Can you trust your tests?



Vilnius RB 2015 Tadas Ščerbinskas & Vaidas Pilkauskas

Tadas Ščerbinskas @tadassce

Vaidas Pilkauskas @liucijus

Agenda

- Test code quality
- Mutation testing
- Mutant mutation testing for Ruby

Prod vs. Test code quality

Prod vs. Test code quality

Code has bugs. Tests are code. Tests have bugs. MATCHES WATCHEN THE ATCHMENT



Test Quality

- Readable
- Focused
- Concise
- Well named

"Program testing can be used to show the presence of bugs, but never to show their absence!"

- Edsger W. Dijkstra



Types of code coverage

- Line
- Branch
- Condition
- Path
- Data

Coverage tool for Ruby

simplecov

```
gem install simplecov
```

```
require 'simplecov'
SimpleCov.start
```

Line

```
def foo(arg = true)
   arg ? "a" : "b"
end

expect(foo).to eq "a"
```

100% line coverage. No test for "b".

Branch

```
def foo(arg)
  arg ? "a" : "b"
end

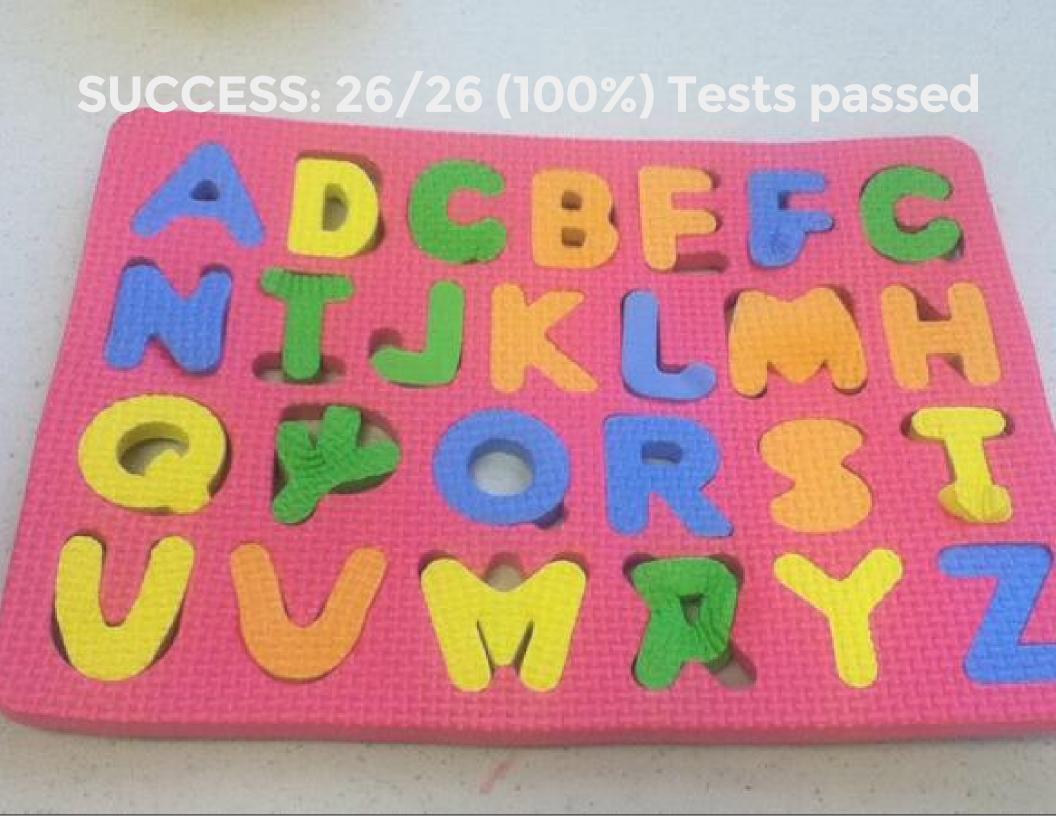
expect(foo(true)).to eq "a"
  expect(foo(false)).to eq "b"
```

100% branch coverage

Demo

Can you trust 100% coverage?

- Code coverage can only show what is not tested
- 100% code coverage for interpreted languages is kind of like full compilation





Mutation testing

Changes your program code and expects your tests to fail.

Terminology

Applying a mutation to some code creates a *mutant*.

If test passes - mutant has *survived*.

If test fails - mutant is *killed*.

Failing is the new passing

Tests' effectiveness is measured by number of killed mutants by your test suite.



What if mutant survives

- Simplify your code
- Add additional tests

 TDD - minimal amount of code to pass the test

Hunting tips

```
# Avoid literals
posts[0] => posts.first

# Don't overuse syntactic constructs
::User => User

# Don't pass literal defaults
num.to_s(10) => num.to_s
```

Challenges

- High computation cost slow
- Equivalent mutants false negatives
- Infinite loops

Equivalent mutations

```
# Original
i = 0
while i != 10
  do_something
  i += 1
end
```

```
# Mutant
i = 0
while i < 10
   do_something
   i += 1
end</pre>
```

Infinite Runtime

```
# Original
while expression
  do_something
end
```

```
# Mutation
while true
  do_something
end
```

Use timeouts

```
config.around(:each) do |example|
  Timeout.timeout(2) do
    example.run
  end
end
```

Mutant

- Active project
- Good reporting
- Only works with **rspec**

How

Can't just do String#gsub

How

Mutant uses a pure Ruby *parser* and an *unparser* to do its magic.

AST:

```
p Parser::CurrentRuby.parse("2 + 2")
# (send
# (int 2) :+
# (int 2))
```

Mutations

- Literal / primitive and compound
- Statement deletion
- Conditional
- Binary connective replacement
- Argument deletion / rename / swap
- Unary operator exchange
- Bitwise

Test-Selection

"Longest rspec example group descriptions' prefix match"

Test-Selection

```
$ mutant --include lib --require bar --use rspec Foo
```

- 1. Foo::Bar#baz
- 2. Foo::Bar
- 3. Foo

Demo

Disadvantages

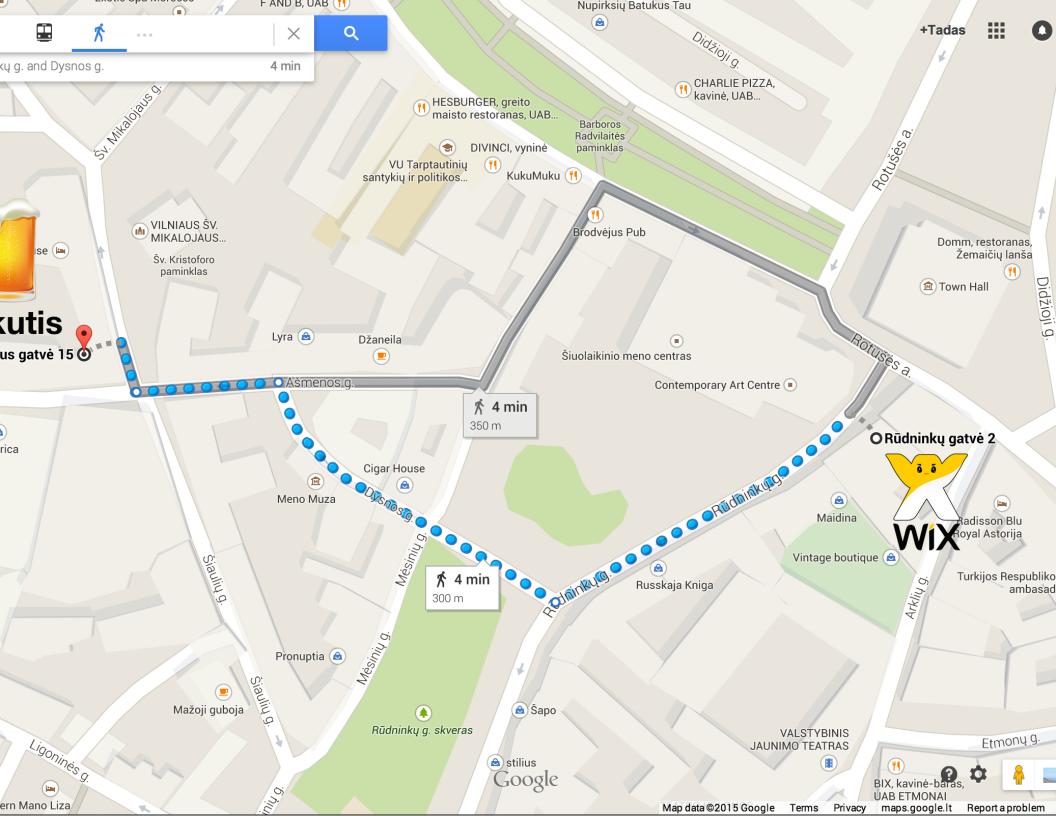
- Slow cannot be part of TDD rhythm
- May be very noisy

Summary

- Normal code coverage highlights code that is definitely *not* tested
- Mutation testing highlights code that definitely *is* tested

Thanks!

Q&A



Pictures

- s05 http://crazy-monkeeey.deviantart.com/art/Who-watches-the-Watchmen-344523349
- s08 http://www.chimpsanctuarynw.org/blog/wp-content/uploads/2009/03/negra-covered-in-pink-blanket-front-room_web_mg_8161.jpg
- s15 https://twitter.com/bloerwald/status/448415935926255618
- s16 http://www.smosh.com/smosh-pit/photos/12-more-bizarre-mutant-animals
- s21 https://tbgsecurity.com/the-history-of-hacking-timeline-of-hacking-techniques-infographic/