Adam Piotrowski

Formerly CTO at 2N IT, currently demoted to the position of CEO

e-mail: ap@2n.pl
Twitter: @pan_sarin
Instagram: @adamsarin

Mutation testing

Adam Piotrowski
15.02.2023

Mutation testing - why & how - case study

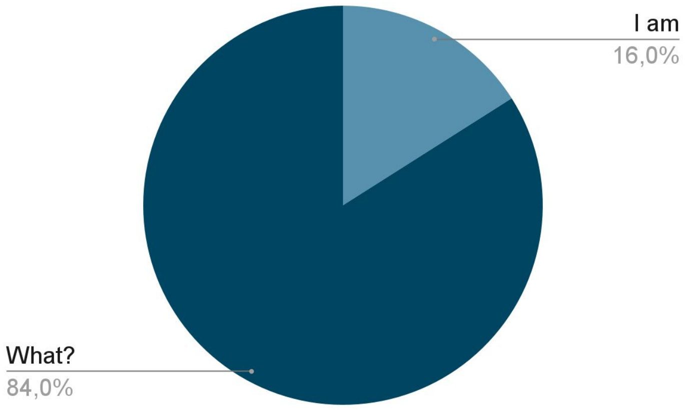
Why this presentation?

Are you testing your code?

On one of the meet.js I heard
“We only test if client allows us to do that”.

Why this presentation?

Who knows what mutation testing is?



Pool from 91000 ruby devs group on LI

Why this presentation?

Concept is well known for long time, and all the most important languages/frameworks have tools for that.

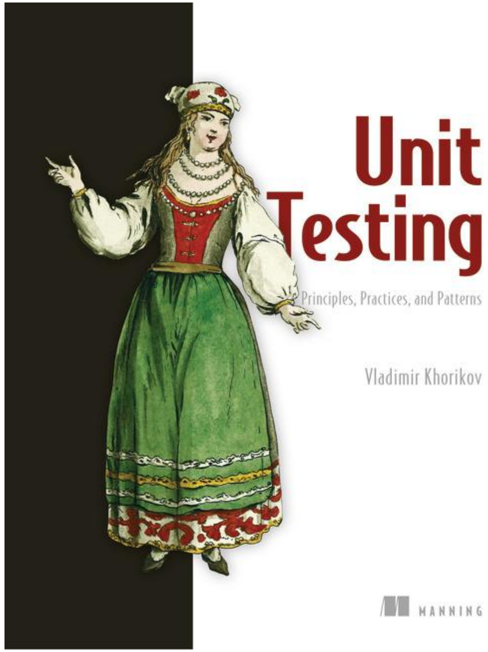
But even though we are in 2022 and we have those tools, most of people don’t use them and there are still a lot of people who have doubts if testing itself is a good idea.

<place proper meme there - probably the one with facepalm>

Why this presentation?

Why this presentation?

Why this presentation?



Why are we testing? What is the purpose?

Why are we testing? What is the purpose?

General:

- To avoid regression

Why are we testing? What is the purpose?

General:

- To avoid regression
- To have documentation

Why are we testing? What is the purpose?

General:

- To avoid regression
- To have documentation
- To think about what and how we are about to implement *(that is more of a positive side effect than a goal itself)*

Why are we testing? What is the purpose?

General:

- To avoid regression
- To have documentation
- To think about what and how we are about to implement *(that is more of a positive side effect than a goal itself)*
- To be able to refactor without risk / fear.

Why are we testing? What is the purpose?

“The goal is to enable sustainable growth of the software project.

The term sustainable is key.”

Vladimir Khorikov

How we measure how good are our tests?

How we measure how good are our tests?

How we measure how good are our tests?

How we measure how good are our tests?

RELIABILITY

How we measure how good are our tests?

SCALABILITY

How we measure how good are our tests?

Team rotation is the biggest issue
CHANGE MY MIND
KAPWING

How we measure how good are our tests?

- Reliability
- Scalability
- Mental stability - or at least lack of constant fear

How we measure how good are our tests?

Allow me to quote Vladimir Khorikov once more, since his summary of chapter “The goal of unit testing” is great fulfillment of our chapter:

“A successful test suite has the following properties:

- It’s **integrated** into the development cycle.
- It targets **only the most important parts** of your code base.
- It provides maximum value with minimum maintenance costs.”

Question

Who is familiar / experienced
with
line test coverage metric?

Question

Who is familiar / experienced with mutation testing?

Mutants

How we measure code coverage?

And that is the **core** of that presentation.

Something that made me think that whole IT industry is a scam when I realized how many companies are bragging about their 100% test coverage.

How we measure code coverage? And what for?

Time for code.

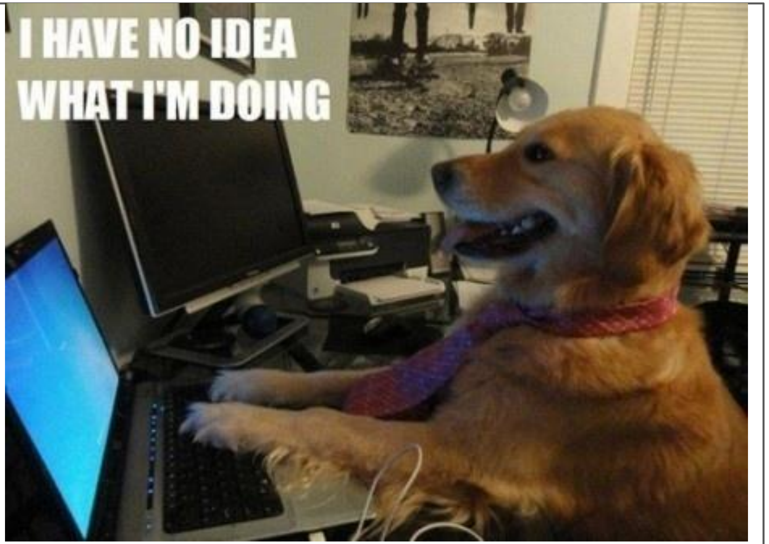
Service and its tests

https://gist.github.com/panSarin/03bd5929c9540389561e7a3d9fa59f35

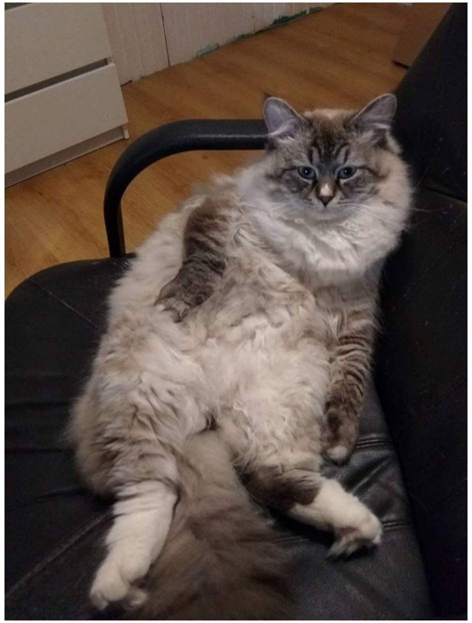
SimpleCov results and how it works

```
app/services/api/users/create.rb      100.00%
```

“false positive” - rings a bell?



False negatives/positives dilemma.



False negatives/positives dilemma.

False negatives/positives dilemma.

False negatives/positives dilemma.

False negatives/positives dilemma.

False negatives/positive.

	Predicted true	Predicted false
Actual true	True positive	False negative
Actual false	False positive	True Negative

SimpleCov results and how it works

app/services/api/users/create.rb 100.00%

Before I explain how it works - numbers

Numbers from mutant ran for the same service.

719 mutations

26 alive

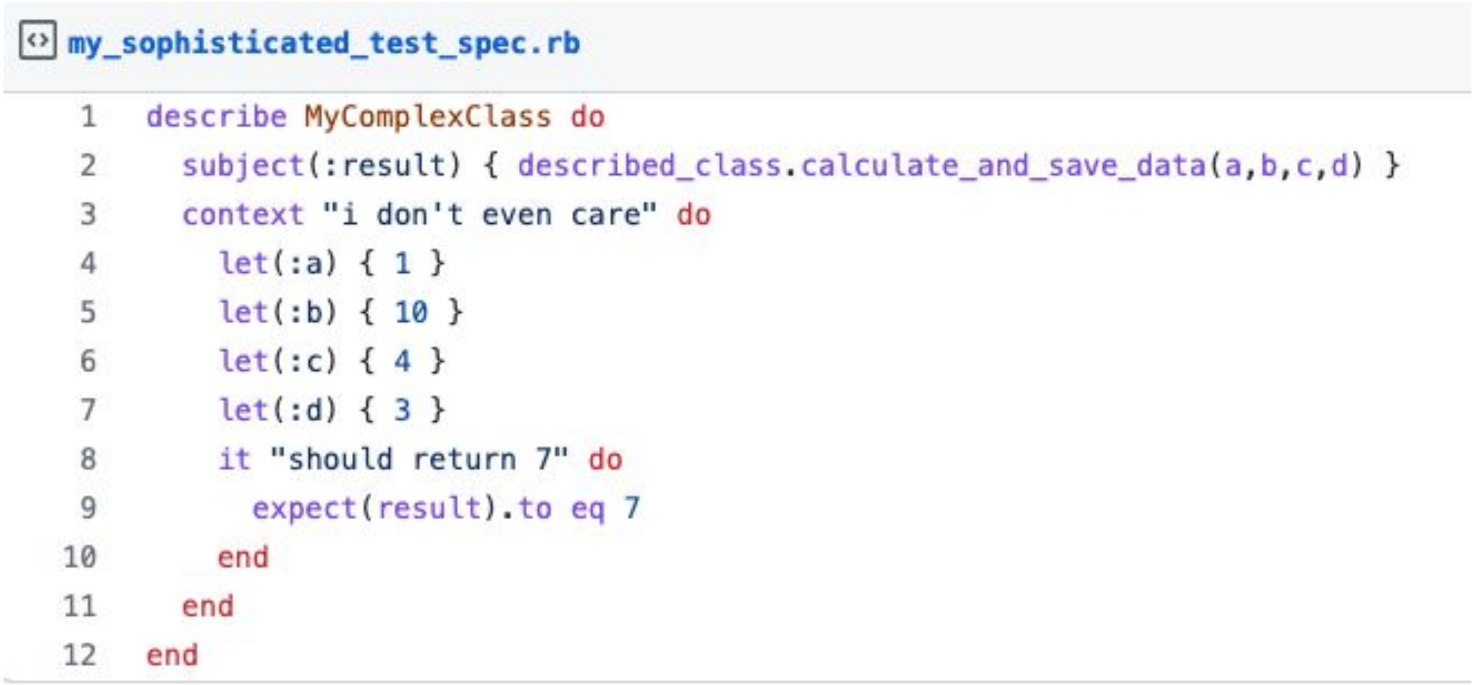
96.35% coverage

How line test coverage works

SimpleCov like most of the tools designed to measure “line test coverage” are really simple or even “ordinary” algorithm.

It runs all of your tests and checks how many lines of your code it ran.
So imagine that:

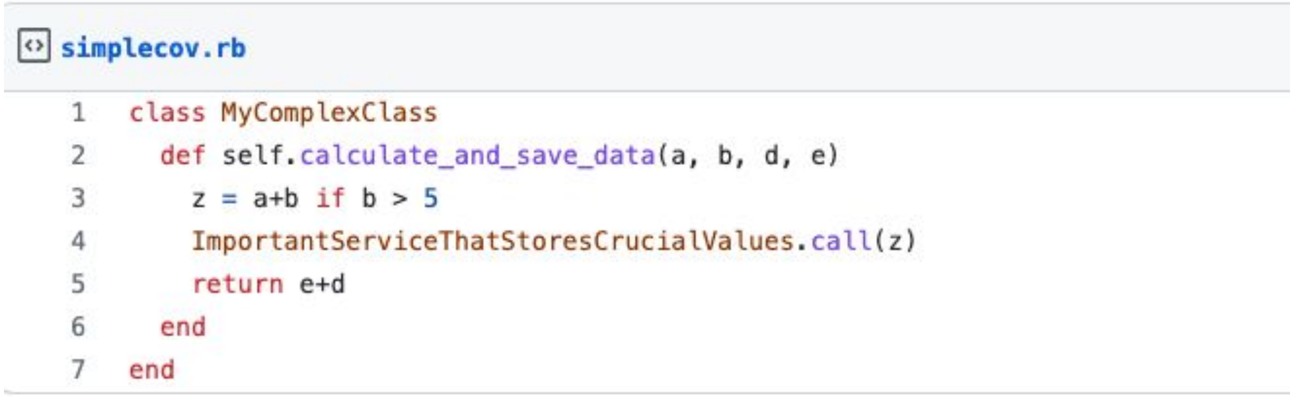
100% vs 26 alive mutants



my_sophisticated_test_spec.rb

```ruby
describe MyComplexClass do
  subject(:result) { described_class.calculate_and_save_data(a,b,c,d) }
  context "i don't even care" do
    let(:a) { 1 }
    let(:b) { 10 }
    let(:c) { 4 }
    let(:d) { 3 }
    it "should return 7" do
      expect(result).to eq 7
    end
  end
end
```

100% vs 26 alive mutants



simplecov.rb

```ruby
class MyComplexClass
  def self.calculate_and_save_data(a, b, d, e)
    z = a+b if b > 5
    ImportantServiceThatStoresCrucialValues.call(z)
    return e+d
  end
end
```

100% vs 26 alive mutants

“Our application is covered by tests in 100%”

app/services/my_complex_class.rb 100%

(oh and our team is young and energetic)

High line test coverage

False sense of having well-tested application can be more dangerous than working with an application that you are aware is not immune for regressions.

Of course it is not like SimpleCov and other line-test-coverage tools are useless and bad - but it is crucial to remember that:

If you have low line-test-coverage, that means you have to improve your tests.
If you have high line-test-coverage, that means nothing.

High line test coverage

Let me quote Vladimir Khorikov again:

“Low coverage numbers - are a certain sign of trouble. They mean there’s a lot of untested code in your code base. But high numbers don’t mean anything. Thus, measuring the code coverage should be only a first step on the way to a quality test suite.”

How mutation testing works

26 alive

26 alive mutations == 26 logical changes in your code, that won’t trigger any failing test.
So there are 26 possible “reasonable” changes in your code, that you didn’t think about and your tool designed to make you feel secure didn’t identify.

Few quotes from @mbj

“Mutation coverage forces you to proof every semantic effect of your code is covered. And usually you have to remove code instead of adding new test. Too many people add ‘just in case code’. “

```
def foo(bar)
  something(bar.to_i)
end
```

“#to_i - is a semantic effect. But why not just require its and Integer in the first place at all call sides? Instead of write a test that shows you coerce here?”

Mutations - case study

Mutations - case study

```diff
-  limit = Setting.player_code_limit || 4
+  limit = self
-  limit = Setting.player_code_limit || 4
+  limit = 4
-  limit = Setting.player_code_limit || 4
+  limit = nil || 4
-  limit = Setting.player_code_limit || 4
+  limit = !Setting.player_code_limit || 4
-  limit = Setting.player_code_limit || 4
+  limit = Setting.player_code_limit || 0
-  limit = Setting.player_code_limit || 4
+  limit = Setting.player_code_limit || 1
-  limit = Setting.player_code_limit || 4
+  limit = Setting.player_code_limit || self
-  limit = Setting.player_code_limit || 4
+  limit = Setting.player_code_limit || 3
```

Mutations - case study

```
it "assigns a correct number of uses to created invite code" do
  result
  expect(User.last.invite_codes.first.uses_left).to eq(4)
end
```

Mutations - case study

```
  ActiveRecord::Base.transaction {
    user = User.create(user_params)
    Address.create(address_params.merge(addressable: user))
    activate_invite_code(user.id)
    create_invite_code(user)
    Api::InviteCodes::GenerateClubCodes.new(user_id: user).call
    change_password(user, params[:password])
    user.generate_activation_code
-   user.update(token: SecureRandom.uuid)
+   nil
    user.token
  }
```

Mutations - case study

```
-  Success(true)
+  Success(false)

-  Success(true)
+  Success()
```

Mutations - case study

```
-  if params[:invite_code] == Rails.application.config.password_data[:invite]
+  if params[:invite_code].equal?(Rails.application.config.password_data[:invite])

-  if params[:invite_code] == Rails.application.config.password_data[:invite]
+  if params[:invite_code] == Rails.application.config.password_data.fetch(:invite)
```

Mutations - case study

https://blog.arkency.com/which-one-to-use-eql-vs-equals-vs-double-equal-mutant-driven-developpment-for-country-value-object/

Mutations - case study

```
-  if repost.nil?
-    return Failure(:invalid_update)
-  end
+  self

-    return Failure(:invalid_update)
+    return nil
-    return Failure(:invalid_update)
+    Failure(:invalid_update)
-    return Failure(:invalid_update)
+    return Failure()
-    return Failure(:invalid_update)
+    return Failure(:invalid_update__mutant__)
```

Mutations - case study

https://github.com/RailsEventStore/ecommerce/pull/96/files

Conclusions

Mutant can:

Conclusions

Mutant can:

- give you hints about what code is useless

Conclusions

Mutant can:

- give you hints about what code is useless
- give you hints about what you forgot to test

Conclusions

Mutant can:

- give you hints about what code is useless
- give you hints about what you forgot to test
- force you to think about other ways to implement some stuff

Conclusions

Mutant can:

- give you hints about what code is useless
- give you hints about what you forgot to test
- force you to think about other ways to implement some stuff
- give you hints if your Unit is too complex

Conclusions

Mutant can:

- give you hints about what code is useless
- give you hints about what you forgot to test
- force you to think about other ways to implement some stuff
- give you hints if your Unit is too complex
- be your code-review partner

Conclusions

Mutant can't:

Conclusions

Mutant can’t:

- take **responsibility** for your code

Conclusions about devs

I WANT MY PROJECT TO BE GREAT
I WILL MAKE MY PROJECT GREAT
imgflip.com

Mutant implementation

What mutations really are?

(If you want more theory: https://troessner.github.io/articles/2016-08-02-how-does-mutant-work.html)

Mutation testing tools for various languages

https://stryker-mutator.io/ - JS & C#

https://github.com/mbj/mutant - ruby

https://mutatest.readthedocs.io/en/latest/ - python

https://infection.github.io/guide/ - php

https://pitest.org/ - java

Q&A

e-mail: ap@2n.pl

Twitter: @pan_sarin

Instagram: @adamsarin

linkedin: https://www.linkedin.com/in/adamsarin/

Sources / links

- https://github.com/mbj/mutant
- https://www.manning.com/books/unit-testing?query=unit%20testing%20prin
- https://troessner.github.io/articles/2016-08-02-how-does-mutant-work.html
- https://blog.arkency.com/which-one-to-use-eql-vs-equals-vs-double-equal-mutant-driven-developpment-for-country-value-object/
- https://docs.google.com/document/d/1ffvST0wMLaQdZfuDyQO2H12ktRYV4Al7yICNqcyedpk/edit?usp=sharing
- https://gist.github.com/panSarin/03bd5929c9540389561e7a3d9fa59f35
- https://gist.github.com/PavloJunior/7ae3d9897722e2531f30d131c1272bb1