.new
params[:nums].each do |n|
  Suture.create :tally,
    old: ->(my_calc, m) {
      my_calc.tally(m)
    },
    args: [calc, n]

end

@result = calc.total
calc = Calculator.new
params[:nums].each { |n|
  Suture.create :tally,
    old: ->(my_calc, m) {
      my_calc.tally(m)
      my_calc.total
    },
    args: [calc, n]
}
@result = calc.total
calc = Calculator.new

params[:nums].each { |n|
  Suture.create :tally,
    old: ->(my_calc, m) {
      my_calc.tally(m)
      my_calc.total
    },
    args: [calc, n]
}

@result = calc.total

Return a value
calc = Calculator.new
params[:nums].each { |n|
  Suture.create :tally,
    old: ->(my_calc, m) {
      my_calc.tally(m)
      my_calc.total
    },
    args: [calc, n]
  
}
@result = calc.total
Record
Pure function
calc = Calculator.new
@result = Suture.create :add,
  old: calc.method(:add),
  args: [
    params[:left],
    params[:right]
  ]
calc = Calculator.new
@result = Suture.create :add,
old: calc.method(:add),
args: [
  params[:left],
  params[:right]
],
record_calls: true
calc = Calculator.new
@result = Suture.create :add,
    old: calc.method(:add),
    args: [params[:left], params[:right]],
    record_calls: true
calc = Calculator.new
@result = Suture.create :add,
old: calc.method(:add),
args: [
  params[:left],
  params[:right]
],
record_calls: true

Most options support ENV:
SUTURE_RECORD_CALLS=true
Record some calls!
Record via CLI
controller = Controller.new

controller.params = {:left => 5, :right => 6}
controller.show

controller.params = {:left => 3, :right => 2}
controller.show

controller.params = {:left => 1, :right => 89}
controller.show
controller = Controller.new

controller.params = {:left => 5, :right => 6}

controller.show

controller.params = {:left => 3, :right => 2}

controller.show

controller.params = {:left => 1, :right => 89}

controller.show
controller = Controller.new

ccontroller.params = {:left => 5, :right => 6}

ccontroller.show

ccontroller.params = {:left => 3, :right => 2}

ccontroller.show

ccontroller.params = {:left => 1, :right => 89}

ccontroller.show
controller = Controller.new
controller.params = {:left => 5, :right => 6}
controller.show
controller.params = {:left => 3, :right => 2}
controller.show
controller.params = {:left => 1, :right => 89}
controller.show
controller = Controller.new
controller.params = { :left => 5, :right => 6 }
controller.show
controller.params = { :left => 3, :right => 2 }
controller.show
controller = Controller.new

controller.params = {:left => 5, :right => 6}
caller.show

caller.params = {:left => 3, :right => 2}
caller.show

caller.params = {:left => 1, :right => 89}
caller.show
controller = Controller.new

controller.params = {:left => 5, :right => 6}
controller.show

controller.params = {:left => 3, :right => 2}
controller.show

controller.params = {:left => 1, :right => 89}
controller.show
controller = Controller.new

controller.params = {:left => 5, :right => 6}
controller.show

controller.params = {:left => 3, :right => 2}
controller.show

controller.params = {:left => 1, :right => 89}
controller.show
Record via browser
Record via browser

add(4, 5)
Record via browser

add(4, 5)
Record in production!
Record in production!
Record in production!
Mutation
calc = Calculator.new
params[:nums].each { |n|
    Suture.create :tally,
        old: ->(my_calc, m) {
            my_calc.tally(m)
            my_calc.total
        },
        args: [calc, n]
}

@result = calc.total
calc = Calculator.new
params[:nums].each {|n|
    Suture.create :tally,
    old: ->(my_calc, m) {
        my_calc.tally(m)
        my_calc.total
    },
    args: [calc, n],
    record_calls: true
}
$result = calc.total
controller = Controller.new
controller.params = {nums: [2, 4, 6]}
controller.index
controller.params = {nums: [10, 20, 30]}
controller.index
controller.params = {nums: [4, 11]}
controller.index
controller.params = {nums: [1, 3, 5, 7, 9]}
controller.index
controller = Controller.new

ccontroller.params = {nums: [2,4,6]}
ccontroller.index

ccontroller.params = {nums: [10,20,30]}
ccontroller.index

ccontroller.params = {nums: [4,11]}
ccontroller.index

ccontroller.params = {nums: [1,3,5,7,9]}
ccontroller.index
controller = Controller.new

controller.params = {nums: [2,4,6]}
controller.index

controller.params = {nums: [10,20,30]}
controller.index

controller.params = {nums: [4,11]}
controller.index

controller.params = {nums: [1,3,5,7,9]}
controller.index
controller = Controller.new

controller.params = {nums: [2,4,6]}
controller.index

ccontroller.params = {nums: [10,20,30]}
ccontroller.index

controller.params = {nums: [1,3,5,7,9]}
ccontroller.index
controller = Controller.new

ccontroller.params = {nums: [2,4,6]}  
ccontroller.index

controller.params = {nums: [10,20,30]}  
ccontroller.index
controller = Controller.new

controller.params = {nums: [2,4,6]}
controller.index

controller.params = {nums: [10,20,30]}
controller.index

controller.params = {nums: [4,11]}
controller.index
controller = Controller.new

controller.params = {nums: [2,4,6]}
controller.index

controller.params = {nums: [10,20,30]}
controller.index

controller.params = {nums: [4,11]}
controller.index
controller = Controller.new

ccontroller.params = {nums: [2,4,6]}
controller.index

ccontroller.params = {nums: [10,20,30]}
controller.index

ccontroller.params = {nums: [4,11]}
controller.index

ccontroller.params = {nums: [1,3,5,7,9]}
controller.index
controller = Controller.new

ccontroller.params = {nums: [2,4,6]}
ccontroller.index

ccontroller.params = {nums: [10,20,30]}
ccontroller.index

ccontroller.params = {nums: [4,11]}
ccontroller.index

ccontroller.params = {nums: [1,3,5,7,9]}
ccontroller.index
Where does it go?
Suture.config({
})
Suture.config(
{
  database_path:
})
Suture.config({
  database_path: "db/suture.sqlite3"
})
I heard Ruby was getting a database!
Suture.config({
  database_path: "db/suture.sqlite3"
})

Marshal.dump
What about Rails?
Gilded Rose Kata
The Gilded Rose Code Kata

This is a Ruby version of the Gilded Rose Kata, found here.

This is a refactoring kata, so you will be starting with a legacy code base. To work the Kata, clone this git repository checkout the tag 'start-here'. Read the description below for the "rules" involving this kata.

Changes from the original

This Ruby version follows the original code very closely, but has the following changes:
require "suture"

class ItemsController < ApplicationController
  def update_all
    Item.all.each do |item|
      Suture.create :gilded_rose,
                        :old => lambda { |item|
                          item.update_quality!
                          item
                        },
                        :args => [item],
                        :record_calls => true
    end
    redirect_to items_path
  end
end
Items

- New item

ID Name Sell-in Quality Updated at Actions

Update Quality
Items

- New item

ID Name Sell-in Quality Updated at Actions

Update Quality
New item

Name: Normal Item
Sell in: 5
Quality: 10

Create item
New item

Name: Aged Brie
Sell in: 3
Quality: 10

Create item
New item

Name: Sulfuras, Hand of Ragnaros
Sell in: 0
Quality: 80

Create item
# Items

- **New item**

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Sell-in</th>
<th>Quality</th>
<th>Updated at</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Normal Item</td>
<td>5</td>
<td>10</td>
<td>2016-09-08 11:46:23</td>
<td>Delete</td>
</tr>
<tr>
<td>2</td>
<td>Aged Brie</td>
<td>3</td>
<td>10</td>
<td>2016-09-08 11:46:35</td>
<td>Delete</td>
</tr>
<tr>
<td>3</td>
<td>Sulfuras, Hand of Ragnaros</td>
<td>0</td>
<td>80</td>
<td>2016-09-08 11:46:55</td>
<td>Delete</td>
</tr>
</tbody>
</table>

[Update Quality]
## Items

- New item

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Sell-in Quality</th>
<th>Updated at</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Normal Item</td>
<td>5</td>
<td>2016-09-08 11:46:23</td>
<td>Delete</td>
</tr>
<tr>
<td>2</td>
<td>Aged Brie</td>
<td>3</td>
<td>2016-09-08 11:46:35</td>
<td>Delete</td>
</tr>
<tr>
<td>3</td>
<td>Sulfuras, Hand of Ragnaros</td>
<td>80</td>
<td>2016-09-08 11:46:55</td>
<td>Delete</td>
</tr>
</tbody>
</table>

[Update Quality]
# Items

- **New item**

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Sell-in</th>
<th>Quality</th>
<th>Updated at</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Normal Item</td>
<td>4</td>
<td>9</td>
<td>2016-09-08 11:46:56</td>
<td>Delete</td>
</tr>
<tr>
<td>2</td>
<td>Aged Brie</td>
<td>2</td>
<td>11</td>
<td>2016-09-08 11:46:56</td>
<td>Delete</td>
</tr>
<tr>
<td>3</td>
<td>Sulfuras, Hand of Ragnaros</td>
<td>0</td>
<td>80</td>
<td>2016-09-08 11:46:55</td>
<td>Delete</td>
</tr>
</tbody>
</table>

[Update Quality]
<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>args</th>
<th>result</th>
<th>error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>gilded_rose</td>
<td>Binary Data; 535 bytes</td>
<td>Binary Data; 731 bytes</td>
<td>NULL</td>
</tr>
<tr>
<td>2</td>
<td>gilded_rose</td>
<td>Binary Data; 533 bytes</td>
<td>Binary Data; 729 bytes</td>
<td>NULL</td>
</tr>
<tr>
<td>3</td>
<td>gilded_rose</td>
<td>Binary Data; 550 bytes</td>
<td>Binary Data; 696 bytes</td>
<td>NULL</td>
</tr>
</tbody>
</table>
# Items

- New item

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Sell-in</th>
<th>Quality</th>
<th>Updated at</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Normal Item</td>
<td>4</td>
<td>9</td>
<td>2016-09-08 11:46:56</td>
<td>[Delete]</td>
</tr>
<tr>
<td>2</td>
<td>Aged Brie</td>
<td>2</td>
<td>11</td>
<td>2016-09-08 11:46:56</td>
<td>[Delete]</td>
</tr>
<tr>
<td>3</td>
<td>Sulfuras, Hand of Ragnaros</td>
<td>0</td>
<td>80</td>
<td>2016-09-08 11:46:55</td>
<td>[Delete]</td>
</tr>
</tbody>
</table>

[Update Quality]
# Items

- **New item**

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Sell-in</th>
<th>Quality</th>
<th>Updated at</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Normal Item</td>
<td>3</td>
<td>8</td>
<td>2016-09-08 11:48:22</td>
<td>Delete</td>
</tr>
<tr>
<td>2</td>
<td>Aged Brie</td>
<td>1</td>
<td>12</td>
<td>2016-09-08 11:48:22</td>
<td>Delete</td>
</tr>
<tr>
<td>3</td>
<td>Sulfuras, Hand of Ragnaros</td>
<td>0</td>
<td>80</td>
<td>2016-09-08 11:46:55</td>
<td>Delete</td>
</tr>
</tbody>
</table>

[Update Quality]
<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>args</th>
<th>result</th>
<th>error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>gilded_rose</td>
<td>Binary Data; 535 bytes</td>
<td>Binary Data; 731 bytes</td>
<td>NULL</td>
</tr>
<tr>
<td>2</td>
<td>gilded_rose</td>
<td>Binary Data; 533 bytes</td>
<td>Binary Data; 729 bytes</td>
<td>NULL</td>
</tr>
<tr>
<td>3</td>
<td>gilded_rose</td>
<td>Binary Data; 550 bytes</td>
<td>Binary Data; 696 bytes</td>
<td>NULL</td>
</tr>
<tr>
<td>4</td>
<td>gilded_rose</td>
<td>Binary Data; 535 bytes</td>
<td>Binary Data; 731 bytes</td>
<td>NULL</td>
</tr>
<tr>
<td>5</td>
<td>gilded_rose</td>
<td>Binary Data; 533 bytes</td>
<td>Binary Data; 729 bytes</td>
<td>NULL</td>
</tr>
<tr>
<td>6</td>
<td>gilded_rose</td>
<td>Binary Data; 535 bytes</td>
<td>Binary Data; 731 bytes</td>
<td>NULL</td>
</tr>
<tr>
<td>7</td>
<td>gilded_rose</td>
<td>Binary Data; 533 bytes</td>
<td>Binary Data; 729 bytes</td>
<td>NULL</td>
</tr>
</tbody>
</table>
## Items

- New item

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Sell-in</th>
<th>Quality</th>
<th>Updated at</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Normal Item</td>
<td>3</td>
<td>8</td>
<td>2016-09-08 11:48:22</td>
<td>Delete</td>
</tr>
<tr>
<td>2</td>
<td>Aged Brie</td>
<td>1</td>
<td>12</td>
<td>2016-09-08 11:48:22</td>
<td>Delete</td>
</tr>
<tr>
<td>3</td>
<td>Sulfuras, Hand of Ragnaros 0</td>
<td>80</td>
<td></td>
<td>2016-09-08 11:46:55</td>
<td>Delete</td>
</tr>
</tbody>
</table>

[Update Quality]
# Items

- [New item](#)

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Sell-in</th>
<th>Quality</th>
<th>Updated at</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Normal Item</td>
<td>2</td>
<td>7</td>
<td>2016-09-08 11:49:43</td>
<td>Delete</td>
</tr>
<tr>
<td>2</td>
<td>Aged Brie</td>
<td>0</td>
<td>13</td>
<td>2016-09-08 11:49:43</td>
<td>Delete</td>
</tr>
<tr>
<td>3</td>
<td>Sulfuras, Hand of Ragnaros</td>
<td>0</td>
<td>80</td>
<td>2016-09-08 11:46:55</td>
<td>Delete</td>
</tr>
</tbody>
</table>

[Update Quality](#)
<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>args</th>
<th>result</th>
<th>error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>gilded_rose</td>
<td>Binary Data; 535 bytes</td>
<td>Binary Data; 731 bytes</td>
<td>NULL</td>
</tr>
<tr>
<td>2</td>
<td>gilded_rose</td>
<td>Binary Data; 533 bytes</td>
<td>Binary Data; 729 bytes</td>
<td>NULL</td>
</tr>
<tr>
<td>3</td>
<td>gilded_rose</td>
<td>Binary Data; 550 bytes</td>
<td>Binary Data; 696 bytes</td>
<td>NULL</td>
</tr>
<tr>
<td>4</td>
<td>gilded_rose</td>
<td>Binary Data; 535 bytes</td>
<td>Binary Data; 731 bytes</td>
<td>NULL</td>
</tr>
<tr>
<td>5</td>
<td>gilded_rose</td>
<td>Binary Data; 533 bytes</td>
<td>Binary Data; 729 bytes</td>
<td>NULL</td>
</tr>
<tr>
<td>6</td>
<td>gilded_rose</td>
<td>Binary Data; 535 bytes</td>
<td>Binary Data; 731 bytes</td>
<td>NULL</td>
</tr>
<tr>
<td>7</td>
<td>gilded_rose</td>
<td>Binary Data; 533 bytes</td>
<td>Binary Data; 729 bytes</td>
<td>NULL</td>
</tr>
<tr>
<td>8</td>
<td>gilded_rose</td>
<td>Binary Data; 535 bytes</td>
<td>Binary Data; 731 bytes</td>
<td>NULL</td>
</tr>
<tr>
<td>9</td>
<td>gilded_rose</td>
<td>Binary Data; 533 bytes</td>
<td>Binary Data; 729 bytes</td>
<td>NULL</td>
</tr>
</tbody>
</table>
It apparently works
<table>
<thead>
<tr>
<th>Plan</th>
<th>Validate</th>
<th>Compare</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cut</td>
<td>Refactor</td>
<td>Fallback</td>
</tr>
<tr>
<td>Record</td>
<td>Verify</td>
<td>Delete</td>
</tr>
</tbody>
</table>
Validate
Pure function
def test_validate_old_add
  calc = Calculator.new
  Suture.verify :add,
  subject: calc.method(:add)
end
def test_validate_old_add
  calc = Calculator.new
end
def test_validate_old_add
  calc = Calculator.new
  Suture.verify :add
end
def test_validate_old_add
  calc = Calculator.new
  Suture.verify :add,
              subject: calc.method(:add)
end
def test_validate_old_add
    calc = Calculator.new
    Suture.verify :add,
        subject: calc.method(:add)
end
def test_validate_old_add
  calc = Calculator.new
  Suture.verify :add,
    subject: calc.method(:add)
end

Verifies each recorded args
yield the recorded result
def test_validate_old_add
    calc = Calculator.new
    Suture.verify :add,
        subject: calc.method(:add)
end
def test_validate_old_add
    calc = Calculator.new
    Suture.verify :add,
    subject: calc.method(:add)
end
def test_validate_old_add
    calc = Calculator.new
    Suture.verify :add,
        subject: calc.method(:add)
end
def test_validate_old_add
  calc = Calculator.new
  Suture.verify :add,
              subject: calc.method(:add)
end

Cheap tests!
Mutation
def test_old_tally

Suture.verify:tally,

subject: ->(calc, n){
  calc.tally(n)
}.total

end
def test_old_tally
    Suture.verify:tally
end
def test_old_tally
  Suture.verify:tally,
  subject: ->(calc, n) {

  }
end
def test_old_tally
    Suture.verify:tally,
    subject: ->(calc, n){
        calc.tally(n)
    }
end
def test_old_tally
  Suture.verify:tally,
  subject: ->(calc, n) {
    calc.tally(n)
    calc.total
  }
end
def test_old_tally
Suture.verify:tally,
subject: ->(calc, n){
calc.tally(n)
calc.total
}
end

Duplicate the lambda exactly
Finally, a good use for code coverage!
Gilded Rose Kata
Trial # 1
Characterization tests
require 'rspec/given'
require 'gilded_rose'

describe '#update_quality' do

  context "with a single" do
    Given(:initial_sell_in) { 5 }
    Given(:initial_quality) { 10 }
    Given(:item) { Item.new(name, initial_sell_in, initial_quality) }

    When { update_quality([item]) }
  end

  context "normal item" do
    Given(:name) { "NORMAL ITEM" }

    Invariant { item.sell_in.should == initial_sell_in-1 }

    context "before sell date" do
      Then { item.quality.should == initial_quality-1 }
    end

    context "on sell date" do
      Given(:initial_sell_in) { 0 }
      Then { item.quality.should == initial_quality-2 }
    end
  end
end
require 'rspec/given'
require 'gilded_rose'

describe "#update_quality" do

context "with a single" do
  Given(:initial_sell_in) { 5 }
  Given(:initial_quality) { 10 }
  Given(:item) { Item.new(name, initial_sell_in, initial_quality) }

  When { update_quality([item]) }

context "normal item" do
  Given(:name) { "NORMAL ITEM" }

  Invariant { item.sell_in.should == initial_sell_in-1 }

  context "before sell date" do
    Then { item.quality.should == initial_quality-1 }
  end

  context "on sell date" do
    Given(:initial_sell_in) { 0 }
    Then { item.quality.should == initial_quality-2 }
  end

end
Trial # 2
Suture.verify
def test_gilded_rose_old

Suture.verify

subject: ->(items) {
  update_quality(items)
  items
}

fail_fast: true
end

def test_gilded_rose_old

end
def test_gilded_rose_old
  Suture.verify :rose
end
def test_gilded_rose_old
Suture.verify :rose,
subject: ->(items) {
    update_quality(items)
    items
},
}
end
def test_gilded_rose_old
  Suture.verify :rose,
    subject: ->(items) { update_quality(items) },
  fail_fast: true
end
def test_gilded_rose_old
  Suture.verify :rose,
    subject: ->(items) {
      update_quality(items)
      items
    },
  fail_fast: true
end
def test_gilded_rose_old
  Suture.verify :rose,
  subject: ->(items) {
    update_quality(items)
    items
  },

  Items in, mutated
  items out
end
def test_gilded_rose_old
  Suture.verify :rose,
    subject: ->(items) {
      update_quality(items)
      items
    },
    fail_fast: true
end
def test_gilded_rose_old
  Suture.verify :rose,
  subject: ->(items) {
    update_quality(items)
    items
  },
  fail_fast: true
end

All recordings expected to pass
Check coverage before continuing
def self.old_update_quality(items)
    items.each do |item|
        if item.name != 'Aged Brie' && item.name != 'Backstage passes to a TAFKAL80ETC concert'
            if item.quality > 0
                if item.name != 'Sulfuras, Hand of Ragnaros'
                    item.quality -= 1
                end
            end
        else
            if item.quality < 50
                item.quality += 1
                if item.name == 'Backstage passes to a TAFKAL80ETC concert'
                    if item.sell_in < 11
                        if item.quality < 50
                            item.quality += 1
                        end
                    end
                end
            end
            if item.sell_in < 6
                if item.quality < 50
                    item.quality += 1
                end
            end
        end
    end
end
100% Coverage and zero tests
<table>
<thead>
<tr>
<th>Plan</th>
<th>Validate</th>
<th>Compare</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cut</td>
<td>Refactor</td>
<td>Fallback</td>
</tr>
<tr>
<td>Record</td>
<td>Verify</td>
<td>Delete</td>
</tr>
</tbody>
</table>
Refactor
I'm no refactoring expert
I'm no refactoring expert
I'm no refactoring expert
That's why I needed this tool
Unlike my book, this book exists.
Pure function
class Calculator
  def add(left, right)
    right.times do
      left += 1
    end
    left
  end
end
class Calculator
  def add(left, right)
    right.times do
      left += 1
    end
    left
  end
end

Doesn't work for negative values!
class Calculator
  def new_add(left, right)
    return left if right < 0
    # ^ FIXME
    later
    left + right
  end
end
class Calculator
  def new_add(left, right)
    return left if right < 0
    left + right
  end
end
class Calculator
    def new_add(left, right)
        return left if right < 0
        left + right
        end
    end
end
class Calculator
  def new_add(left, right)
    return left if right < 0
    # ^ FIXME later
    left + right
  end
end
class Calculator
  def new_add(left, right)
    return left if right < 0
    # ^ FIXME later
    left + right
  end
end
end
class Calculator
  def new_add(left, right)
    return left if right < 0
    # ^ FIXME later
    left + right
  end
end

Retain current behavior exactly, bugs & all
class Calculator
  def new_add(left, right)
    return left if right < 0
    # ^ FIXME later
    left + right
  end
end

We don't know what else depends on bad behavior
Mutation
class Calculator
  attr_reader :total
  def tally(n)
    @total ||= 0
    n.downto(0) do |i|
      if i * 2 == n
        @total += i * 2
      end
    end
    return
  end
end
class Calculator
  attr_reader :total
  def tally(n)
    @total ||= 0
    n.downto(0) do |i|
      if i * 2 == n
        @total += i * 2
      end
    end
    return
  end
end
Skips odd values!
class Calculator
  def new_tally(n)
    return if n.odd?
    # ^ FIXME later
    @total ||= 0
    @total += n
    return
  end
end
class Calculator
  def new_tally(n)
    return if n.odd?
    # ^ FIXME later
    @total ||= 0
    @total += n
    return
  end
end
class Calculator
  def new_tally(n)
    return if n.odd?
    # ^ FIXME later
    @total ||= 0
    @total += n
    return
  end
end
class Calculator
  def new_tally(n)
    return if n.odd?
    # ^ FIXME later
    @total ||= 0
    @total += n
    return
  end
end
class Calculator
  def new_tally(n)
    return if n.odd?
    # ^ FIXME later
    @total ||= 0
    @total += n
    return
  end
end

Still returns nil
class Calculator
  def new_tally(n)
    return if n.odd?
    # ^ FIXME later
    @total ||= 0
    @total += n
    return
  end
end
class Calculator
  def new_tally(n)
    return if n.odd?
    ^ FIXME later
    @total ||= 0
    @total += n
    return
  end
end
class Calculator
  def new_tally(n)
    return if n.odd?
    # ^ FIXME later
    @total ||= 0
    @total += n
    return
  end
end
class Calculator
  def new_tally(n)
    return if n.odd?
    # ^ FIXME later
    @total ||= 0
    @total += n
    return
  end
end

"Make the change easy, then
make the easy change" - Beck
<table>
<thead>
<tr>
<th>Plan</th>
<th>Validate</th>
<th>Compare</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cut</td>
<td>Refactor</td>
<td>Fallback</td>
</tr>
<tr>
<td>Record</td>
<td>Verify</td>
<td>Delete</td>
</tr>
</tbody>
</table>
Verify
Pure function
def test_new_add
end
def test_new_add
  calc = Calculator.new
end
def test_new_add
    calc = Calculator.new
    Suture.verify :add
end
def test_new_add
  calc = Calculator.new
  Suture.verify :add,
    subject: calc.method(:new_add)
end
def test_new_add
    calc = Calculator.new
    Suture.verify :add,
        subject: calc.method(:new_add)
end
def test_new_add
    calc = Calculator.new
    Suture.verify :add,
        subject: calc.method(:new_add)
end
def test_new_tally

Suture.verify :tally

subject: ->(calc, n)

  calc.new_tally(n)

  calc.total

end

def test_new_tally

end
def test_new_tally
  Suture.verify :tally
end
def test_new_tally
  Suture.verify :tally,
  subject: ->(calc, n){
    calc.new_tally(n)
    calc.total
  }
end
def test_new_tally
  Suture.verify :tally,
  subject: ->(calc, n) {
    calc.new_tally(n)
  }
end
def test_new_tally
  Suture.verify :tally,
  subject: ->(calc, n){
    calc.new_tally(n)
    calc.total
  }
end
def test_new_tally
  Suture::verify(:tally, 
  subject: ->(calc, n) { 
    calc.new_tally(n) 
    calc.total 
  })
end
def test_new_tally
  Suture.verify :tally,
  subject: ->(calc, n)
    calc.new_tally(n)
    calc.total
  }
end
Judge a library by its messages.
Verification of your seam failed!

Descriptions of each unsuccessful verification follows:

## Failures

1. Recorded call for seam `tally` (ID: 13) ran and failed comparison.

   Arguments: ````
   `[<Calculator:@total=nil>, 1]
   ```

   Expected returned value: ````
   `0`
   ```

   Actual returned value: ````
   `nil`
   ```

   Ideas to fix this:
   
   * Focus on this test by setting ENV var `SUTURE_VERIFY_ONLY=13`
1. Recorded call for seam :tally (ID: 13) ran and failed comparison.

   Arguments: ```
       [<Calculator:@total=nil>, 1]
   ```

   Expected returned value: ```
       0
   ```

   Actual returned value: ```
       nil
   ```

Ideas to fix this:
* Focus on this test by setting ENV var `SUTURE_VERIFY_ONLY=13`
# Verification of your seam failed!

Descriptions of each unsuccessful verification follows:

## Failures

1.) Recorded call for seam :tally (ID: 13) ran and failed comparison.

Arguments: 
```
[<Calculator:@total=nil>, 1]
```

Expected returned value: 
```
0
```

Actual returned value: 
```
nil
```

Ideas to fix this:
* Focus on this test by setting ENV var 
`SUTURE_VERIFY_ONLY=13`
* Is the recording wrong? Delete it! `Suture.delete!(13)`
1.) Recorded call for seam :tally (ID: 13) ran and failed comparison.

Arguments: ```
[<Calculator:@total=nil>, 1]
```  

Expected returned value: ```
0
```  

Actual returned value: ```
nil
```
Ideas to fix this:

* Focus on this test by setting ENV var `SUTURE_VERIFY_ONLY=13`

* Is the recording wrong? Delete it! `Suture.delete!(13)`
Ideas to fix this:

* Focus on this test by setting ENV var `SUTURE_VERIFY_ONLY=13`

* Is the recording wrong? Delete it! `Suture.delete!(13)`

Only run this failure
Ideas to fix this:

* Focus on this test by setting `ENV var `SUTURE_VERIFY_ONLY=13``

* Is the recording wrong? Delete it! `Suture.delete!(13)`

Delete bad recordings
Failure advice
### Fixing these failures

#### Custom comparator

If any comparison is failing and you believe the results are equivalent, we suggest you look into creating a custom comparator. See more details here:

https://github.com/testdouble/suture#creating-a-custom-comparator

#### Random seed

Suture runs all verifications in random order by default. If you're seeing an erratic failure, it's possibly due to order-dependent behavior somewhere in your subject's code.

To re-run the tests with the same random seed as was used in this run, set the env var `SUTURE_RANDOM_SEED=74735` or the config entry `:random_seed => 74735`.

To re-run the tests without added shuffling (that is, in the order the calls were recorded in), then set the random seed explicitly to nil with env var `SUTURE_RANDOM_SEED=nil` or the config entry `:random_seed => nil`. 
Fixing these failures

Custom comparator

If any comparison is failing and you believe the results are equivalent, we suggest you look into creating a custom comparator. See more details here:

https://github.com/testdouble/suture#creating-a-custom-comparator
Comparing Results
Default Comparator
or
What about ActiveRecord?!
AR

_attributes

==

AR

_attributes
😊 ≠ 😞
Custom Comparators
What if Calculator had many other fields?

class Calculator
  attr_reader :created_at,
             :memory_val,
             :total

  # ...
end
Suture.verify : tally,
subject: ->(calc, n) {
    calc.new_tally(n)
    calc.total
}

Suture.verify : tally,  
subject: ->(calc, n) {
    calc.new_tally(n)
    calc.total
}

Suture.verify : tally, subject: ->(calc, n) { calc.new_tally(n) calc }
Suture.verify : tally,
subject: ->(calc, n) {
  calc.new_tally(n)
calc
},
comparator: ->(recorded, actual) {
}
Suture.verify : tally, 
subject: ->(calc, n) { 
calc.new_tally(n) 
calc 
},
comparator: ->(recorded, actual) { 
recorded.total == 
actual.total 
}
Suture.verify : tally,
subject: ->(calc, n) {
    calc.new_tally(n)
    calc
},
comparator: ->(recorded, actual) {
    recorded.total == actual.total
}
Classes also exist!
class CalcPare < Suture::Comparator

  def call(left, right)
    if super then return true end
    left.total == right.total
  end

end

Suture.verify:
  tally,
  subject: ->(calc, n) {
    calc.new_tally(n)
    calc.total
  }

comparator: CalcPare.new
class CalcPare < Suture::Comparator
  def call(left, right)
    if super then return true end
    left.total == right.total
  end
end
end
class CalcPare < Suture::Comparator
def call(left, right)
    return true if super
end
end
Suture.verify
:tally,
subject: ->(calc, n) {
calc.new_tally(n)
calc.total
}
comparator: CalcPare.new
class CalcPare < Suture::Comparator
def call(left, right)
  return true if super
  left.total == right.total
end
end
class CalcPare < Suture::Comparator
  def call(left, right)
    return true if super
    left.total == right.total
  end
end

Suture.verify :tally,
  subject: ->(calc, n) {
    calc.new_tally(n)
    calc
  },
  comparator:
class CalcPare < Suture::Comparator
  def call(left, right)
    return true if super
    left.total == right.total
  end
end

Suture.verify :tally,
  subject: ->(calc, n) {
    calc.new_tally(n)
    calc
  },
  comparator: CalcPare.new
Returning to the error message
### Random seed

Suture runs all verifications in random order by default. If you're seeing an erratic failure, it's possibly due to order-dependent behavior somewhere in your subject's code.

To re-run the tests with the same random seed as was used in this run, set the env var `SUTURE_RANDOM_SEED=74735` or the config entry `:random_seed => 74735`.

To re-run the tests without added shuffling (that is, in the order the calls were recorded in), then set the random seed explicitly to nil with env var `SUTURE_RANDOM_SEED=nil` or the config entry `:random_seed => nil`. 
### Random seed

Suture runs all verifications in random order by default. If you're seeing an erratic failure, it's possibly due to order-dependent behavior somewhere in your subject's code.

To re-run the tests with the same random seed as was used in this run, set the env var `SUTURE_RANDOM_SEED=74735` or the config entry `:random_seed => 74735`.

To re-run the tests without added shuffling (that is, in the order the calls were recorded in), then set the random seed explicitly to nil with env var `SUTURE_RANDOM_SEED=nil` or the config entry `:random_seed => nil`.
### Random seed

Suture runs all verifications in random order by default. If you're seeing an erratic failure, it's possibly due to order-dependent behavior somewhere in your subject's code.

To re-run the tests with the same random seed as was used in this run, set the env var `SUTURE RANDOM SEED=74735` or the config entry `:random_seed => 74735`.

To re-run the tests without added shuffling (that is, in the order the calls were recorded in), then set the random seed explicitly to nil with env var `SUTURE RANDOM SEED=nil` or the config entry `:random_seed => nil`.
Discoverable configuration
# Configuration

This is the configuration used by this test run:

```ruby
{
  :comparator => Suture::Comparator.new,
  :database_path => "db/suture.sqlite3",
  :fail_fast => false,
  :call_limit => nil, # (no limit)
  :time_limit => nil, # (no limit)
  :error_message_limit => nil, # (no limit)
  :random_seed => 74735
}
```
# Configuration

This is the configuration used by this test run:

```ruby
{
  :comparator => Suture::Comparator.new,
  :database_path => "db/suture.sqlite3",
  :fail_fast => false,
  :call_limit => nil,  # (no limit)
  :time_limit => nil,  # (no limit)
  :error_message_limit => nil,  # (no limit)
  :random_seed => 74735
}
```
# Configuration

This is the configuration used by this test run:

```ruby
{
  :comparator => Suture::Comparator.new,
  :database_path => "db/suture.sqlite3",
  :fail_fast => false,
  :call_limit => nil, # (no limit)
  :time_limit => nil, # (no limit)
  :error_message_limit => nil, # (no limit)
  :random_seed => 74735
}
```
```
# Configuration

This is the configuration used by this test run:

...  

```
{
  :comparator => Suture::Comparator.new,
  :database_path => "db/suture.sqlite3",
  :fail_fast => false,
  :call_limit => nil, # (no limit)
  :time_limit => nil, # (no limit)
  :error_message_limit => nil, # (no limit)
  :random_seed => 74735
}
```

...
# Configuration

This is the configuration used by this test run:

```ruby
{
  :comparator => Suture::Comparator.new,
  :database_path => "db/suture.sqlite3",
  :fail_fast => false,
  :call_limit => nil, # (no limit)
  :time_limit => nil, # (no limit)
  :error_message_limit => nil, # (no limit)
  :random_seed => 74735
}
```
# Configuration

This is the configuration used by this test run:

```ruby
{
  :comparator => Suture::Comparator.new,
  :database_path => "db/suture.sqlite3",
  :fail_fast => false,
  :call_limit => nil, # (no limit)
  :time_limit => nil, # (no limit)
  :error_message_limit => nil, # (no limit)
  :random_seed => 74735
}
```
This is the configuration used by this test run:

```ruby
{
  :comparator => Suture::Comparator.new,
  :database_path => "db/suture.sqlite3",
  :fail_fast => false,
  :call_limit => nil, # (no limit)
  :time_limit => nil, # (no limit)
  :error_message_limit => nil, # (no limit)
  :random_seed => 74735
}
```
This is the configuration used by this test run:

```ruby
{
  :comparator => Suture::Comparator.new,
  :database_path => "db/suture.sqlite3",
  :fail_fast => false,
  :call_limit => nil, # (no limit)
  :time_limit => nil, # (no limit)
  :error_message_limit => nil, # (no limit)
  :random_seed => 74735
}
```
A sense of progress
# Result Summary

- Passed........12
- Failed........1
  - with error..0
- Skipped.......0
- Total calls...13

## Progress

Here's what your progress to initial completion looks like so far:

[●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●● ●
# Result Summary

- Passed........12
- Failed........1
  - with error..0
- Skipped.......0
- Total calls...13

## Progress

Here's what your progress to initial completion looks like so far:

```
[●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●● ◈◌◌◌◌]
```

0f 13 recorded interactions, 12 are currently passing. That's 92%!
Judge a library by its messages
Wait, why did verification fail?
1.) Recorded call for seam :tally (ID: 13) ran and failed comparison.

Arguments: ```
[<Calculator:@total=nil>, 1]
```

Expected returned value: ```
0
```

Actual returned value: ```
nil
```
1.) Recorded call for seam :tally (ID: 13) ran and failed comparison.

Arguments: ```
[<Calculator:@total=\nil>, 1]```

Expected returned value: ```
0```

Actual returned value: ```
\nil```

1.) Recorded call for seam : tally (ID: 13) ran and failed comparison.

Arguments: ```
[<Calculator:@total=nil>, 1]
```

Expected returned value: ```
0
```

Actual returned value: ```
nil
```
class Calculator
  attr_reader :total
  def new_tally(n)
    return if n.odd?
    # ^ FIXME later
    @total ||= 0
    @total += n
    return
  end
end
class Calculator
  attr_reader :total
  def new_tally(n)
    return if n.odd?
    # ^ FIXME later
    @total ||= 0
    @total += n
    return
  end
end
class Calculator
  attr_reader :total
  def new_tally(n)

    return if n.odd?
    # ^ FIXME later

    @total ||= 0
    @total += n
    return
  end
end
class Calculator
  attr_reader :total
  def new_tally(n)
    @total ||= 0
    return if n.odd?
    # ^ FIXME later
    @total ||= 0
    @total += n
    return
  end
end
def test_new_tally
  Suture.verify :tally,
  subject: ->(calc, n){
    calc.new_tally(n)
    calc.total
  }
end
def test_new_tally
  Suture.verify :tally,
  subject: ->(calc, n) {
    calc.new_tally(n)
    calc.total
  }
end
<table>
<thead>
<tr>
<th>Plan</th>
<th>Validate</th>
<th>Compare</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cut</td>
<td>Refactor</td>
<td>Fallback</td>
</tr>
<tr>
<td>Record</td>
<td>Verify</td>
<td>Delete</td>
</tr>
</tbody>
</table>
Compare
Our mission
Our mission 😃 Development
Our mission

😃 Development

😃 Testing
Our mission

😃 Development
😃 Testing
😃 Staging
Our mission

- Development
- Testing
- Staging
- Production
Our progress
Our progress

✅ Development
Our progress

- Development
- Testing
Our progress

✅ Development

✅ Testing

❓ Staging
Our progress

✅ Development
✅ Testing
❓ Staging
❓ Production
Pure function
class Controller
  def show
    calc = Calculator.new
    @result = Suture.create :add,
    old: calc.method(:add),
    new: calc.method(:new_add),
    args: [
      params[:left],
      params[:right]
    ]
  end
end
class Controller
  def show
    calc = Calculator.new
    @result = Suture.create :add,
      old: calc.method(:add),
      new: calc.method(:new_add),
      args: [
        params[:left],
        params[:right]
      ],
      call_both: true
  end
end
class Controller
  def show
    calc = Calculator.new
    @result = Suture.create :add,
      old: calc.method(:add),
      new: calc.method(:new_add),
      args: [
        params[:left],
        params[:right]
      ],
      call_both: true
  end
end
class Controller
  def show
    calc = Calculator.new
    @result = Suture.create :add,
      old: calc.method(:add),
      new: calc.method(:new_add),
      args: [params[:left], params[:right]],
      call_both: true
  end
end

Calls :new & :old
class Controller
  def show
    calc = Calculator.new
    @result = Suture.create :add,
      old: calc.method(:add),
      new: calc.method(:new_add),
      args: [params[:left],
              params[:right]],
      call_both: true
  end
end

 Calls : new & : old

✅
You will find surprising inputs & outputs 😲
Mutation
calc = Calculator.new
params[:nums].each { |n|
  Suture.create :tally,
    old: ->(my_calc, m) {
      my_calc.tally(m)
      my_calc.total
    },
    new: ->(my_calc, m) {
      my_calc.new_tally(m)
      my_calc.total
    },
    args: [calc, n]
}
@result = calc.total
calc = Calculator.new
params[:nums].each { |n|
  Suture.create :tally,
      old: ->(my_calc, m) {
        my_calc.tally(m)
        my_calc.total
      },
      new: ->(my_calc, m) {
        my_calc.new_tally(m)
        my_calc.total
      },
      args: [calc, n],
      call_both: true
  }
@result = calc.total
calc = Calculator.new
params[:nums].each{|n|
  Suture.create :tally,
  old: ->(my_calc, m) {
    my_calc.tally(m)
    my_calc.total
  },
  new: ->(my_calc, m) {
    my_calc.new_tally(m)
    my_calc.total
  },
  args: [calc, n],
  call_both: true
}
@result = calc.total
calc = Calculator.new
params[:nums].each { |n|
    Suture.create :tally,
        old: ->(my_calc, m) {
            my_calc.tally(m)
            my_calc.total
        },
        new: ->(my_calc, m) {
            my_calc.new_tally(m)
            my_calc.total
        },
        args: [calc, n],
        call_both: true
}
@result = calc.total
Another huge error message
The results from the old & new code paths did not match for the seam (Suture::Error::ResultMismatch):tally and Suture is raising this error because the `:call_both` option is enabled, because both code paths are expected to return the same result.

Arguments: ```
[<Calculator:@total=4>, 2]
```

The new code path returned: ```
2
```

The old code path returned: ```
4
```

Here's what we recommend you do next:

1. Verify that this mismatch does not represent a missed requirement in the new code path. If it does, implement it!

2. If either (or both) code path has a side effect that impacts the return value of the other, consider passing an `:after_old` and/or `:after_new` hook to clean up your application's state well enough to run both paths one-after-the-other safely.

3. If the two return values above are sufficiently similar for the purpose of your application, consider writing your own custom comparator that relaxes the comparison (e.g. only checks equivalence of the attributes that matter). See the README for more info on custom comparators.

4. If the new code path is working as desired (i.e. the old code path had a bug for this argument and you don't want to reimplement it just to make them perfectly in sync with one another), consider writing a one-off comparator for this seam that will ignore the affected range of arguments. See the README for more info on custom comparators.

By default, Suture's :call_both mode will log a warning and raise an error when the results of each code path don't match. It is intended for use in any pre-production environment to "try out" the new code path before pushing it to production. If, for whatever reason, this error is too disruptive and logging is sufficient for monitoring results, you may disable this error by setting `:raise_on_result_mismatch` to false.
Suture is raising this error because the `:call_both` option is enabled, because both code paths are expected to return the same result.

Arguments: ```
[<Calculator:@total=2>, 2]
``` The new code path returned: ```
2
``` The old code path returned: ```
4
```
Suture is raising this error because the `:call_both` option is enabled, because both code paths are expected to return the same result.

Arguments: ```
[<Calculator:@total=2>, 2]
``` The new code path returned: ```
2
``` The old code path returned: ```
4
```
Suture is raising this error because the `:call_both` option is enabled, because both code paths are expected to return the same result.

Arguments: ```
[<Calculator:@total=2>, 2]
``` The new code path returned: ```
2
``` The old code path returned: ```
4
```
Suture is raising this error because the `:call_both` option is enabled, because both code paths are expected to return the same result.

Arguments: ````
    [<Calculator:@total=2>, 2]
````

The new code path returned: ````
    2
````

The old code path returned: ````
    4
````
calc = Calculator.new
params[:nums].each { |n|
    Suture.create :tally,
        old: ->(my_calc, m) {
            my_calc.tally(m)
            my_calc.total
        },
        new: ->(my_calc, m) {
            my_calc.new_tally(m)
            my_calc.total
        },
        args: [calc, n],
        call_both: true
}

@result = calc.total
calc = Calculator.new
params[:nums].each { |n|
    Suture.create :tally,
    old: ->(my_calc, m) {
        my_calc.tally(m)
        my_calc.total
    },
    new: ->(my_calc, m) {
        my_calc.new_tally(m)
        my_calc.total
    },
    args: [calc, n],
    call_both: true,
    dup_args: true
}
@result = calc.total
calc = Calculator.new
params[:nums].each { |n|
    Suture.create :tally,
        old: ->(my_calc, m) {
            my_calc.tally(m)
            my_calc.total
        },
        new: ->(my_calc, m) {
            my_calc.new_tally(m)
            my_calc.total
        },
        args: [calc, n],
        call_both: true,
        dup_args: true
}
@result = calc.total
calc = Calculator.new
params[:nums].each { |n|
  Suture.create :tally,
  old: ->(my_calc, m) {
    my_calc.tally(m)
    my_calc.total
  },
  new: ->(my_calc, m) {
    my_calc.new_tally(m)
    my_calc.total
  },
  args: [calc, n],
  call_both: true,
  dup_args: true
},
@result = calc.total

protect from arg mutation
calc = Calculator.new
params[:nums].each { |n|
    Suture.create :tally,
      old: ->(my_calc, m) {
        my_calc.tally(m)
        my_calc.total
      },
      new: ->(my_calc, m) {
        my_calc.new_tally(m)
        my_calc.total
      },
      args: [calc, n],
      call_both: true,
      dup_args: true
    }

@result = calc.total
calc = Calculator.new
params[:nums].each { |n|
  Suture.create :tally,
    old: ->(my_calc, m) {
      my_calc.tally(m)
      my_calc.total
    },
    new: ->(my_calc, m) {
      my_calc.new_tally(m)
      my_calc.total
    },
    args: [calc, n],
    call_both: true,
    dup_args: true
}
calc = Calculator.new
params[:nums].each { |n|
  Suture.create :tally,
    old: ->(my_calc, m) {
      my_calc.tally(m)
      my_calc.total
    },
    new: ->(my_calc, m) {
      my_calc.new_tally(m)
      my_calc.total
    },
    args: [calc, n],
    call_both: true,
    dup_args: true
}
@result = calc.total
calc = Calculator.new
params[:nums].each {|n|
  Suture.create :tally,
    old: ->(my_calc, m) {
      my_calc.tally(m)
      my_calc.total
    },
    new: ->(my_calc, m) {
      my_calc.new_tally(m)
      my_calc.total
    },
    args: [calc, n],
    call_both: true,
    dup_args: true
  }
@result = calc.total

calc never changes now!
calc = Calculator.new
params[:nums].each {|n|
  Suture.create :tally,
  old: ->(my_calc, m) {
    my_calc.tally(m)
    my_calc.total
  },
  new: ->(my_calc, m) {
    my_calc.new_tally(m)
    my_calc.total
  },
  args: [calc, n],
  call_both: true,
  dup_args: true
}
@result = calc.total
calc = Calculator.new

params[:nums].each {|n|
    Suture.create :tally,
        old: ->(my_calc, m) {
            my_calc.tally(m)
            my_calc.total
        },
        new: ->(my_calc, m) {
            my_calc.new_tally(m)
            my_calc.total
        },
        args: [calc, n],
        call_both: true,
        dup_args: true
    }

@result = calc.total

total is always nil
calc = Calculator.new
params[:nums].each { |n|
    Suture.create :tally,
        old: ->(my_calc, m) {
            my_calc.tally(m)
            my_calc.total
        },
        new: ->(my_calc, m) {
            my_calc.new_tally(m)
            my_calc.total
        },
        args: [calc, n],
        call_both: true,
        dup_args: true
    }
@result = calc.total
calc = Calculator.new
params[:nums].each { |n|
    Suture.create :tally,
    old: ->(my_calc, m) {
        my_calc.tally(m)
        calc = my_calc
        my_calc.total
    },
    new: ->(my_calc, m) {
        my_calc.new_tally(m)
        my_calc.total
    },
    args: [calc, n],
    call_both: true,
    dup_args: true
}
@result = calc.total
calc = Calculator.new
params[:nums].each { |n|
    Suture.create :tally,
        old: ->(my_calc, m) {
            my_calc.tally(m)
            calc = my_calc
            my_calc.total
        },
        new: ->(my_calc, m) {
            my_calc.new_tally(m)
            calc = my_calc
            my_calc.total
        },
        args: [calc, n],
        call_both: true,
        dup_args: true
    }
@result = calc.total
calc = Calculator.new
params[:nums].each { |n|
  Suture.create :tally,
  old: ->(my_calc, m) {
    my_calc.tally(m)
    calc = my_calc
    my_calc.total
  },
  new: ->(my_calc, m) {
    my_calc.new_tally(m)
    calc = my_calc
    my_calc.total
  },
  args: [calc, n],
  call_both: true,
  dup_args: true
}
@result = calc.total
calc = Calculator.new
params[:nums].each { |n|
    Suture.create :tally,
    old: ->(my_calc, m) {
        my_calc.tally(m)
        calc = my_calc
        my_calc.total
    },
    new: ->(my_calc, m) {
        my_calc.new_tally(m)
        calc = my_calc
        my_calc.total
    },
    args: [calc, n],
    call_both: true,
    dup_args: true
}
@result = calc.total
calc = Calculator.new
params[:nums].each { |n|
    Suture.create :tally,
        old: ->(my_calc, m) {
            my_calc.tally(m)
            calc = my_calc
            my_calc.total
        },
        new: ->(my_calc, m) {
            my_calc.new_tally(m)
            calc = my_calc
            my_calc.total
        },
        args: [calc, n],
        call_both: true,
        dup_args: true
    }
@result = calc.total
Remember: every mode is optional!
<table>
<thead>
<tr>
<th>Plan</th>
<th>Validate</th>
<th>Compare</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cut</td>
<td>Refactor</td>
<td>Fallback</td>
</tr>
<tr>
<td>Record</td>
<td>Verify</td>
<td>Delete</td>
</tr>
</tbody>
</table>
Fallback
Make change safe for users
New path errored? Try the old one!
Pure function
class Controller
  def show
    calc = Calculator.new
    @result = Suture.create :add,
    old: calc.method(:add),
    new: calc.method(:new_add),
    args: [
      params[:left],
      params[:right]
    ],
    call_both: true
  end
end
class Controller
  def show
    calc = Calculator.new
    @result = Suture.create :add,
    old: calc.method(:add),
    new: calc.method(:new_add),
    args: [
      params[:left],
      params[:right]
    ],
    call_both: true
  end
end
class Controller
  def show
    calc = Calculator.new
    @result = Suture.create :add,
      old: calc.method(:add),
      new: calc.method(:new_add),
      args: [
        params[:left],
        params[:right]
      ],
      fallback_on_error: true
  end
end
class Controller
  def show
    calc = Calculator.new
    @result = Suture.create :add,
      old: calc.method(:add),
      new: calc.method(:new_add),
      args: [
        params[:left],
        params[:right]
      ],
    fallback_on_error: true
  end
end
class Controller
  def show
    calc = Calculator.new
    @result = Suture.create :add,
      old: calc.method(:add),
      new: calc.method(:new_add),
      args: [params[:left], params[:right]],
      fallback_on_error: true
  end
end
Rescues :new with :old
class Controller
  def show
    calc = Calculator.new
    @result = Suture.create :add,
    old: calc.method(:add),
    new: calc.method(:new_add),
    args: [params[:left],
      params[:right]],
    fallback_on_error: true
  end
end

Rescues :new with :old
calc = Calculator.new
params[:nums].each { |n|
  Suture.create :tally, 
    old: ->(my_calc, m) {
      my_calc.tally(m)
      calc = my_calc
      my_calc.total
    },
    new: ->(my_calc, m) {
      my_calc.new_tally(m)
      calc = my_calc
      my_calc.total
    },
    args: [calc, n],
    call_both: true,
    dup_args: true
  }
@result = calc.total
calc = Calculator.new
params[:nums].each { |n|
    Suture.create :tally,
        old: ->(my_calc, m) {
            my_calc.tally(m)
            calc = my_calc
            my_calc.total
        },
        new: ->(my_calc, m) {
            my_calc.new_tally(m)
            calc = my_calc
            my_calc.total
        },
        args: [calc, n],
        call_both: true,
        dup_args: true
    }
@result = calc.total
calc = Calculator.new
params[:nums].each { |n|
  Suture.create :tally,
  old: ->(my_calc, m) {
    my_calc.tally(m)
    calc = my_calc
    my_calc.total
  },
  new: ->(my_calc, m) {
    my_calc.new_tally(m)
    calc = my_calc
    my_calc.total
  },
  args: [calc, n],
  fallback_on_error: true,
  dup_args: true
}
@result = calc.total
calc = Calculator.new
params[:nums].each { |n|
  Suture.create :tally,
    old: ->(my_calc, m) {
      my_calc.tally(m)
      calc = my_calc
      my_calc.total
    },
    new: ->(my_calc, m) {
      my_calc.new_tally(m)
      calc = my_calc
      my_calc.total
    },
    args: [calc, n],
    fallback_on_error: true,
    dup_args: true
  }
@result = calc.total
calc = Calculator.new
params[:nums].each { |n|
  Suture.create :tally, 
      old: ->(my_calc, m) {
        my_calc.tally(m)
        calc = my_calc
        my_calc.total
      },
    new: ->(my_calc, m) {
      my_calc.new_tally(m)
      calc = my_calc
      my_calc.total
    },
  args: [calc, n],
  fallback_on_error: true,
  dup_args: true
}
@result = calc.total
Faster than call_both
Fewer side effects
All errors are logged
Allow certain errors via `expected_error_types`
<table>
<thead>
<tr>
<th>Plan</th>
<th>Validate</th>
<th>Compare</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cut</td>
<td>Refactor</td>
<td>Fallback</td>
</tr>
<tr>
<td>Record</td>
<td>Verify</td>
<td>Delete</td>
</tr>
</tbody>
</table>
Delete
Like stitches, remove once the wound heals
Pure function
def test_new_add
    calc = Calculator.new
    Suture.verify :add,
        subject: calc.method(:new_add)
end
Suture.delete_all!(:add)
class Controller
  def show
    calc = Calculator.new
    @result = Suture.create :add,
      old: calc.method(:add),
      new: calc.method(:new_add),
      args: [
        params[:left],
        params[:right]
      ],
      fallback_on_error: true
  end
end
class Controller
  def show
    calc = Calculator.new
    @result = Suture.create :add,
      old: calc.method(:add),
      new: calc.method(:new_add),
      args: [
        params[:left],
        params[:right]
      ],
      fallback_on_error: true
  end
end
class Controller
  def show
    calc = Calculator.new
    @result = Suture.create :add,
        old: calc.method(:add),
        new: calc.method(:new_add),
        args: [
            params[:left],
            params[:right]
        ],
        fallback_on_error: true
  end
end
class Controller
  def show
    calc = Calculator.new
    @result = Suture.create :add,
    args: [
      params[:left],
      params[:right]
    ],
    fallback_on_error: true
  end
end
class Controller
  def show
    calc = Calculator.new
    @result = Suture.create :add,
    params[:left],
    params[:right]
    fallback_on_error: true
  end
end
class Controller
  def show
    calc = Calculator.new
    @result = Suture.create :add,
      old: calc.method(:add),
      new: calc.method(:new_add),
      args: [params[:left],
             params[:right]]
    fallback_on_error: true
  end
end
class Controller
  def show
    calc = Calculator.new
    @result = Suture.create :add,
                      old: calc.method(:add),
                      new: calc.method(:new_add),
                      args: [
                        params[:left],
                        params[:right]
                      ]
  end
end
class Controller
  def show
    calc = Calculator.new
    @result = calc.new_add(
      old: calc.method(:add),
      new: calc.method(:new_add),
      args: [params[:left], params[:right]]
    )
    fallback_on_error: true
  end
end
class Controller
  def show
    calc = Calculator.new
    @result = calc.new_add(
      params[:left],
      params[:right]
    )
  end
end
class Controller
  def show
    calc = Calculator.new
    @result = calc.new_add(
      params[:left],
      params[:right]
    )
  end
end
Mutation
def test_new_tally
  Suture.verify :tally, 
  subject: ->(calc, n){
    calc.new_tally(n)
    calc.total
  }
}
end
Suture.delete_all!(:tally)
calc = Calculator.new
params[:nums].each { |n|
  Suture.create :tally,
    old: ->(my_calc, m) {
      my_calc.tally(m)
      calc = my_calc
      my_calc.total
    },
    new: ->(my_calc, m) {
      my_calc.new_tally(m)
      calc = my_calc
      my_calc.total
    },
    args: [calc, n],
    fallback_on_error: true,
    dup_args: true
  }
@result = calc.total
calc = Calculator.new
params[:nums].each { |n|
  Suture.create :tally,
    old: ->(my_calc, m) {
      my_calc.tally(m)
      calc = my_calc
      my_calc.total
    },
    new: ->(my_calc, m) {
      my_calc.new_tally(m)
      calc = my_calc
      my_calc.total
    },
    args: [calc, n],
    fallback_on_error: true,
    dup_args: true
}
@result = calc.total
calc = Calculator.new
params[:nums].each{|n|
  Suture.create :tally,
    old: ->(my_calc, m) {
      my_calc.tally(m)
      calc = my_calc
      my_calc.total
    },
    new: ->(my_calc, m) {
      my_calc.new_tally(m)
      calc = my_calc
      my_calc.total
    },
    args: [calc, n],
    fallback_on_error: true,
    dup_args: true
}
@result = calc.total
calc = Calculator.new
params[:nums].each { |n|
  Suture.create :tally,

new: ->(my_calc, m) {
  my_calc.new_tally(m)
  calc = my_calc
  my_calc.total
},
args: [calc, n],
fallback_on_error: true,
dup_args: true
}
@result = calc.total
calc = Calculator.new
params[:nums].each { |n|
  Suture.create :tally,
    new: ->(my_calc, m) {
      my_calc.new_tally(m)
      calc = my_calc
      my_calc.total
    },
    args: [calc, n],
    fallback_on_error: true,
    dup_args: true
  }
@result = calc.total
calc = Calculator.new
params[:nums].each { |n|
    Suture.create :tally,
        new: ->(my_calc, m) {
            my_calc.new_tally(m)
            calc = my_calc
            my_calc.total
        },
        args: [calc, n],
        dup_args: true
}
@result = calc.total
calc = Calculator.new
params[:nums].each { |n|
  Suture.create :tally,
    new: ->(my_calc, m) {
      my_calc.new_tally(m)
      calc = my_calc
      my_calc.total
    },
    args: [calc, n],
  },
}@result = calc.total
calc = Calculator.new
params[:nums].each { |n|
  Suture.create :tally,
    new: ->(my_calc, m) {
      my_calc.new_tally(m)
      calc = my_calc
      my_calc.total
    },
}

@result = calc.total
calc = Calculator.new
params[:nums].each { |n|
  Suture.create :tally,

  new: ->(my_calc, m) {
    my_calc.new_tally(m)
    calc = my_calc
    my_calc.total
  },

}
calc = Calculator.new
params[:nums].each{|n|
  Suture.create :tally,
  old: ->(my_calc, m) {
    my_calc.tally(m)
    calc = my_calc
    my_calc.total
  },
  new: ->(my_calc, m) {
    my_calc.new_tally(n)
    calc = my_calc
    my_calc.total
  },
  args: [calc, n],
  fallback_on_error: true,
  dup_args: true
}
@result = calc.total
calc = Calculator.new
params[:nums].each {|n|
    Suture.create :tally,
        calc = my_calc
    my_calc.total

    new: ->(my_calc, m) {
        my_calc.new_tally(n)
        calc = my_calc
        my_calc.total
    }
}

@result = calc.total
calc = Calculator.new
params[:nums].each { |n|
  Suture.create :tally,
  old: ->(my_calc, m) {
    my_calc.tally(m)
    calc = my_calc
    my_calc.total
  },
  new: ->(my_calc, m) {
    my_calc.new_tally(n)
    calc = my_calc
    my_calc.total
  },
  args: [calc, n],
  fallback_on_error: true,
  dup_args: true
}
@result = calc.total
calc = Calculator.new
params[[:nums]].each { |n|
    calc.new_tally(n)
}
@result = calc.total
calc = Calculator.new
params[:nums].each { |n|
  calc.new_tally(n)
}
@result = calc.total
We did it!
Suture is ready to use!
Suture is ready to use!

github.com/testdouble/suture
Suture is ready to use!

1.0.0
Together, let's make refactors less scary 😊
One last thing...
How I met Ruby
My homestay brother was also a programmer.
皆さんへ、
皆さんへ、
お話する機会を
いただきありがとうございます！
皆さんへ、
お話する機会を
いただきありがとう！
感謝の気持ちで
いっぱいです！


皆さんへ、
お話しする機会を
いただきありがとうございます！
感謝の気持ちで
いっぱいです！
I'm @searls—tell me what you think 😁!
I'm in Kansai all month!

justin@testdouble.com
もう一回ありがとう！❤️