Automated Testing Challenges

(a practical approach)

Duarte Henriques, CTO@Seedrs

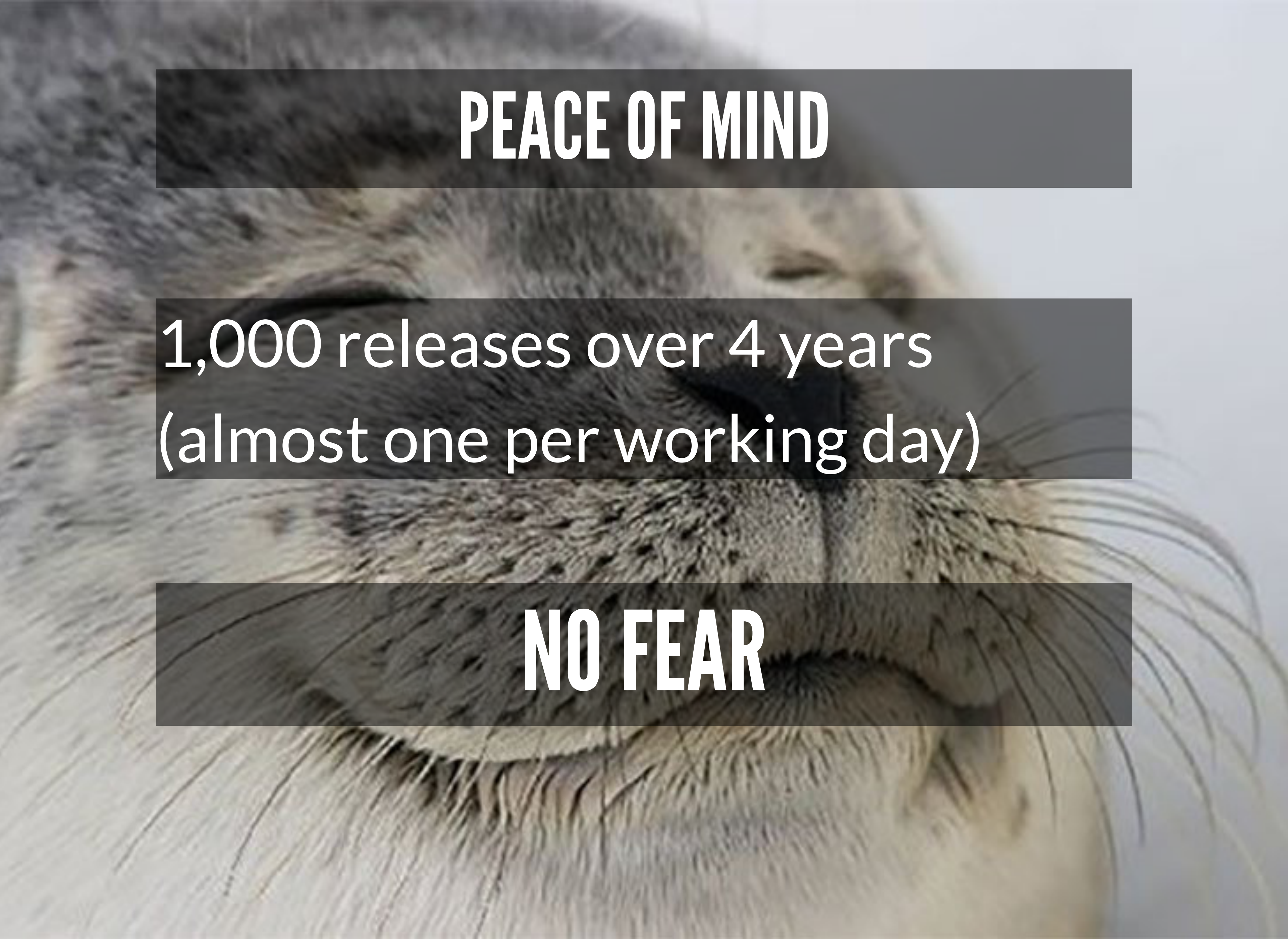
THE MONOLITH

```
~/workspace/seedrs$ bin/rake stats | tail -n2
  Code LOC: 36604     Test LOC: 49799     Code to Test Ratio: 1:1.4
```

700 cucumber scenarios

5,000 rspec examples

45 minutes to run (single cpu)



PEACE OF MIND

1,000 releases over 4 years
(almost one per working day)

NO FEAR

TESTING STACK - RSPEC

It's just another testing framework...

tenderlovemaking.com/2015/01/23/my-experience-with-minitest-and-rspec.html

TESTING STACK - CUCUMBER

```
Feature: Users can sign in

  Scenario: Sign in with password
    Given I am registered with "jimbo@mail.com", "the password"
    When I sign in with "jimbo@mail.com", "the password"
    Then I should be signed in
```

```
When(/^I sign in with "(\S+)", "(.*)"$/) do |email, password|
  sign_in_via_form(email, password)
end
```

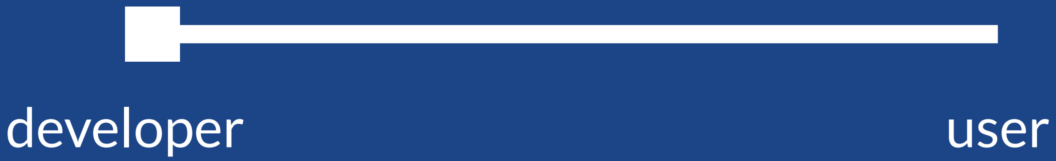
TESTING STACK - CAPYBARA

```ruby
def sign_in_via_form(email, password)
  visit(sign_in_path)
  fill_in("session_email", :with => email)
  fill_in("session_password", :with => password)
  click_button("Sign in")
end
```

WHITEBOX | BLACKBOX

- the subject of the test
- the 'real' context
- the stubbed context

It's a slider



DEVELOPER | USER

For any new feature developed

- lots of developer-centric tests
- a few user-centric tests

For any bug found in production

- regression test - user-centric!

1. DESERT TESTING

Run tests locally - Internet shouldn't be needed

- connection could go down, causing flakyness
- you can be blocked for over-usage
- it slows down tests

1. DESERT TESTING

gems such as webmock, fakeweb, etc

```
%html
  %head
    - unless Rails.env.test?
      = javascript_include_tag("//use.typekit.com/gmd7txr.js")
      = javascript_tag("try{Typekit.load();}catch(e){}")
```

2. THIRD-PARTY EVENTS

```ruby
def before_customer_io
  @customer_io_events = []
  allow(CustomerIo).to receive(:track) do |user, event_name, event_attributes|
    @customer_io_events << [user, event_name, event_attributes]
  end
end

def after_customer_io
  @customer_io_events = nil
  allow(CustomerIo).to receive(:track).and_call_original
end
```

2. THIRD-PARTY EVENTS

```
Feature: Deposits expire

  @customer-io
  Scenario: User receives email when his deposit is about to expire
    Given a deposit exists with valid_till: 9 days from now
    When the daily maintenance tasks are run
    Then 1 customerio event with name: "deposit_about_to_expire" should have been sent
```

3. AJAX

Simple problem:

```
Feature: Flaky web-app navigation
  Given I am on the homepage
  When I follow "foo"
  And I follow "bar" # flaky error!
```

"Simple" solution:

```
Feature: Stable web-app navigation
  Given I am on the homepage
  When I follow "foo"
  Then I should see "bar" # waits for content
  When I follow "bar"
```

3. AJAX

Contrived Problem:

```
Feature: Flaky web-app navigation
  Given I am on the homepage
  When I follow "foo"
  Then a foo should exist # flaky error!
```

Solution:

```
Feature: Flaky web-app navigation
  Given I am on the homepage
  When I follow "foo"
  Given I wait for the ajax request to finish
  Then a foo should exist # flaky error!
```

3. AJAX

Another example:

```
@javascript
Scenario: Something that triggers ajax requests that we don't care about
  Given some context
  When I do something
  Then this should happen

  # This test fires ajax requests that we don't otherwise wait for, so the
  # database is cleaned while the server tries to write to it, causing errors
  # the errors will only appear when other tests are already running.
  # This line fixes it:
  And I wait for the ajax request to finish
```

3. AJAX

in javascript:

```
// adapted from: https://gist.github.com/424127
window.runningAjaxCalls = 0;

jQuery(function($) {
  var originalAjax = $.ajax;

  var countDown = function(callback) {
    return function() { // would also handle exceptions
      callback.apply(this, arguments);
      window.runningAjaxCalls -= 1;
    };
  };
  var ajaxWithCount = function(url, options) {
    window.runningAjaxCalls += 1;
    options.success = countDown(options.success);
    options.error = countDown(options.error);
    return originalAjax(url, options);
  };

  $.ajax = ajaxWithCount;
});
```

3. AJAX

in ruby:

```ruby
# adapted from https://gist.github.com/424127
def wait_for_ajax_requests
  loop do
    sleep 1
    break if page.evaluate_script("window.runningAjaxCalls").to_i == 0
  end
end
```

in cucumber:

```ruby
When(/^I wait for the ajax requests? to finish$/) do
  wait_for_ajax_requests
end
```

4. PERFORMANCE

Sometimes N+1 queries creep up

4. PERFORMANCE

```
it "closing a campaign is not affected by N+1 queries on investments" do
  campaign = create(:approved_campaign)
  11.times{ create(:investment, :campaign => campaign) }

  expect{ campaign.close_with_success }.not_to exceed_query_limit(10)
end
```

4. PERFORMANCE

```ruby
RSpec::Matchers.define(:exceed_query_limit) do |expected|
  match do |block|
    query_count(&block) > expected
  end

  failure_message_for_should_not do |_actual|
    "Expected to run maximum #{expected} queries, got #{@counter.query_count}"
  end

  def query_count(&block)
    @counter = ActiveRecord::QueryCounter.new
    ActiveSupport::Notifications.subscribed(@counter.to_proc,
                                            "sql.active_record",
                                            &block)
    @counter.query_count
  end

  def supports_block_expectations?
    true
  end
end
```

4. PERFORMANCE

```ruby
module ActiveRecord
  class QueryCounter
    attr_reader :query_count

    def initialize
      @query_count = 0
    end

    def callback(_s, _start, _finish, _message_id, values)
      unless query_to_ignore?(values)
        @query_count += 1
        puts "#{@query_count}: #{query_desc(values)}" if verbose?
      end
    end

    def to_proc
      lambda(&method(:callback))
    end

    private
    def query_to_ignore?(values) # ...
    def query_desc(values) # ...
    def verbose? # ...
  end
```

5. CONCURRENCY

fork

5. CONCURRENCY

```ruby
def make_concurrent_calls(count: 2)
  ActiveRecord::Base.connection.disconnect!

  Array.new(count) do |i|
    pid = Process.fork do
      $stderr.reopen(File.new(File::NULL, "w"))
      $stdout.reopen(File.new(File::NULL, "w"))
      ActiveRecord::Base.establish_connection
      yield i
    end
    Process.wait(pid)
  end

  ActiveRecord::Base.establish_connection
end
```

5. CONCURRENCY

```ruby
it "only creates one funds movement when confirming deposit concurrently"
  deposit = Deposit.new

  make_concurrent_calls do
    deposit.confirm!
  end

  expect(Movement.count).to eq(1)
end
```

5. CONCURRENCY

forkbreak: fork + breakpoints

5. CONCURRENCY

```
def run_with_breakpoints(*execution_blocks)
  processes = execution_blocks.map do |block|
    ForkBreak::Process.new do |breakpoints|
      $stderr.reopen(File.new(File::NULL, "w"))
      $stdout.reopen(File.new(File::NULL, "w"))
      ActiveRecord::Base.establish_connection
      block.call(breakpoints)
    end
  end

  ActiveRecord::Base.connection.disconnect!
  yield(*processes)
  ActiveRecord::Base.establish_connection
end
```

5. CONCURRENCY

```ruby
investment = Investment.new

block1 = lambda do |breakpoints|
  add_breakpoint(breakpoints, investment, :before_cancel)
  investment.cancel
end

block2 = lambda do |breakpoints|
  add_breakpoint(breakpoints, investment, :after_process)
  investment.process
end

breakpoint_names = [:before_cancel, :after_process]

run_with_breakpoints(block1, block2) do |*execution_processes|

  execution_processes.each_with_index do |execution_process, index|
    execution_process.run_until(breakpoint_names[index]).wait
  end

  execution_processes.each do |execution_process|
    execution_process.finish.wait
  end
```

5. CONCURRENCY

```
# add_breakpoint(breakpoints, investment, :before_cancel)

def add_breakpoint(breakpoints, object, breakpoint_name)

  flow, method_name = breakpoint_name.to_s.split(/_/, 2).map(&:to_sym)

  original_method = object.method(method_name)

  if flow == :before
    allow(object).to receive(method_name) do |*args|
      breakpoints << breakpoint_name
      original_method.call(*args)
    end

  elsif flow == :after
    allow(object).to receive(method_name) do |*args|
      value = original_method.call(*args)
      breakpoints << breakpoint_name
      value
    end
  end
end
```

SEEDRS

Thank you

Duarte Henriques, CTO@Seedrs