Mutation testing

Andy Zaidman

This talk is about…

Not really…

Software Failures

Many software failures each year…

A 2002 estimate: software failures cost the US economy $59.5 billion

Tassey, G. (2002). The economic impacts of inadequate infrastructure for software testing. Technical report, National Institute of Standards and Technology RTI Project.

Many software failures each year…

Software failure caused $1.7 trillion in financial losses in 2017

https://www.techrepublic.com/article/report-software-failure-caused-1-7-trillion-in-financial-losses-in-2017/

Many software failures each year…

2002

15 years

2017

$59 500 000 000

x 28

$1 700 000 000 000

What is the problem?

1. Do developers test (enough)?
2. Do developers test well?

nl.tudelft.watchdog.logic.interval.IntervalToJsonConverterTest
testJsonSessionIntervalRepresentation (0.052 s)
testJsonReadingIntervalRepresentation (0.003 s)
testJsonTypingIntervalRepresentation (0.002 s)
testCreateReadInterval (0.004 s)
testWritingIntervalsGetClosedOnHigherCancel (0.00
testCreateReadIntervalOnlyOnce (0.006 s)

Research Tool

"Big Brother" in your IDE
→ measures testing activities

- > 2 400 SW engineers, 118 countries
- Java / Eclipse / Android Studio
- Some SW engineers observed for 2.5 years

Observation 1

- In > 50% of the projects that participated in the WatchDog study
  - No test activities in > 3 month observation period
- For 47% of developers who indicated to do testing, we found no testing activities for at least a 3 month observation period

Observation 2

Time spent on testing…

If we ask those SW engineers…

Can you estimate how much time you spend on engineering **test code** versus **production code?**

50% test code

50% production code

30% test code

70% production code

WatchDog Pre-Test Questionnaire

Can you estimate how much time you spend on engineering **test code** versus **production code?**

48% - 52%

test code production code

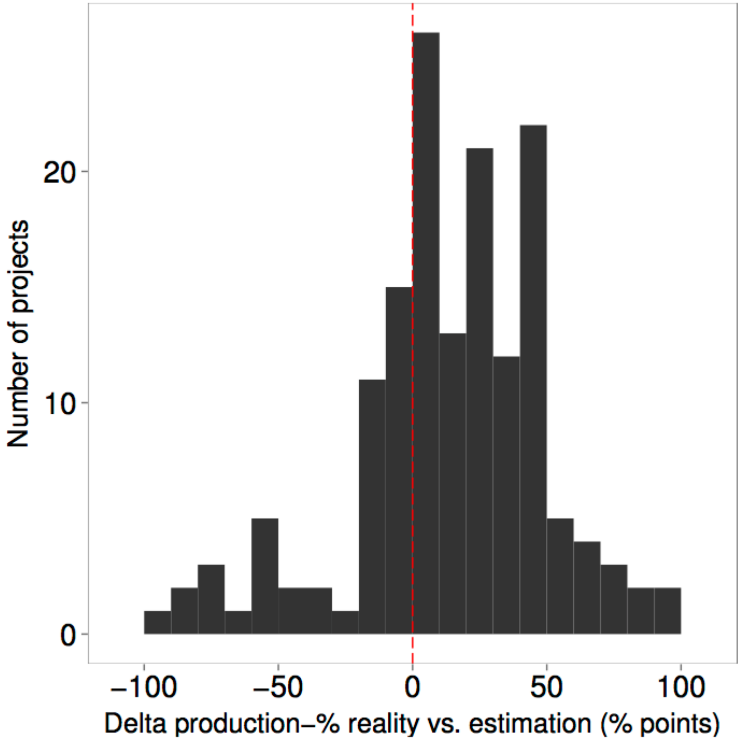
After measuring ≥ 3 months

25% - 75%

test code engineering

production code engineering

Most participants overestimate their test engineering activities

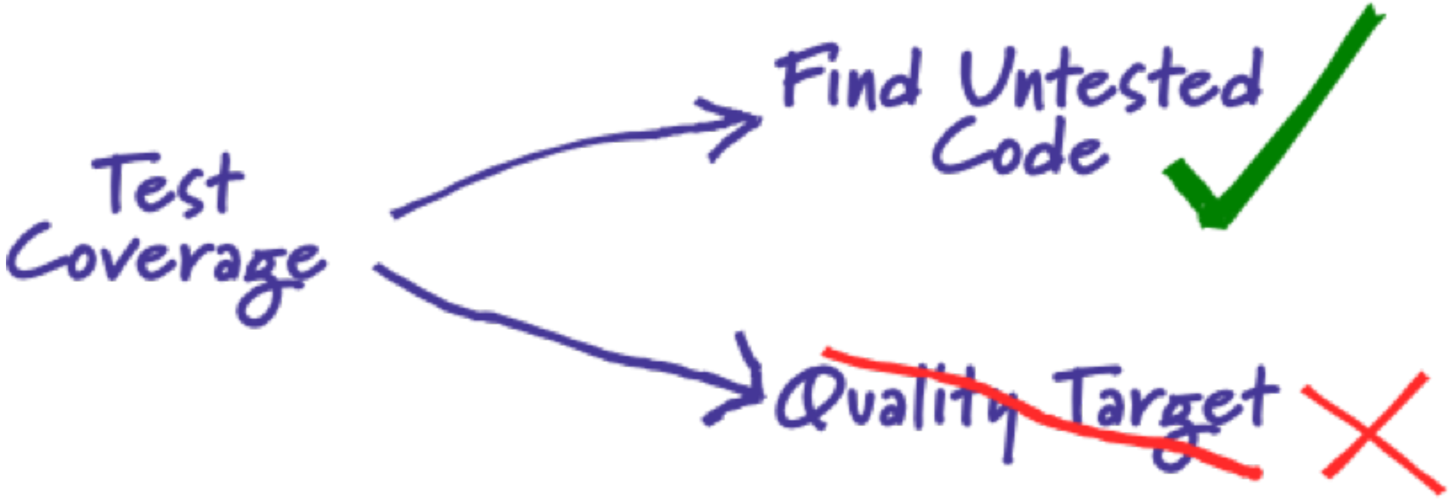


So what, testing is overestimated?

- SW engineers tend to:
    - Underestimate difficult tasks
    - Overestimate easy tasks
- As SW often dislike testing and consider it "tedious", they might overestimate testing.

How to check whether we’ve tested it all?

- Test coverage



- Production code can be covered, yet the tests covering it might still miss a bug (i.e., the tests are not of sufficient quality)
  → **code coverage can give you a false sense of security**

- Is there another way of looking into the quality of tests?

Side note: psychology of testing

- People *mostly* test positive scenarios
  - Confirmation bias: people test for what they expect, not for the unexpected things…

- Example:
  - Positive: A user logs into a website using the correct credentials
  - Negative: A user tries to log into a website using wrong credentials
    → **is that scenario also covered in a test?**

Gül Çalikli, Ayse Basar Bener:
Influence of confirmation biases of developers on software quality: an empirical study.
Software Quality Journal 21(2): 377-416 (2013)

Mutation testing by example

Original

```
if( i >= 0 ) {
    return "foo";
} else {
    return "bar";
}
```

Test

🟩

Code is transformed, mutant introduced

Tests remain identical

Mutant

```
if( i < 0 ) {
    return "foo";
} else {
    return "bar";
}
```

Test

Scenario 1 🟩

→ Mutant alive

Scenario 2 🟥

→ Mutant killed

Mutation testing…

- Is not new!
- Origins in the 1970s already!

= a systematic way to seed faults in your program

A mutant is killed if the test ***fails***

What about fault injection?

- With fault detection you test your code
  - Inject faults/mutations to see how your **system** reacts

- With mutation tests your test your tests
  - Inject faults/mutations and see how the **tests** react

So…

- Invented in the 1970s
- Seemingly a great + intuitive technology

- Why isn’t everyone using it?

Well

- First, you have to test (see earlier part of the lecture)

- Secondly, mutation testing is not all good news

Simple example

Apache Commons Lang: 113 classes, 3869 tests

- Normal test execution
  → 1 minute and 14 seconds

- With mutation analysis
  >> Generated 13021 mutations
  >> Killed 11113 (85%)
  >> Ran 51176 tests (3.93 tests per mutation)
  → Total: 31 minutes and 38 seconds

Simple example

Apache Commons Lang: 113 classes, 3869 tests

- Normal test execution
  → 1 minute and 14 seconds

- With mutation analysis
  >> Generated 13021 mutations
  >> Killed 11113 (85%)
  >> Ran 51176 tests (3.93 tests per mutation)
  → Total: 31 minutes and 38 seconds

Tests * mutations
?=
Test executions

Simple example

Apache Commons Lang: 113 classes, 3869 tests

- Normal test execution
  → 1 minute and 14 seconds

- With mutation analysis
  >> Generated 13021 mutations
  >> Killed 11113 (85%)
  >> Ran 51176 tests (3.93 tests per mutation)
  → Total: 31 minutes and 38 seconds

Tests * mutations

- Normally: 3869 tests * 13021 mutations
  → 50.3M test executions
- With PIT mutation tool: ~50K tests executed

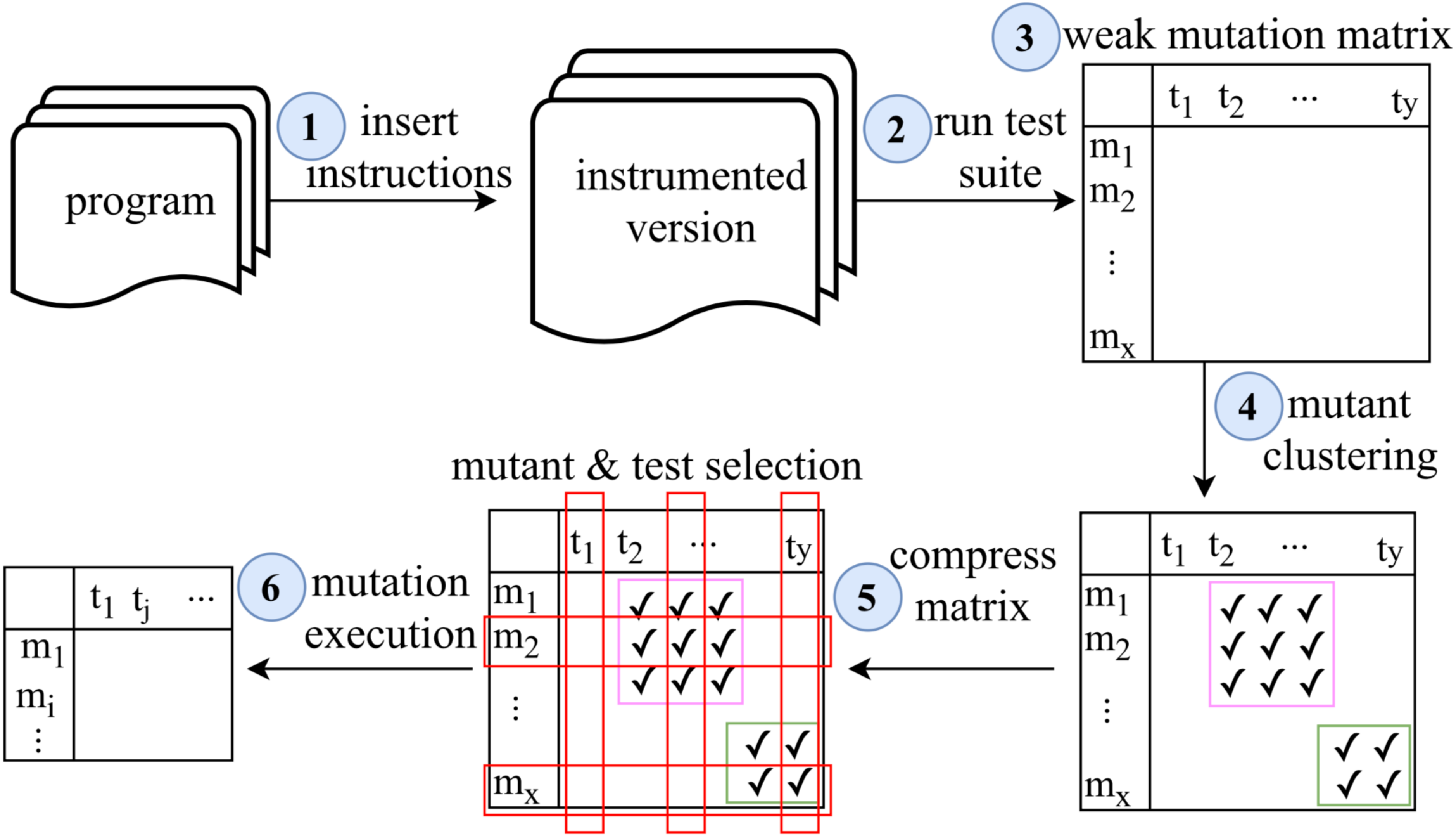
→ optimizations

Weak versus strong mutation

- Weak mutation: check whether an applied mutation causes a change in state, but output does not necessarily change

- Strong mutation: check whether an applied mutation causes a change in the output (e.g. in the test)

More optimizations?



Mutant equivalence problem

```
public void m(int a, int b)
{
        a=b;

        int x = a+b;

        print x;

}
```

```
public void m(int a, int b)
{
        a=b;

        int x = a+a;

        print x;

}
```

What happens if x is the same?

- Weak mutation?
- Strong mutation?
- Mutation score?

Manual assessment of equivalent mutant
→ Anywhere between 2m05s and 26m40s*

* D. Schuler and A. Zeller, Covering and Uncovering Equivalent Mutants, STVR, 2012.

Mutant equivalence problem

- Many automated or semi-automated detection techniques exist
  - Costly in terms of processing power
  - Rarely used in practice

See: L. Madeyeski, C. Orzeszyna, R. Torkar, M. Jozala, Overcoming the Equivalent Mutant Problem: A Systematic Literature Review and a Comparative Experiment of Second Order Mutation, IEEE Transaction on Software Engineering, 2014

So SW engineers don’t test (sufficiently)…

What to do about it?

- Show benefits
- TDD
- Regression testing
- Continuous Integration
- Minimal % coverage
- Policy
- Tooling
- Automated testing

What to do about it? (my list)

- Continuous Integration / Continuous Deployment
- Pull-based development (integrator model) / code review
- Online testing
- DevOps
- …

Continuous Integration (CI)

- Make sure that automated testing is an integral part of the build
- People don’t want to be responsible for breaking the build
  - Actually, failing tests are the #1 reason for build breakage
- From CI to CD (Continuous Deployment)

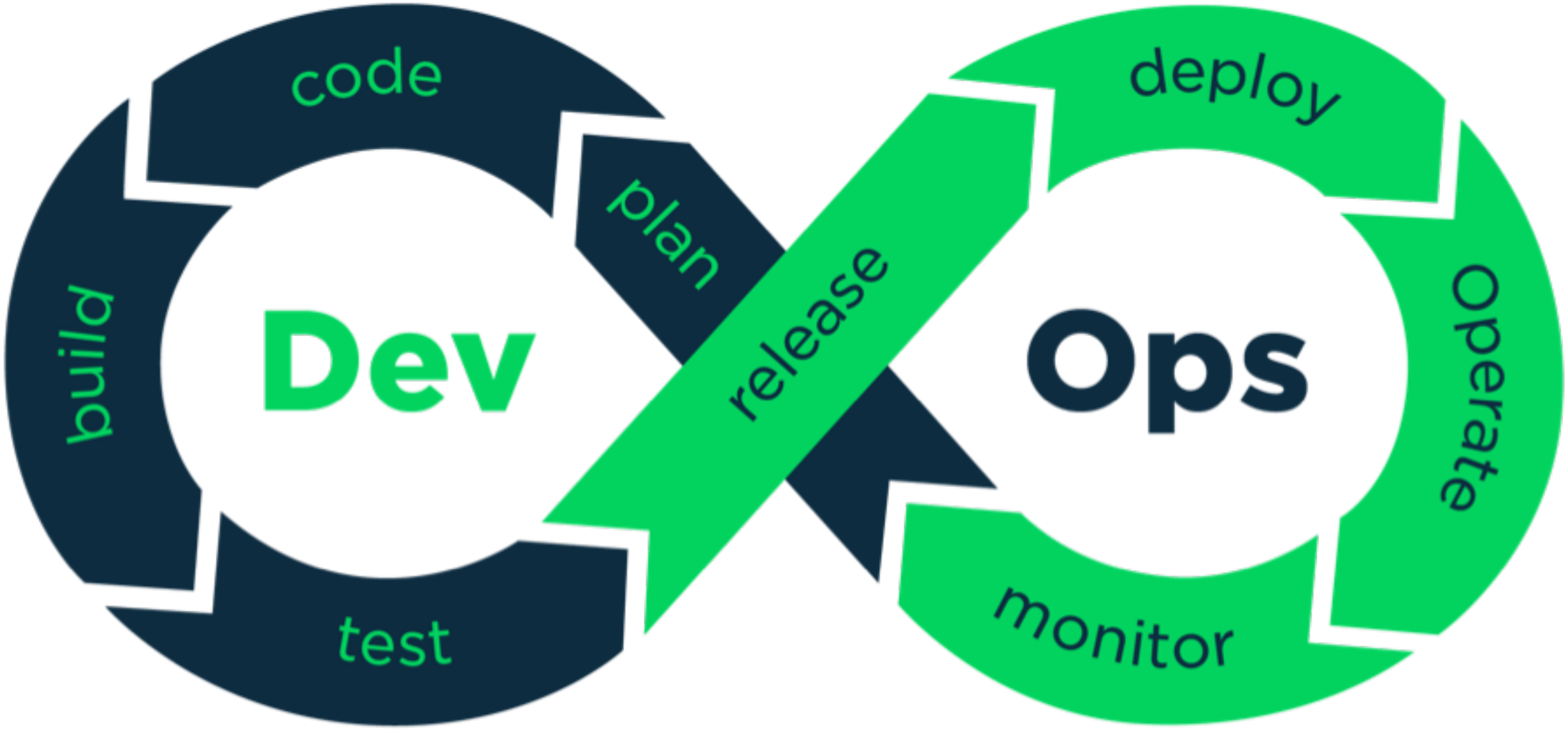
Pull requests (PR) / code review

- Integrators are known to look into tests and code coverage
- Tools like **coveralls** also do differential coverage, allowing an integrator to specifically look into the coverage of a PR

Online testing

- Proponents argue that not everything can be tested at design time
  - Is this for example true when considering REST API?
  - Microservices?

DevOps


code
plan
build
Dev
test
release
deploy
Ops
Operate
monitor

Your mission

(1st round assignment)

Read (at least) these papers

- René Just, Darioush Jalali, Laura Inozemtseva, Michael D. Ernst, Reid Holmes, Gordon Fraser: Are mutants a valid substitute for real faults in software testing? SIGSOFT FSE 2014: 654-665
- Laura Inozemtseva, Reid Holmes: Coverage is not strongly correlated with test suite effectiveness. ICSE 2014: 435-445

PITest (1)

- Use PITest http://www.pitest.org
- Useful PIT tutorial https://bitbucket.org/hascode/pitest-tutorial
- Use it on your own project + at least 3 other OSS projects
- Compare code coverage (line/branch) with mutation score
  - What do you see?
  - How does this relate to work of Inozemtseva?

PITest (2)

- What kind of “things” does PITest point to that you would not otherwise have caught (e.g., with test coverage)
  - Try to look at projects individually
  - Try to abstract over projects
- What kind of mutation operators are most likely to induce a surviving mutant

Questions?