Tricky Testing

@AndrewRadev

Test structure

- Setup
- Exercise
- Verify
- Teardown

```ruby
describe Stack do

  it "allows popping the top element" do
    # Setup
    stack = Stack.new
    stack.push("one")
    stack.push("two")

    # Exercise
    top_element = stack.pop

    # Verify
    expect(top_element).to eq "two"
  end

end
```

Tricky Testing

waiting_on_rails

(demo)

`waiting-on-rails server`

- Run `rails server` command
- Run music
- Output lines to stdout
- When we see “Listening on tcp://...”, ding!

How would I test this?

```ruby
module WaitingOnRails
  class Player

    def initialize(music_path)
      @music_path = full_path(music_path)
    end

    private

    def full_path(path)
      File.expand_path(
        "#{File.dirname(__FILE__)}/../../vendor/#{path}"
      )
    end

  end
end
```

```ruby
module WaitingOnRails
  class Player
    # ...

    attr_reader :pid

    def start
      @pid = Kernel.spawn("mplayer #{@music_path}", {
        out: '/dev/null',
        err: '/dev/null',
      })
    end

    def stop
      return true if @pid.nil?
      Process.kill(15, @pid)
      Process.wait(@pid)
      true
    rescue Errno::ESRCH, Errno::ECHILD
      false
    end

    # ...
  end
end
```

```ruby
module WaitingOnRails
  describe Player do

    let(:player) { Player.new('test.mp3') }

    it "spawns an external process" do
      expect(Kernel).to receive(:spawn)
      player.start
    end

    it "kills itself correctly" do
      expect(Kernel).to receive(:spawn).and_return(123)
      player.start

      expect(Process).to receive(:kill).with(15, 123)
      expect(Process).to receive(:wait).with(123)
      player.stop
    end

  end
end
```

Changing code

```ruby
module WaitingOnRails
  class Player
    # ...

    attr_reader :pid

    def start
      @pid = spawn("mplayer #{@music_path}", {
        out: '/dev/null',
        err: '/dev/null',
      })
    end

    def stop
      return true if @pid.nil?
      Process.kill(9, @pid)
      Process.wait(@pid)
      true
    rescue Errno::ESRCH, Errno::ECHILD
      false
    end

    # ...
  end
end
```

Big idea:
Test what’s important

```ruby
module WaitingOnRails
  class Player
    # ...

    attr_reader :pid

    def start
      @pid = Kernel.spawn("mplayer #{@music_path}", {
        out: '/dev/null',
        err: '/dev/null',
      })
    end

    def stop
      return true if @pid.nil?
      Process.kill(15, @pid)
      Process.wait(@pid)
      true
    rescue Errno::ESRCH, Errno::ECHILD
      false
    end

    # ...
  end
end
```

```ruby
module WaitingOnRails
  describe Player do

    let(:player) { Player.new('test.mp3') }

    it "spawns an external process" do
      player.start
      expect(player.pid).to be_a_running_process
    end

    it "kills itself correctly" do
      player.start
      player.stop

      expect(player.pid).to_not be_a_running_process
    end

  end
end
```

```ruby
RSpec::Matchers.define :be_a_running_process do

  match do |pid|
    if pid.nil?
      false
    else
      begin
        Process.getpgid(pid)
        true
      rescue Errno::ESRCH
        false
      end
    end
  end

end
```

```ruby
module WaitingOnRails
  describe Player do

    let(:player) { Player.new('test.mp3') }

    it "spawns an external process" do
      player.start
      expect(player.pid).to be_a_running_process
    end

    it "kills itself correctly" do
      player.start
      player.stop

      expect(player.pid).to_not be_a_running_process
    end

  end
end
```

```ruby
config.around do |example|
  original_path = ENV['PATH']
  ENV['PATH'] =
    "#{File.expand_path('spec/support/bin')}:#{original_path}"

  example.call

  ENV['PATH'] = original_path
end



# ./spec/support/bin/mplayer
#!/bin/sh

while true; do
  sleep 1
done
```

```ruby
module WaitingOnRails
  class Exit < StandardError; end

  class Rails
    def initialize(music_player, ding_player)
      @music_player = music_player
      @ding_player  = ding_player
    end

    # ...
  end
end
```

```ruby
module WaitingOnRails
  class Rails
    # ...

    def run(args)
      if not should_play_music?(args)
        exec_rails_command(args)
      end

      # ...
    end

    private

    def should_play_music?(args)
      args.find { |arg| ['server', 's'].include? arg }
    end

    def exec_rails_command(args)
      exec 'rails', *args
    end

    # ...
  end
end
```

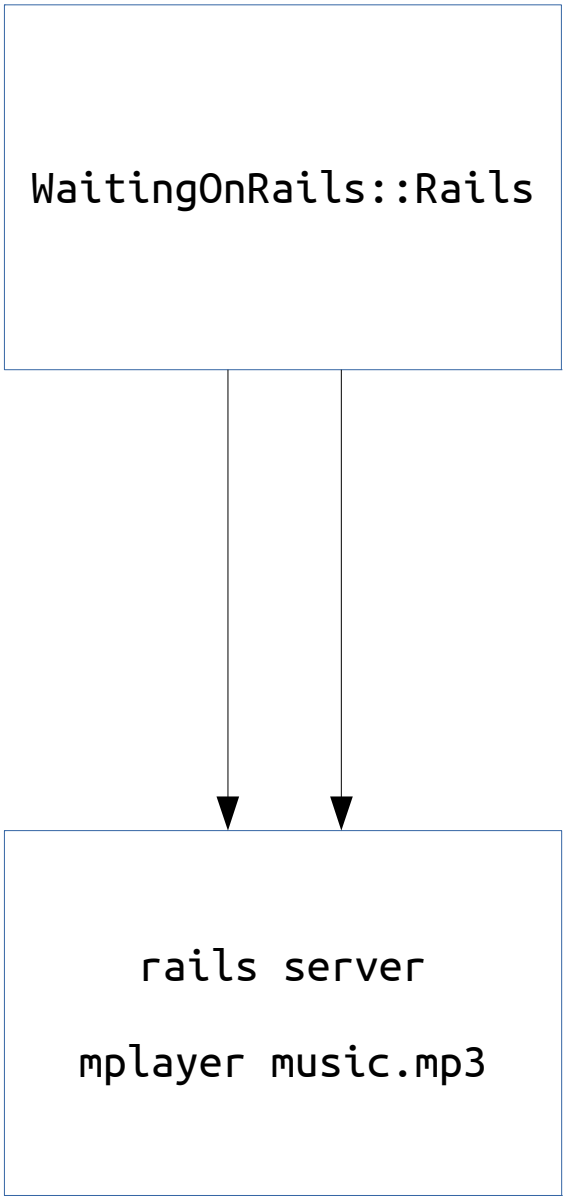
```ruby
module WaitingOnRails
  class Rails
    # ...

    def run(args)
      if not should_play_music?(args)
        exec_rails_command(args)
      end

      spawn_rails_subprocess(args) do |output, pid|
        # ...
      end
    end

    private

    def spawn_rails_subprocess(args)
      PTY.spawn('rails', *args) do |output, input, pid|
        yield output, pid
      end
    end

    # ...
  end
end
```

```ruby
module WaitingOnRails
  class Rails
    # ...

    def run(args)
      if not should_play_music?(args)
        exec_rails_command(args)
      end

      spawn_rails_subprocess(args) do |output, pid|
        @music_player.start

        # ...
      end
    ensure
      @music_player.stop
    end

    private

    # ...
  end
end
```

```ruby
module WaitingOnRails
  class Rails
    # ...

    def run(args)
      if not should_play_music?(args)
        exec_rails_command(args)
      end

      spawn_rails_subprocess(args) do |output, pid|
        @music_player.start
        handle_signals(pid, output)

      end
    rescue Exit
      exit(1)
    ensure
      @music_player.stop
    end

    private

    # ...
  end
end
```

```ruby
module WaitingOnRails
  class Rails
    # ...

    def run(args)
      if not should_play_music?(args)
        exec_rails_command(args)
      end

      spawn_rails_subprocess(args) do |output, pid|
        @music_player.start
        handle_signals(pid, output)
        main_loop(output)
      end
    rescue Exit
      exit(1)
    ensure
      @music_player.stop
    end

    private

    # ...
  end
end
```

```ruby
module WaitingOnRails
  class Rails
    # ...

    private

    def main_loop(io)
      loop do
        begin
          line = io.readline
          puts line
          if matches_server_start?(line)
            @music_player.stop
            sleep 0.5
            @ding_player.start if @ding_player
          end
        rescue EOFError
          break
        rescue Errno::EIO
          raise Exit
        end
      end
    end

    # ...
  end
end
```

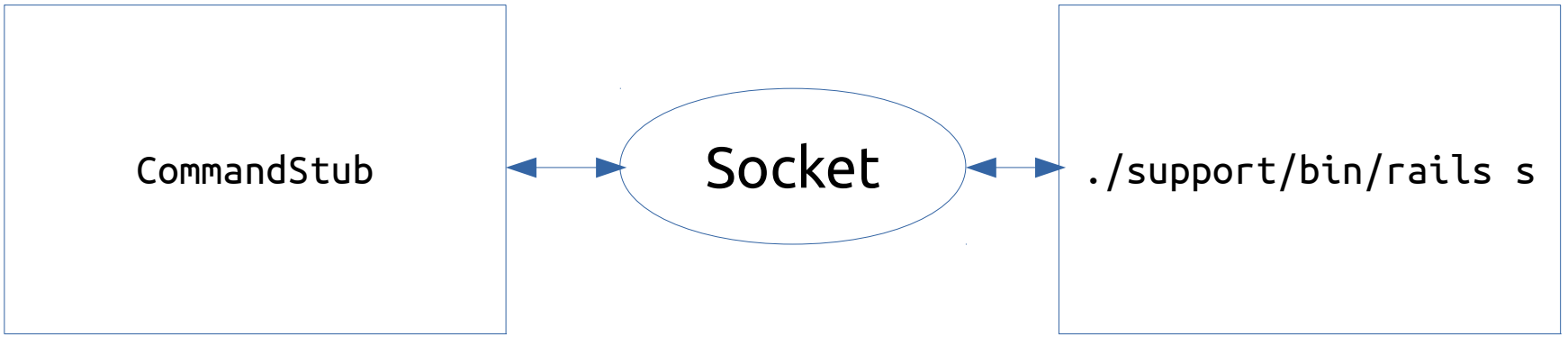

WaitingOnRails::Rails
rails server
mplayer music.mp3

CommandStub

```ruby
# ./spec/support/bin/rails
#! /usr/bin/env ruby

require 'socket'

filename = File.expand_path(
  File.dirname(__FILE__) + '/../../../tmp/rails.socket'
)
socket = UNIXServer.new(filename).accept

begin
  while line = socket.readline
    puts line
  end
rescue EOFError
end
```

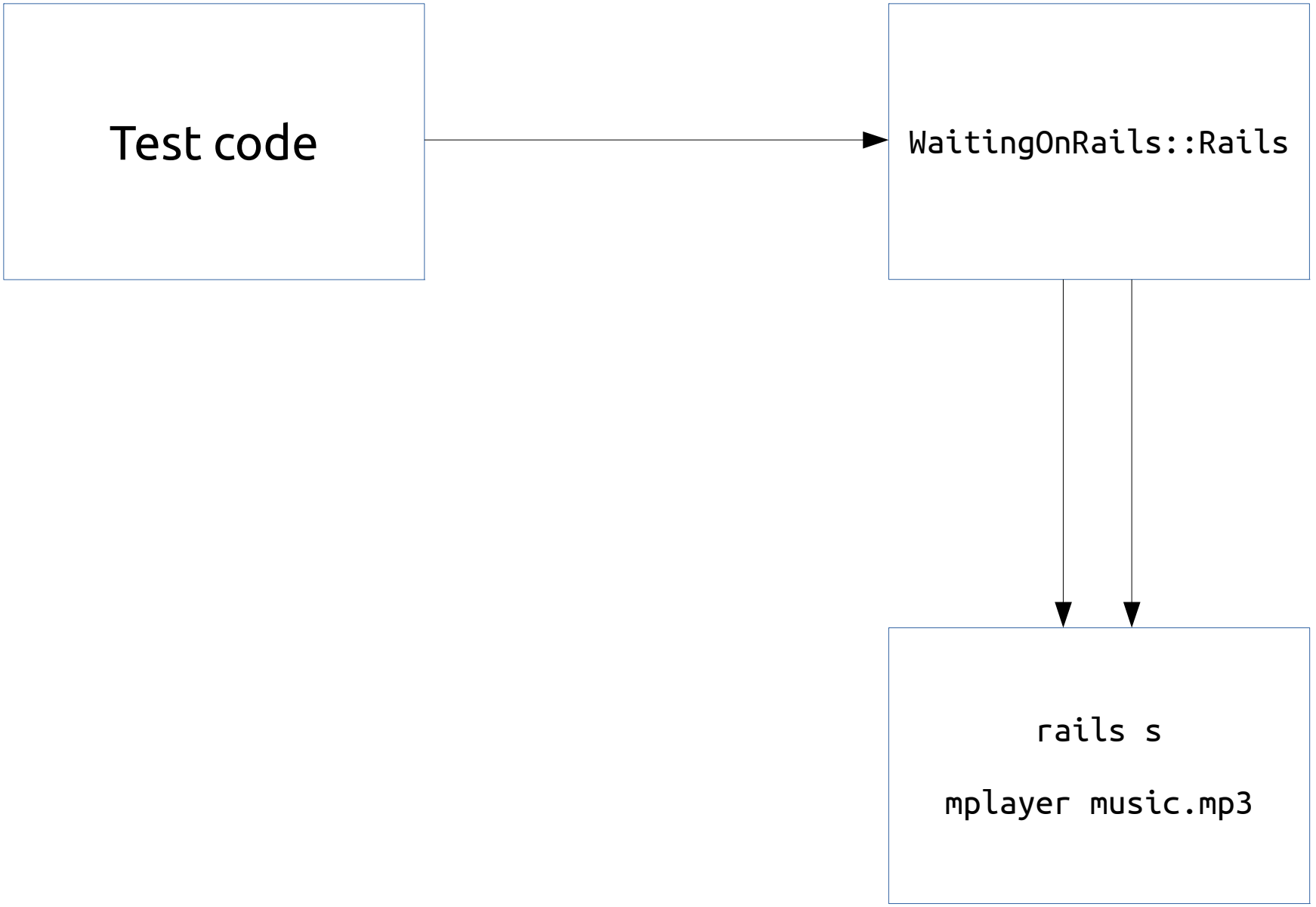
```ruby
module Support
  class CommandStub
    def initialize(command)
      @command = command
    end

    def init
      return if @socket

      Timeout.timeout(2) do
        @socket = UNIXSocket.new(socket_path)
      end
    rescue Errno::ENOENT
      retry
    end

    # ...

    private

    def socket_path
      File.expand_path(
        File.dirname(__FILE__) + "/../../tmp/#{@command}.socket"
      )
    end
  end
end
```

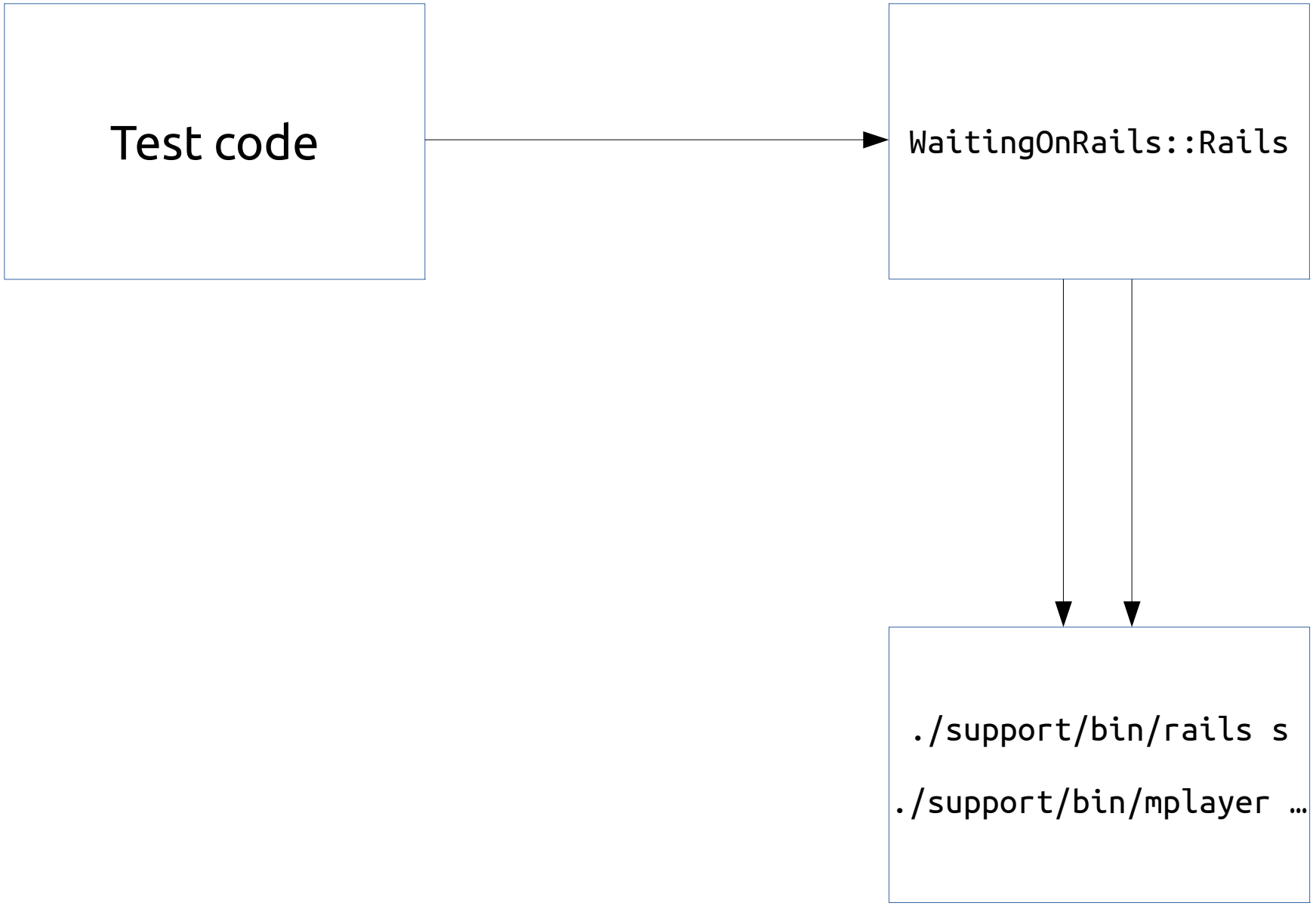
```ruby
module Support
  class CommandStub
    # ...

    def add_output(string)
      init
      @socket.write(string)
    end

    def finish
      if @socket
        @socket.close
        @socket = nil
      end
    end

    # ...
  end
end
```

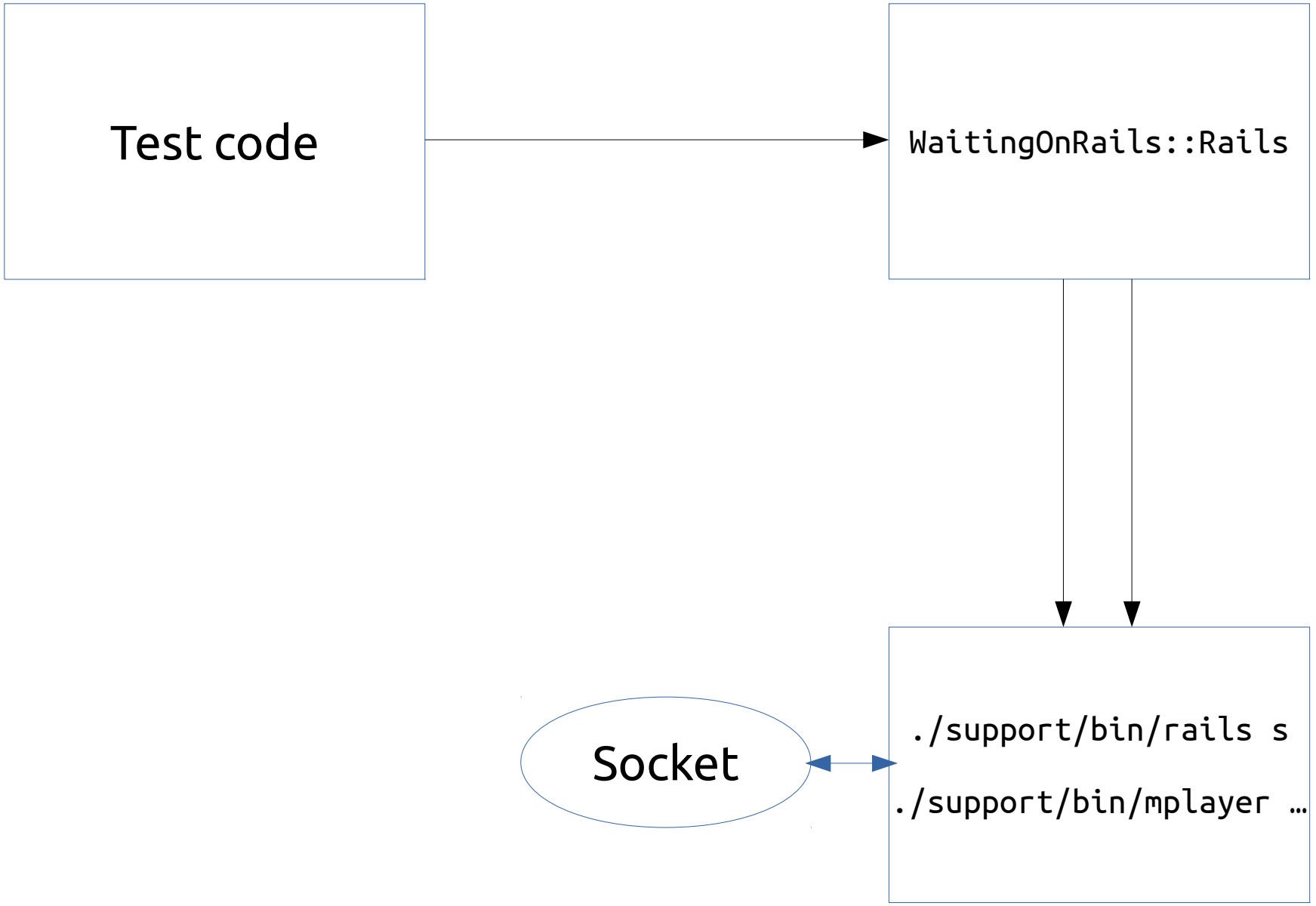

CommandStub
Socket
./support/bin/rails s

```ruby
module WaitingOnRails
  describe Rails do
    let(:player)     { Player.new('test.mp3') }
    let(:runner)     { Rails.new(player) }
    let(:rails_stub) { Support::CommandStub.new('rails') }

    it "stops the music after seeing that the server was started" do
      thread = Thread.new { runner.run(['server']) }

      # ...
    end
  end
end
```

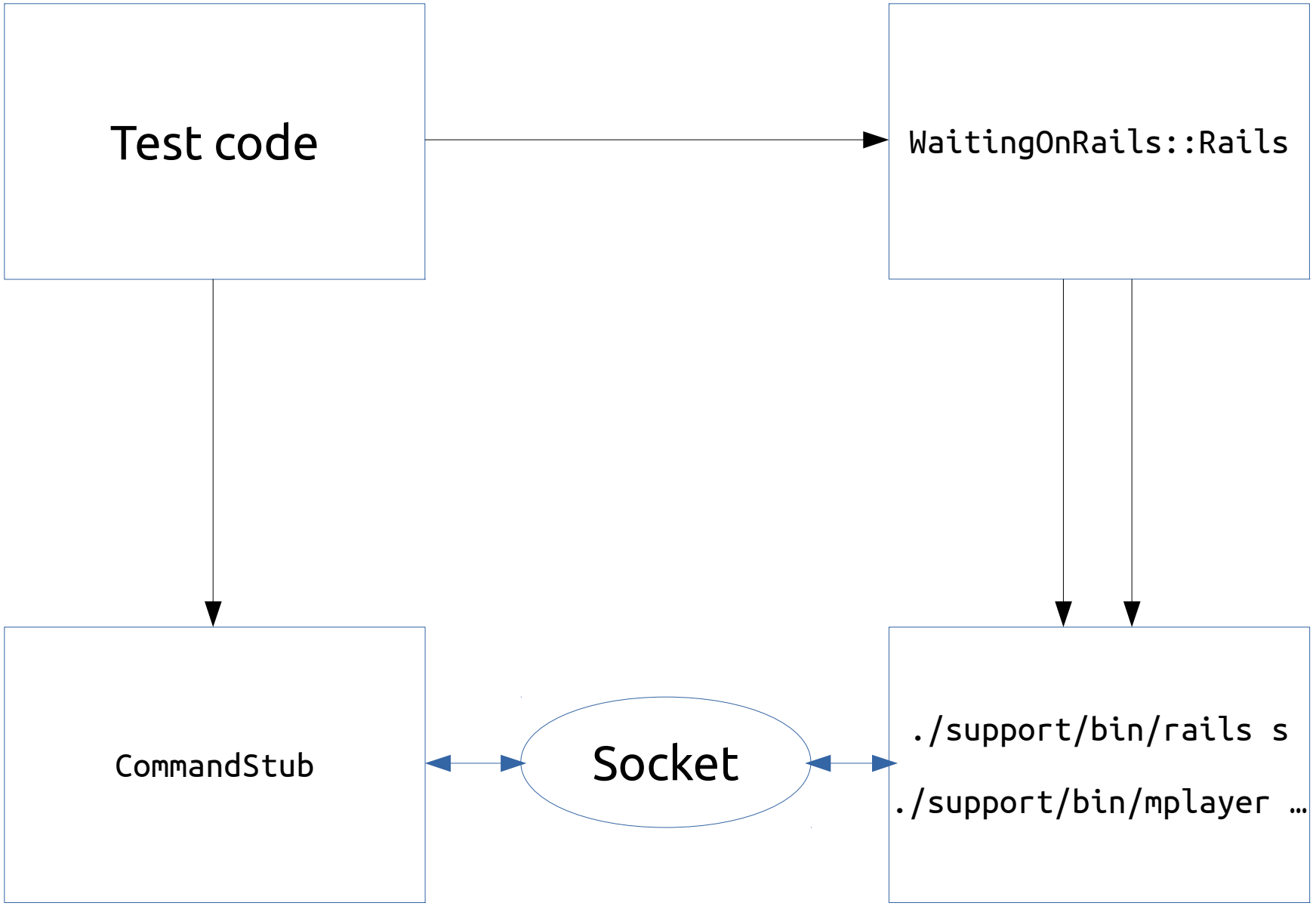

Test code
WaitingOnRails::Rails


Test code
WaitingOnRails::Rails
rails s
mplayer music.mp3


Test code
WaitingOnRails::Rails
./support/bin/rails s
./support/bin/mplayer …

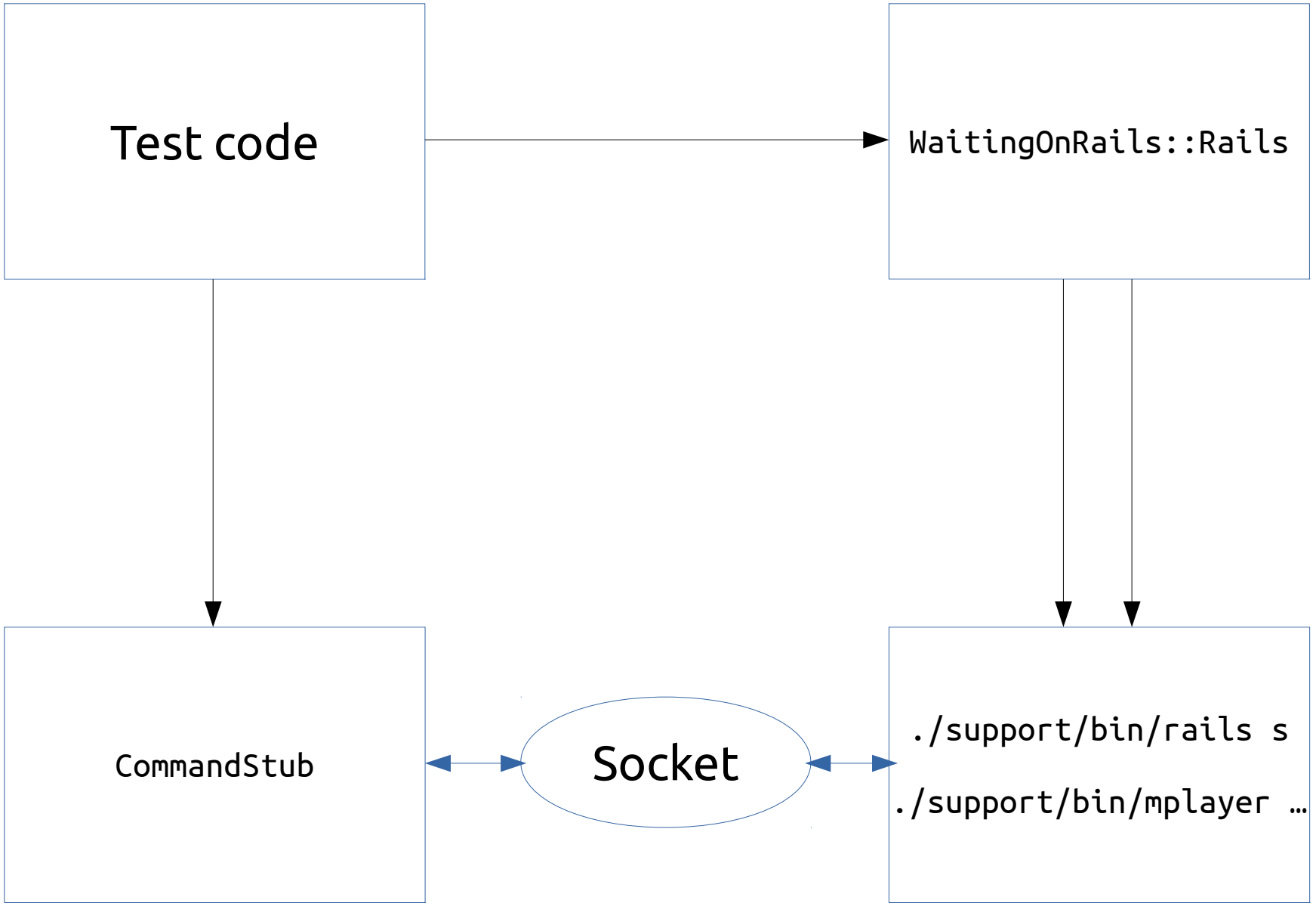

Test code
WaitingOnRails::Rails
./support/bin/rails s
./support/bin/mplayer …
Socket

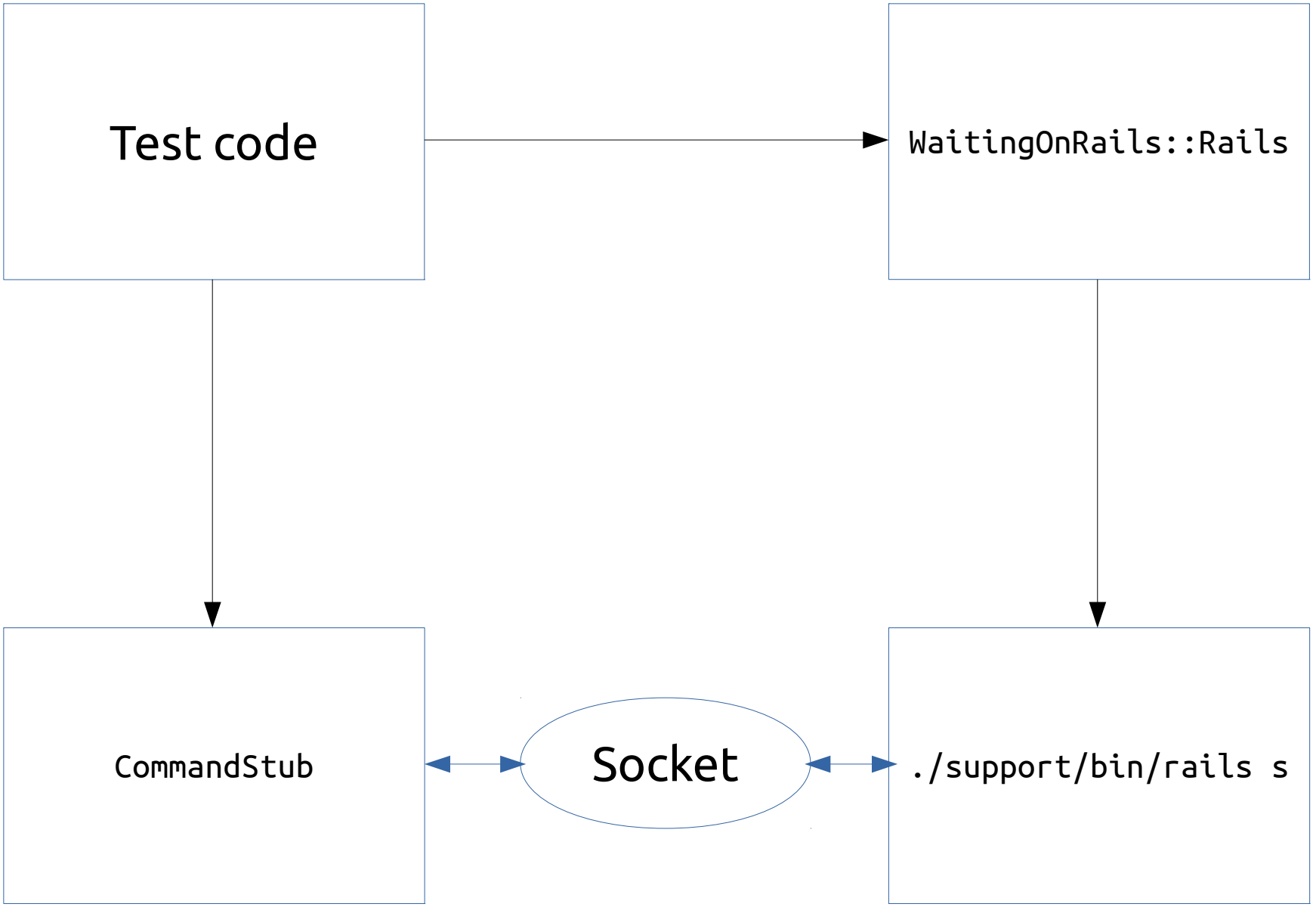
```ruby
module WaitingOnRails
  describe Rails do
    # ...

    it "stops the music after seeing that the server was started" do
      thread = Thread.new { runner.run(['server']) }

      rails_stub.init

      # ...
    end
  end
end
```


Test code
WaitingOnRails::Rails
CommandStub
Socket
./support/bin/rails s
./support/bin/mplayer …

```ruby
module WaitingOnRails
  describe Rails do
    # ...

    it "stops the music after seeing that the server was started" do
      thread = Thread.new { runner.run(['server']) }

      rails_stub.init
      rails_stub.add_output <<-EOF
        => Booting Puma
        => Rails 5.x.x application starting in development ...
        => Run `rails server -h` for more startup options
      EOF

      expect(player.pid).to be_a_running_process

      # ...
    end
  end
end
```

```ruby
module WaitingOnRails
  describe Rails do
    # ...

    it "stops the music after seeing that the server was started" do
      # ...

      rails_stub.add_output <<-EOF
        Puma starting in single mode...
        * Version 3.x.x (ruby 2.x.x-pxxx), codename: ...
        * Min threads: 5, max threads: 5
        * Environment: development
        * Listening on tcp://localhost:3000
        Use Ctrl-C to stop
      EOF

      # ...
    end
  end
end
```


Test code
WaitingOnRails::Rails
CommandStub
Socket
./support/bin/rails s
./support/bin/mplayer …


Test code
WaitingOnRails::Rails
CommandStub
Socket
./support/bin/rails s

```ruby
module WaitingOnRails
  describe Rails do
    # ...

    it "stops the music after seeing that the server was started" do
      # ...

      rails_stub.add_output <<-EOF
        Puma starting in single mode...
        * Version 3.x.x (ruby 2.x.x-pxxx), codename: ...
        * Min threads: 5, max threads: 5
        * Environment: development
        * Listening on tcp://localhost:3000
        Use Ctrl-C to stop
      EOF
      sleep 0.5

      expect(player.pid).to_not be_a_running_process
    end
  end
end
```

`waiting-on-rake routes`

- Run `rake routes` command
- Run music
- Output lines to stdout
- When `rake` process exits, ding!

```ruby
module WaitingOnRails
  describe Rake do
    let(:player)    { Player.new('test.mp3') }
    let(:runner)    { Rake.new(player) }
    let(:rake_stub) { Support::CommandStub.new('rake') }

    it "plays music during the entire running of the command" do
      thread = Thread.new { runner.run(['routes']) }

      rake_stub.init

      expect(player.pid).to be_a_running_process

      rake_stub.finish
      thread.join

      expect(player.pid).to_not be_a_running_process
    end
  end
end
```

Big idea:
Test what’s important

Value / Cost

- High value, low cost – super!
- Low value, low cost – not so great!
- High value, high cost – ugh!
- More: High Cost Tests and High Value Tests

Faker

```
irb(main):001:0> require 'faker'
=> true

irb(main):002:0> Faker::Lorem.sentence
=> "Omnis maiores eum qui."

irb(main):003:0> Faker::PhoneNumber.phone_number
=> "(910) 406-8835 x66233"

irb(main):004:0> Faker::Address.street_address
=> "5020 Charity Court"
```

```
irb(main):005:0> Faker::Hipster.words
=> ["tilde", "try-hard", "butcher"]
irb(main):006:0> Faker::Hipster.sentence
=> "Stumptown gluten-free before they sold out 3 wolf moon tilde
iphone tattooed."

irb(main):007:0> Faker::Company.bs
=> "benchmark mission-critical portals"
irb(main):008:0> Faker::Company.bs
=> "streamline frictionless partnerships"

irb(main):009:0> Faker::Hacker.say_something_smart
=> "We need to quantify the auxiliary RSS program!"

irb(main):010:0> Faker::Hacker.say_something_smart
=> "If we back up the feed, we can get to the PCI interface through
the virtual GB pixel!"

irb(main):011:0> Faker::Hacker.say_something_smart
=> "The JBOD program is down, compress the haptic array so we can
compress the XSS application!"
```

```ruby
require 'faker'

describe Faker::Hipster do
  it "generates the right words, I guess" do
    Kernel.srand(123)

    expect(Faker::Hipster.words(3)).to eq ['tumblr', 'hella', 'yr']
  end

  it "generates a particular sentence with this particular seed" do
    Kernel.srand(123)

    expect(Faker::Hipster.sentence).to eq(
      'Tumblr seitan yr lomo plaid retro normcore biodiesel salvia.'
    )
  end
end
```

A single case

```ruby
class TestFakerHipster < Test::Unit::TestCase

  def setup
    @tester = Faker::Hipster
  end

  def test_words_without_spaces
    @words = @tester.words(1000)
    @words.each { |w| assert !w.match(/\s/) }
  end

  def test_words_with_large_count_params
    exact = @tester.words(500)
    range = @tester.words(250..500)
    array = @tester.words([250, 500])

    assert(exact.length == 500)
    assert(250 <= range.length && range.length <= 500)
    assert(array.length == 250 || array.length == 500)
  end

end
```

Big idea:
Property-based testing

Randomness

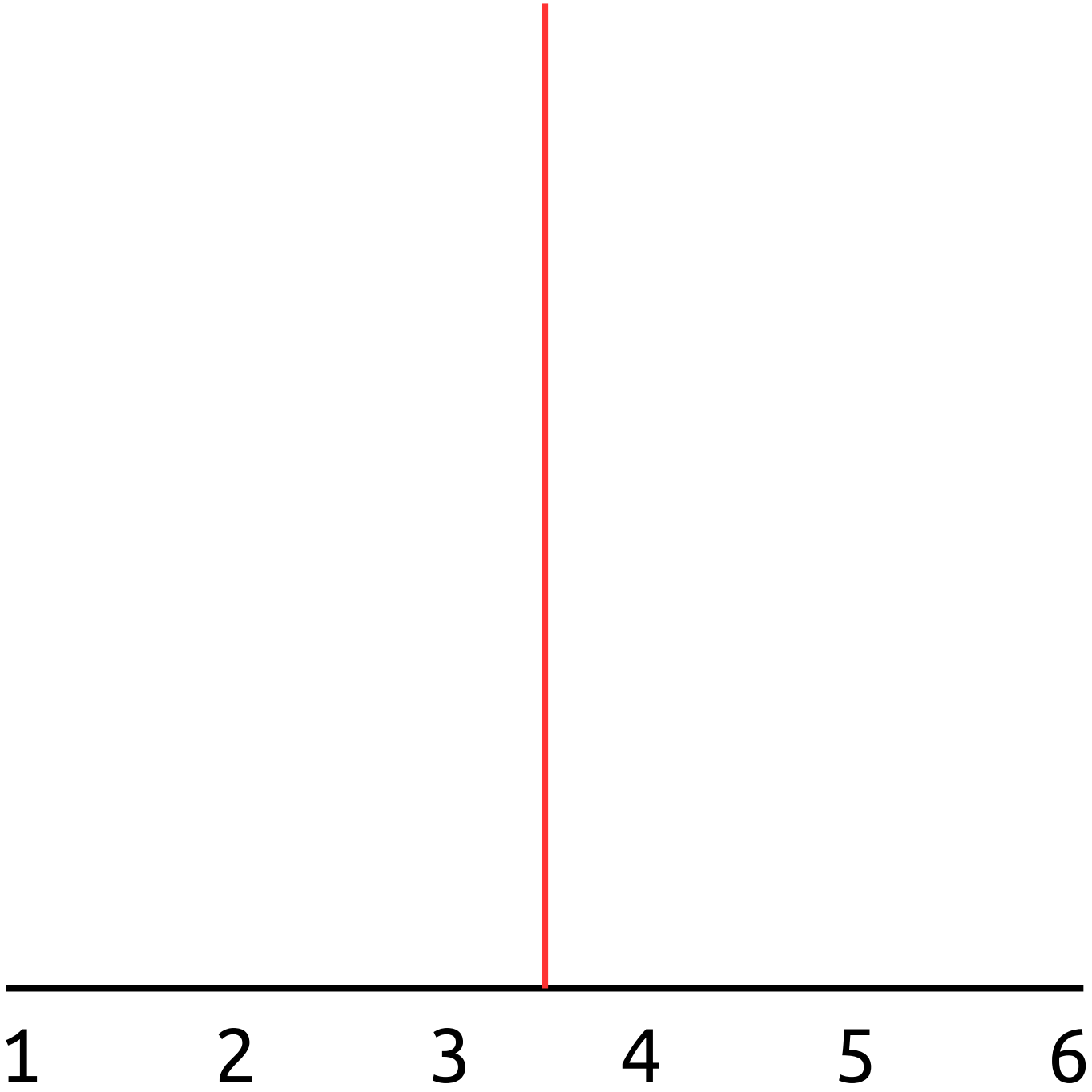
```ruby
describe Dice do
  describe "result of #roll" do

    it "is between 1 and 6" do
      10_000.times do
        expect(Dice.roll).to be_between(1, 6).inclusive
      end
    end

  end
end
```

```ruby
module Dice

  def self.roll
    rand(5) + 1
  end

end
```

```ruby
module Dice

  def self.roll
    # chosen by a fair dice roll
    return 4
  end

end
```

Properties of a 6-sided die

- Value is 1 to 6
- Mean value is 3.5


1
2
3
4
5
6

```ruby
describe Dice do
  describe "result of #roll" do

    it "is between 1 and 6" do
      10_000.times do
        expect(Dice.roll).to be_between(1, 6).inclusive
      end
    end

    it "returns results with the right mean" do
      results = 10_000.times.map { Dice.roll }

      expect(results.mean.round(1)).to eq 3.5
    end

  end
end

class Array
  def mean
    inject(&:+).to_f / size
  end
end
```

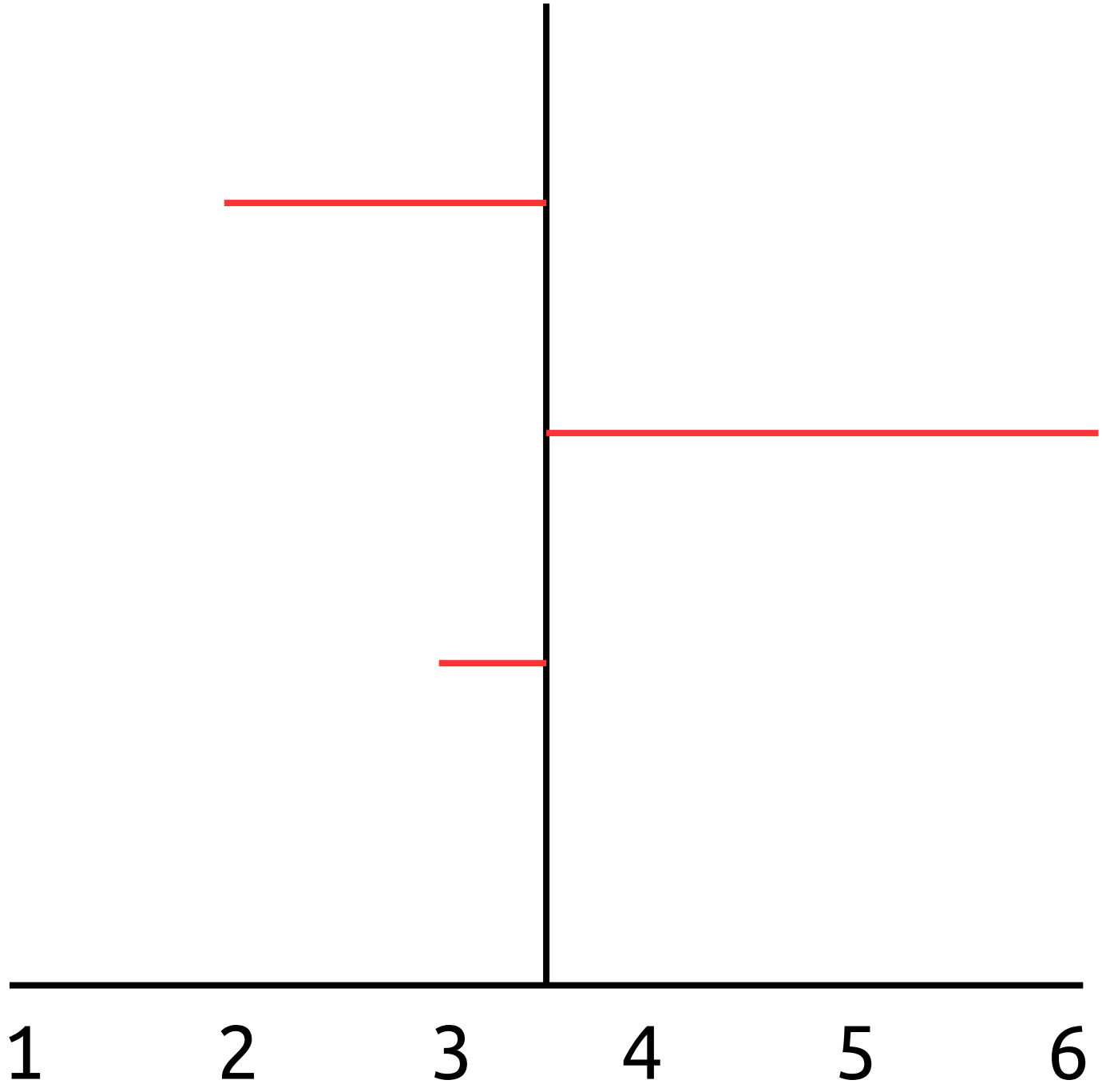
`rand(max)`

"When max is an `Integer`, rand returns a random integer greater than or equal to zero and less than max."

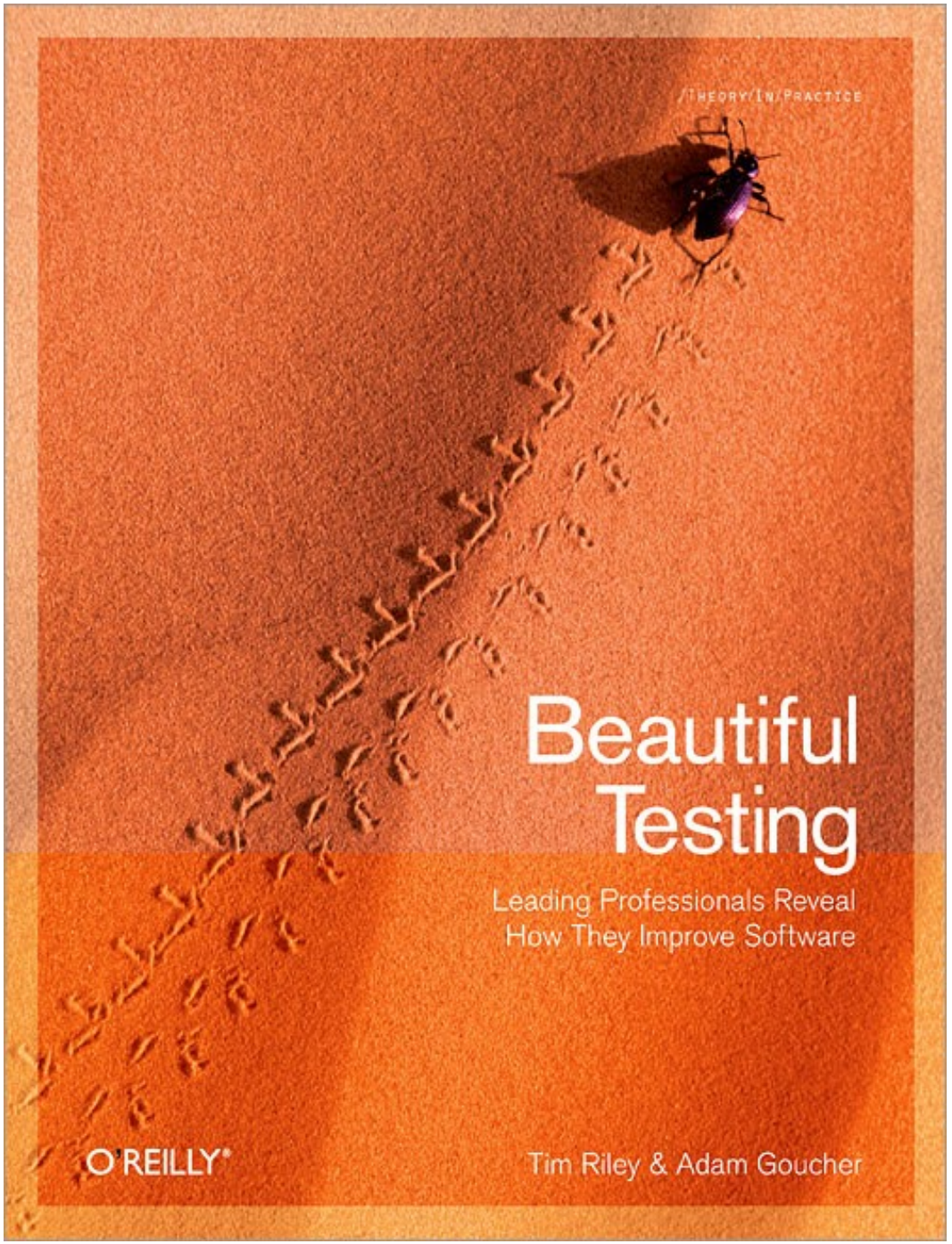
```ruby
module Dice

  def self.roll
    rand(6) + 1
  end

end
```

Properties of a 6-sided die

- Value is 1 to 6
- Mean value is ~3.5
- Variance is ~2.9


1
2
3
4
5
6

```ruby
describe Dice do
  describe "result of #roll" do

    it "returns results with the right variance" do
      results = 10_000.times.map { Dice.roll }

      expect(results.variance.round(1)).to eq 2.9
    end

  end
end

class Array
  def variance
    expected_value = mean
    map { |value| (value - expected_value) ** 2 }.mean
  end
end
```

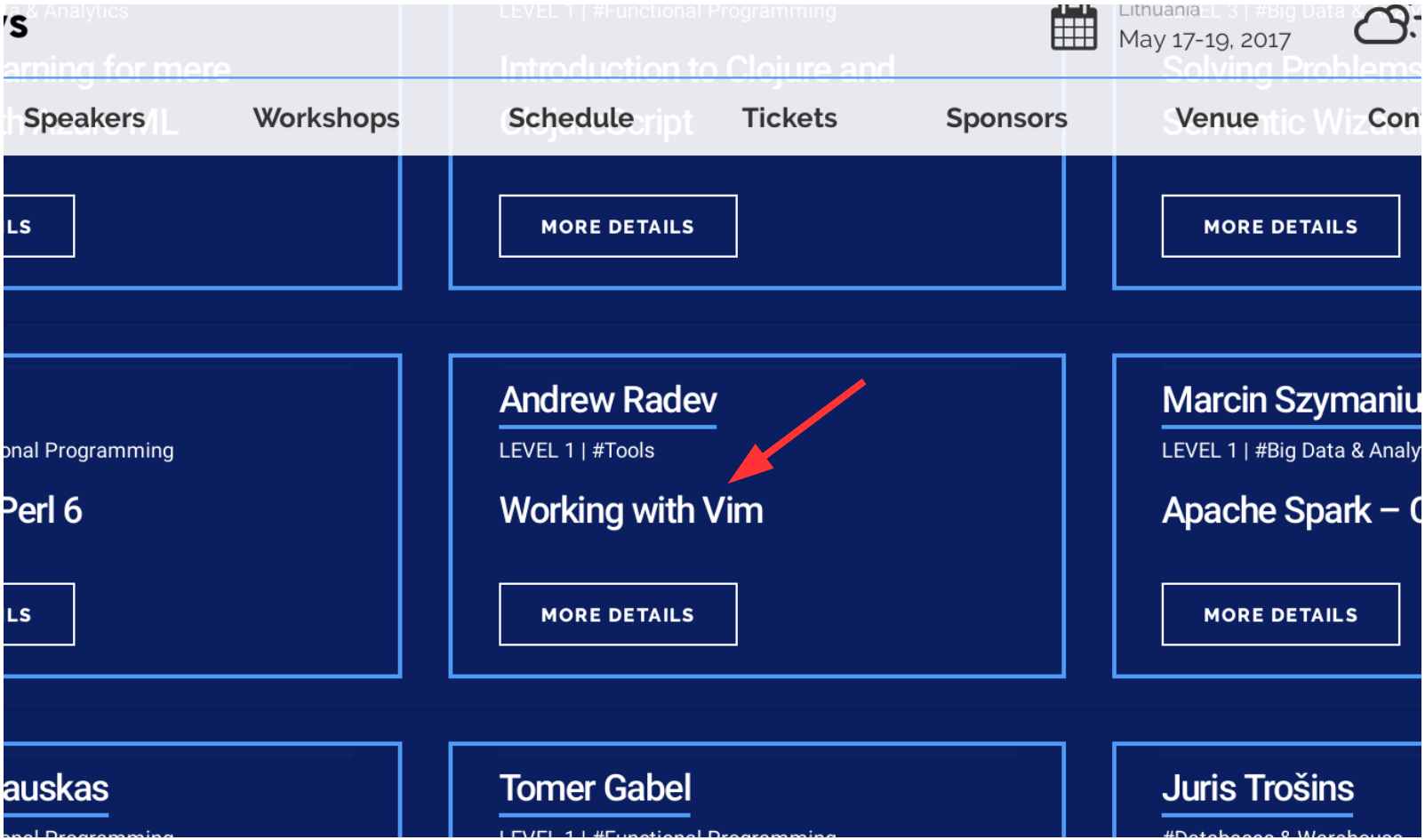

/THEORY/IN/PRACTICE
Beautiful Testing
Leading Professionals Reveal How They Improve Software
O'REILLY
Tim Riley & Adam Goucher

Beautiful testing

“Coding errors are likely to cause the tests to fail every time, not just a little more often than expected.”

– John D. Cook

Text Editors


Speakers
Workshops
Schedule
Tickets
Sponsors
Venue
May 17-19, 2017
MORE DETAILS
MORE DETAILS
Andrew Radev
LEVEL 1 | #Tools
Working with Vim
MORE DETAILS
Marcin Szymaniu
LEVEL 1 | #Big Data & Analy
Apache Spark –
MORE DETAILS
Tomer Gabel
Juris Trošins

Andrew Radev

AndrewRadev

Don't be fooled by all the Vimscript, I write Ruby for a living.

Sofia, Bulgaria

andrey.radev@gmail.com

http://andrewradev.com/

Organizations

Overview | Repositories 76 | Stars 372 | Followers 270 | Following 6

Search repositories…

Type: All | Language: Vim s

37 results for repositories written in **Vim script**

splitjoin.vim

A vim plugin that simplifies the transition between multiline and single-line code

Vim script ★ 600 46 Updated 3 days ago

linediff.vim

A vim plugin to perform diffs on blocks of code

refactoring vim merge-conflicts diff

Vim script ★ 164 20 Updated 12 days ago

deleft.vim

Delete a wrapping if-clause, try-catch block, etc. and shift left.

Vim script ★ 5 1 Updated 14 days ago

Text Editors


Vim

Splitjoin
(demo)

```
vim --servername FOO

vim --remote-send
vim --remote-expr
```

Vimrunner

(demo)

```ruby
describe "ruby" do

  specify "if-clauses" do
    # Setup
    set_file_contents 'test.rb', <<~EOF
      return "the answer" if 6 * 9 == 42
    EOF
    vim.command 'edit test.rb'

    # Exercise
    vim.command 'SplitjoinSplit'
    vim.write

    # Verify
    expect_file_contents 'test.rb', <<~EOF
      if 6 * 9 == 42
        return "the answer"
      end
    EOF
  end

end
```

Big idea:
Sometimes, it’s not *that* tricky

```ruby
describe Stack do
  it "allows popping the top element" do
    # Setup
    stack = Stack.new
    stack.push("one")
    stack.push("two")

    # Exercise
    top_element = stack.pop

    # Verify
    expect(top_element).to eq "two"
  end
end
```

```ruby
describe "ruby" do

  specify "if-clauses" do
    # Setup
    set_file_contents 'test.rb', <<~EOF
      return "the answer" if 6 * 9 == 42
    EOF
    vim.command 'edit test.rb'

    # Exercise
    vim.command 'SplitjoinSplit'
    vim.write

    # Verify
    expect_file_contents 'test.rb', <<~EOF
      if 6 * 9 == 42
        return "the answer"
      end
    EOF
  end

end
```

```ruby
module WaitingOnRails
  describe Rake do
    let(:player)    { Player.new('test.mp3') }
    let(:runner)    { Rake.new(player) }
    let(:rake_stub) { Support::CommandStub.new('rake') }

    it "plays music during the entire running of the command" do
      # Exercise
      thread = Thread.new { runner.run(['routes']) }

      # Setup
      rake_stub.init

      # Verify
      expect(player.pid).to be_a_running_process
    end
  end
end
```

Big idea:
Test code is code

Test code is code

- Vimrunner
- CommandStub
- Selenium


The Architecture of
Open Source Applications
Elegance, Evolution, and a Few Fearless Hacks
Edited by Amy Brown & Greg Wilson

What would the authors of Selenium do?

Thank you

Further reading

- Beautiful Testing
- Beautiful Testing chapter 10 (PDF)
- The Architecture of Open Source Applications
- The Architecture of Open Source Applications (Selenium Chapter)
- Javascript Testing Recipes