How Mutation Testing Got Practical

FOSDEM '24

Hi!

Jan-Jelle Kester

Info Support

Software Engineering Consultant
Trainer
Research Supervisor

jjkester
jjkester

Mutation testing framework

for JS/TS, C#, Scala, ~~Kotlin~~

stryker-mutator.io

In the next 25 minutes

- **Why** we need to understand our tests
- **What** mutation testing is
- **How** mutation testing got to practical applicability

In the next 25 minutes

- **Why** we need to understand our tests
- **What** mutation testing is
- **How** mutation testing got to practical applicability

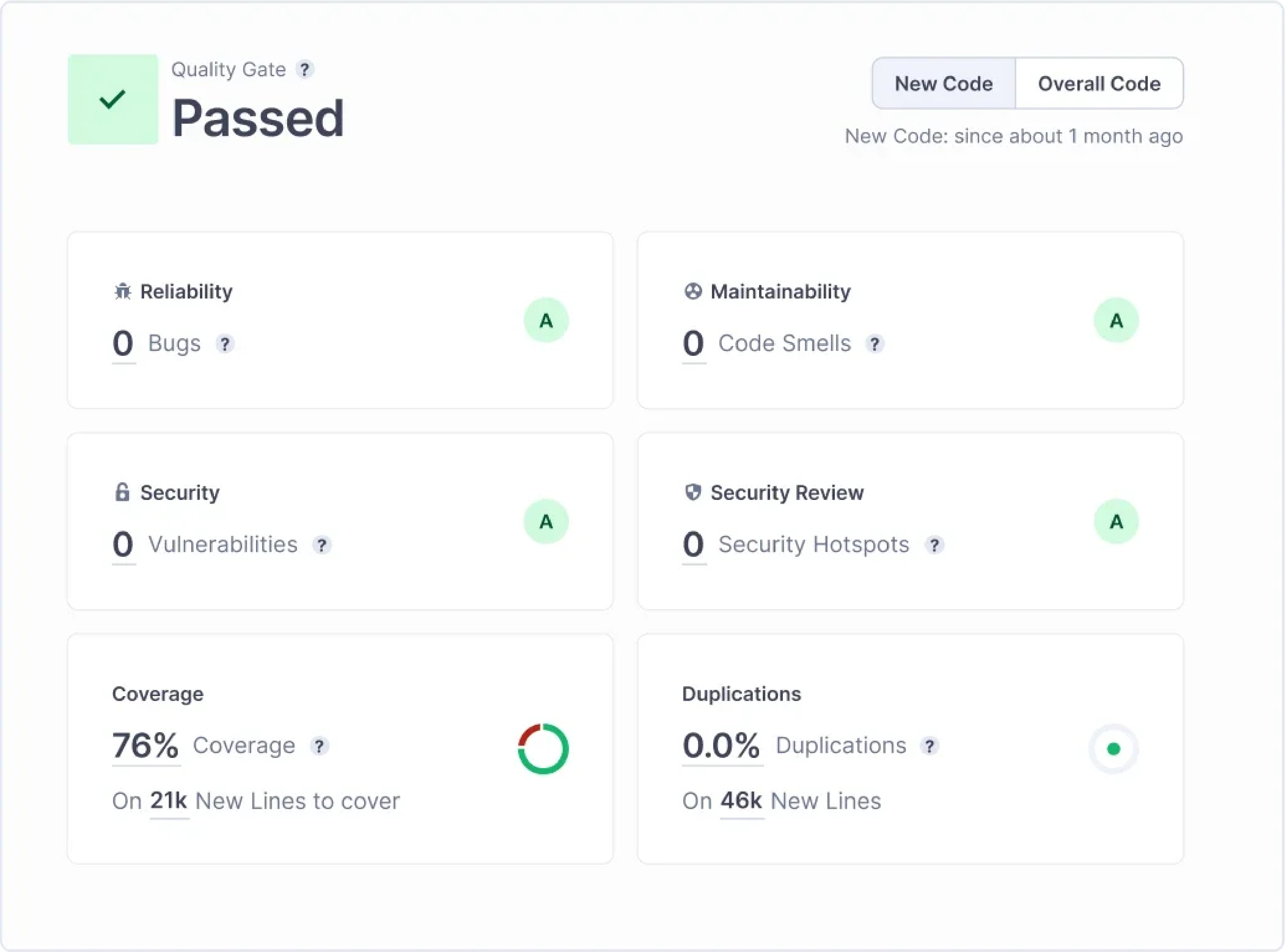
In the next 25 minutes

- **Why** we need to understand our tests
- **What** mutation testing is
- **How** mutation testing got to practical applicability

In the next 25 minutes

- **Why** we need to understand our tests
- **What** mutation testing is
- **How** mutation testing got to practical applicability
  - State-of-art performance improvements

A false sense of security



Quality Gate ?

Passed

New Code: since about 1 month ago

Reliability

0 Bugs ? A

Maintainability

0 Code Smells ? A

Security

0 Vulnerabilities ? A

Security Review

0 Security Hotspots ? A

Coverage

76% Coverage ?

On 21k New Lines to cover

Duplications

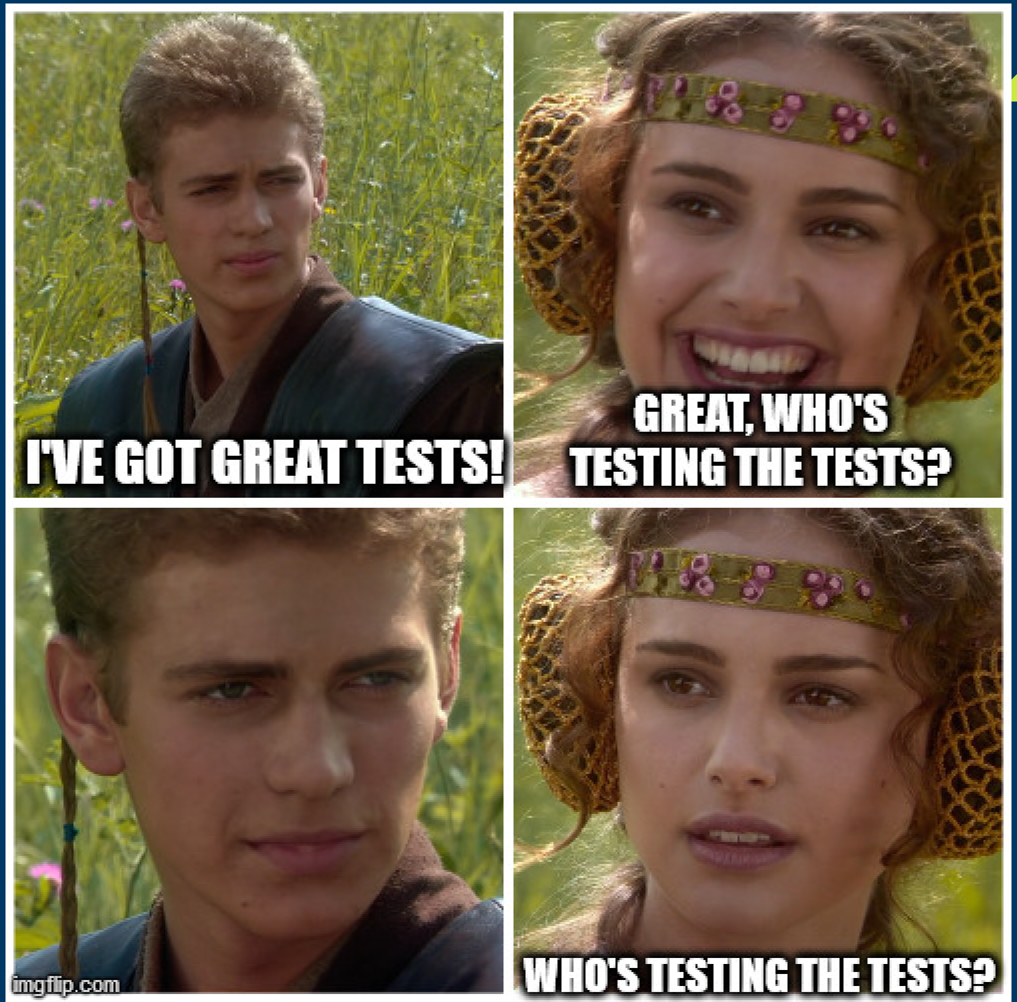
0.0% Duplications ?

On 46k New Lines

Source: https://www.sonarsource.com/products/sonarqube/ (2024-01-28)

Coverage only means that code is executed

We can have high code coverage without asserting anything!



Testing the tests

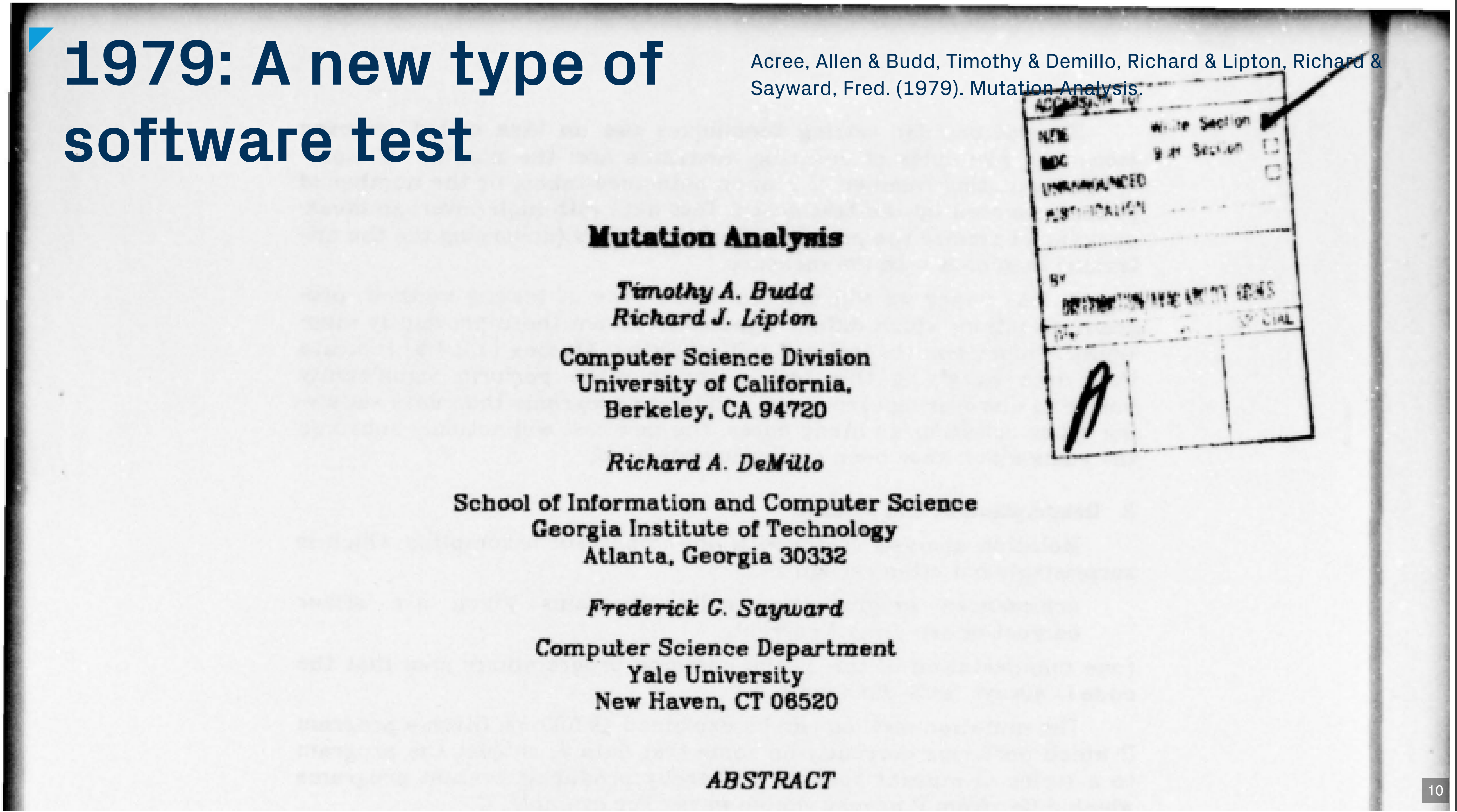
Mutation testing

Introducing **changes** in production code,
then checking whether the test suite **fails** to detect those changes

- White-box testing

1979: A new type of software test

Acree, Allen & Budd, Timothy & Demillo, Richard & Lipton, Richard & Sayward, Fred. (1979). Mutation Analysis.



Mutation Analysis

Timothy A. Budd
Richard J. Lipton

Computer Science Division
University of California,
Berkeley, CA 94720

Richard A. DeMillo

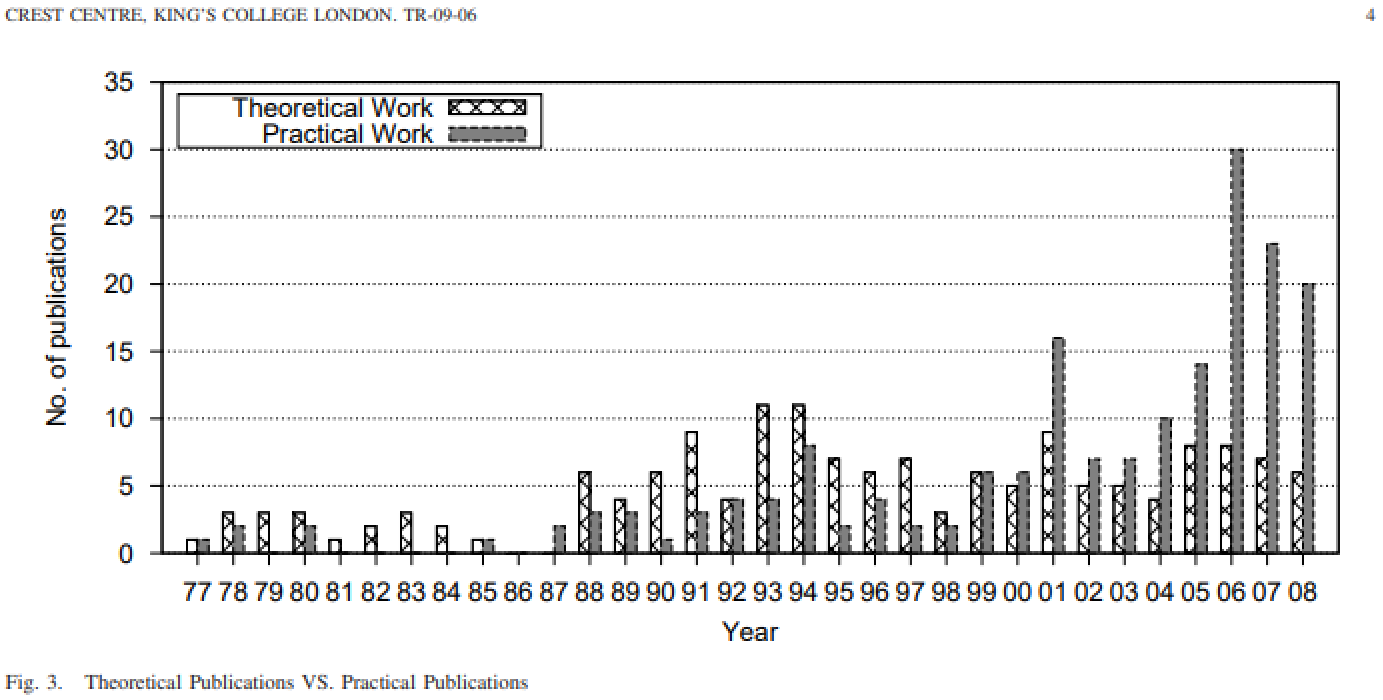
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332

Frederick G. Sayward

Computer Science Department
Yale University
New Haven, CT 06520

ABSTRACT

"Recent" popularity



CREST CENTRE, KING'S COLLEGE LONDON. TR-09-06 4

Fig. 3. Theoretical Publications VS. Practical Publications

Y. Jia and M. Harman, "An Analysis and Survey of the Development of Mutation Testing," in IEEE Transactions on Software Engineering, vol. 37, no. 5, pp. 649-678, Sept.-Oct. 2011, DOI: 10.1109/TSE.2010.62.

Mutation testing process

😀

1. Source code

Mutation testing process

😀 ⟶ 👽

1. Source code
2. Mutant

Mutation testing process

😀 → 👽 → ✅ ❌

1. Source code
2. Mutant
3. Killed / survived

Mutation testing process

😀 → 👽 → ✅❌ → 📊

1. Source code
2. Mutant
3. Killed / survived
4. Report

Mutation operators

Transform operations in source code to one or more mutated versions of that source code

Common mutations

Original	Mutated
`a + b`	`a - b`
`a / b`	`a * b`
`a < b`	`a > b`
`a == b`	`a != b`
`a && b`	`a || b`
`"Cola"`	`""`
`[1, 2, 3, 4]`	`[]`
`a > b`	`true`
`{ ... }`	`{}`

Mutant states

- ✅ Killed
- 👽 Survived

Mutant states

- ✅ Killed
- 👽 Survived
- 🙈 No coverage
- ⌛ Timeout
- 💥 Runtime
- 💥 Compile

Mutant states

- ✅ Killed
- 👽 Survived
- 🙈 No coverage — *no tests are reaching the code*
- ⏳ Timeout — *mutation caused an infinite loop*
- 💥 Runtime — *mutation caused an exception*
- 💥 Compile — *mutation resulted in invalid code*

Mutant states

- ✅ Killed
- 👽 Survived
- 🙈 No coverage — *no tests are reaching the code*
- ⏳ Timeout — *mutation caused an infinite loop*
- 💥 Runtime — *mutation caused an exception*
- 💥 Compile — *mutation resulted in invalid code*
- 🤥 Ignored

Mutation score

Is the code tested adequately?

$$M = \text{ set of mutants } \{m_1, \ldots, m_n\}$$

$$mutationScore(M) = \frac{M_{\text{✅}} + M_{\text{⏳}}}{M_{\text{✅}} + M_{\text{⏳}} + M_{\text{👽}} + M_{\text{🙈}}} \times 100\%$$

Mutation score

Is the code *that is tested* being tested adequately?

$$M = \text{ set of mutants } \{m_1, ..., m_n\}$$

$$coveredMutationScore(M) = \frac{M_{\text{✅}} + M_{\text{⏳}}}{M_{\text{✅}} + M_{\text{⏳}} + M_{\text{👽}}} \times 100\%$$

Not all mutants can be killed

While it is easy to *reach* all your code, it is not possible to write a test case for every possible internal change of your program

Equivalent mutants

```javascript
function calculateInLoop() {
    var value = 0;
    for (i = 0; i < 10; i++) {
        value += 1;
    }
    return value
}
```

```javascript
expect(calculateInLoop).to.equal(45); /* ✅ Passes */
```

Equivalent mutants

```
function calculateInLoop() {
    var value = 0;
    for (i = 0; i < 10; i++) {
        value += 1;
    }
    return value
}
```

```
expect(calculateInLoop).to.equal(45); /* ✅ Passes */
```

Equivalent mutants

```
function calculateInLoop() {
    var value = 0;
    for (i = 0; i != /* 👽 */ 10; i++) { /* ❌ Survived */
        value += 1;
    }
    return value
}
```

```
expect(calculateInLoop).to.equal(45); /* ✅ Passes */
```

Mutation testing is challenging

- 🐌 Takes *a lot* of time
- 🛠️ Requires configuration
- 👷 Requires tooling support

For a long time, mutation testing was simply **not feasible** and/or **not easy**

Bridging the gap

Performance

For every mutation we run the whole test suite once.

$$t_m = |T| \;\Big|\; m \in M$$

$$t_M = \sum_{m \in M} |T| = |T| \times |M|$$

Performance

For every mutation we run the whole test suite once.

$$t_m = |T| \;\Big|\; m \in M$$

$$t_M = \sum_{m \in M} |T| = |T| \times |M|$$

We need to be smarter: $t_M < |T| \times |M|$!

Performance

Three approaches to improving performance

- 🏎️ Do **faster**
- 🦥 Do **fewer**
- 🧐 Do **smarter**

A. Pizzoleto, F. Ferrari, J. Offutt, L. Fernandes, and M. Ribeiro, “A systematic literature review of techniques and metrics to reduce the cost of mutation testing,” Journal of Systems and Software, vol. 157, Jul. 2019. DOI: 10.1016/j.jss.2019.07.100.

Performance

Three approaches to improving performance

- 🏎️ Do **faster**: 27 studies
- 🦥 Do **fewer**: 118 studies
- 🧐 Do **smarter**: 75 studies

A. Pizzoleto, F. Ferrari, J. Offutt, L. Fernandes, and M. Ribeiro, “A systematic literature review of techniques and metrics to reduce the cost of mutation testing,” Journal of Systems and Software, vol. 157, Jul. 2019. DOI: 10.1016/j.jss.2019.07.100.

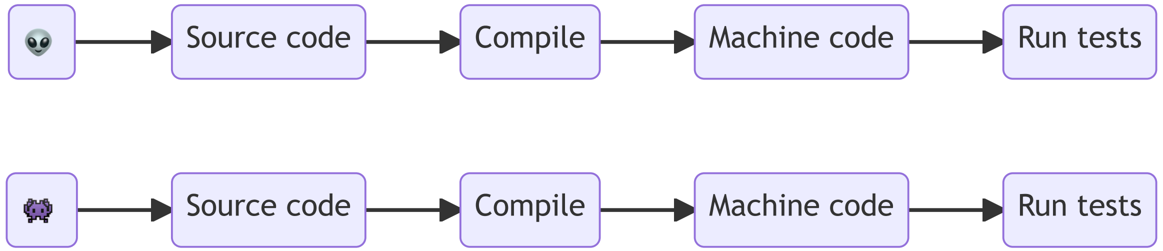
Common techniques

- 🦥 Random mutation
- 🧐 Higher order mutation
- 🏎️ Parallel execution
- 🦥 Data-flow analysis
- 🦥 Control-flow analysis
- 🧐 Minimization and prioritization of test sets
- 🦥 Constrained mutation
- 🧐 Evolutionary algorithms
- 🧐 Model-based mutation
- 🧐 State-based analysis
- 🦥 Minimal mutation
- 🦥 Selective mutation

Mutation strategies

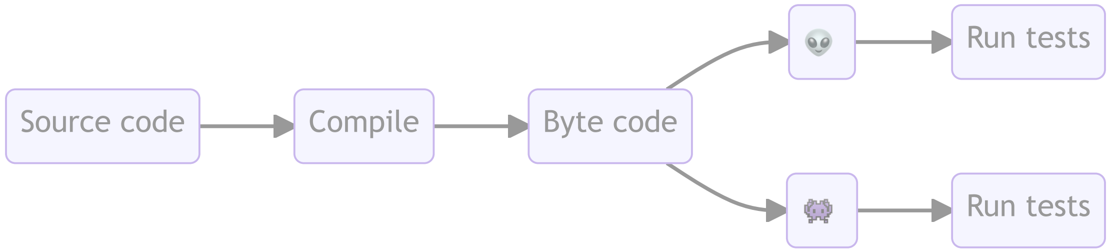
Placing mutations into source code

Source code mutation



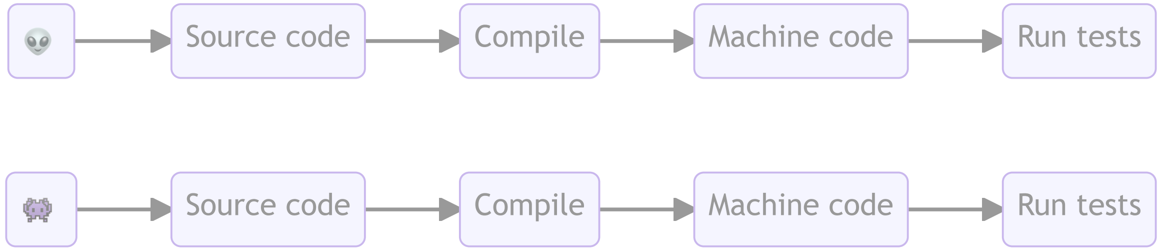
- ✅ Precise
- ✅ Easy
- ❌ Slow

Byte code mutation



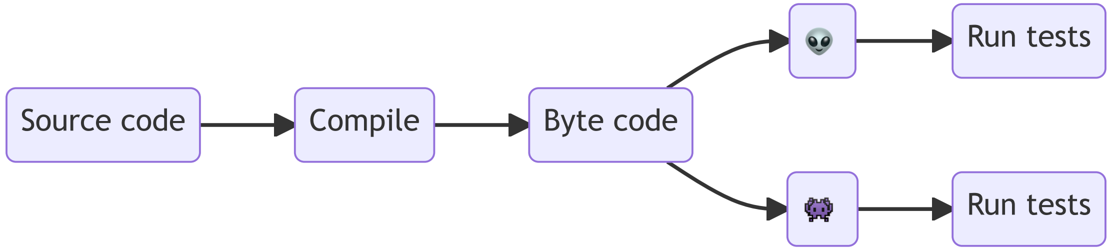
- ✅ Fast...ish
- ❌ False positives
- ❌ Complicated

Source code mutation



- ✅ Precise
- ✅ Easy
- ❌ Slow

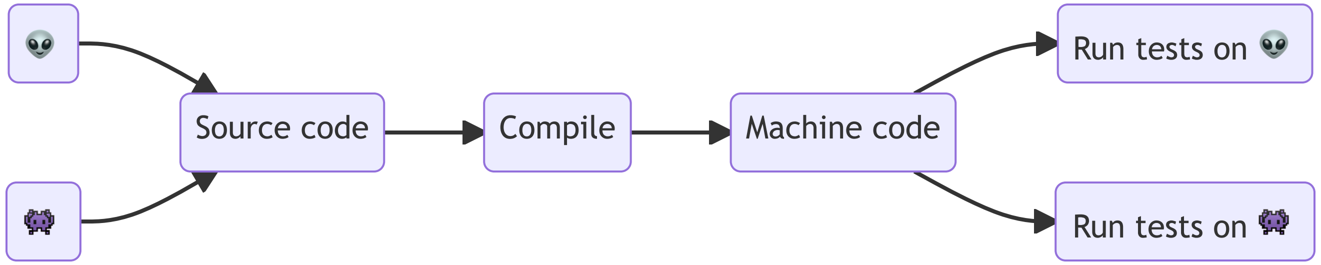
Byte code mutation



- ✅ Fast...ish
- ❌ False positives
- ❌ Complicated

Mutant schemata 🏎️

Generate mutants based on source code, but compile once



👽

👾

Source code

Compile

Machine code

Run tests on 👽

Run tests on 👾

- ✅ Precise
- ✅ Fast
- 🟡 Complicated (but manageable)

Roland H. Untch, A. Jefferson Offutt, and Mary Jean Harrold. 1993. Mutation analysis using mutant schemata. SIGSOFT Softw. Eng. Notes 18, 3 (July 1993), 139–148. DOI: 10.1145/174146.154265.

Coverage analysis 🧐

Test coverage: which code is hit by which tests

- Only run tests that cover a mutation instead of the whole test suite

Incremental analysis 🦥

Re-use results from a previous run

- Only analyze changes from previous run

Mutation levels 🦥

Selective mutation approach by Info Support's Jan Smits

- User choice depending on requirements
  - Type of project / domain
  - Pull request / nightly build

Mutation levels 🦥

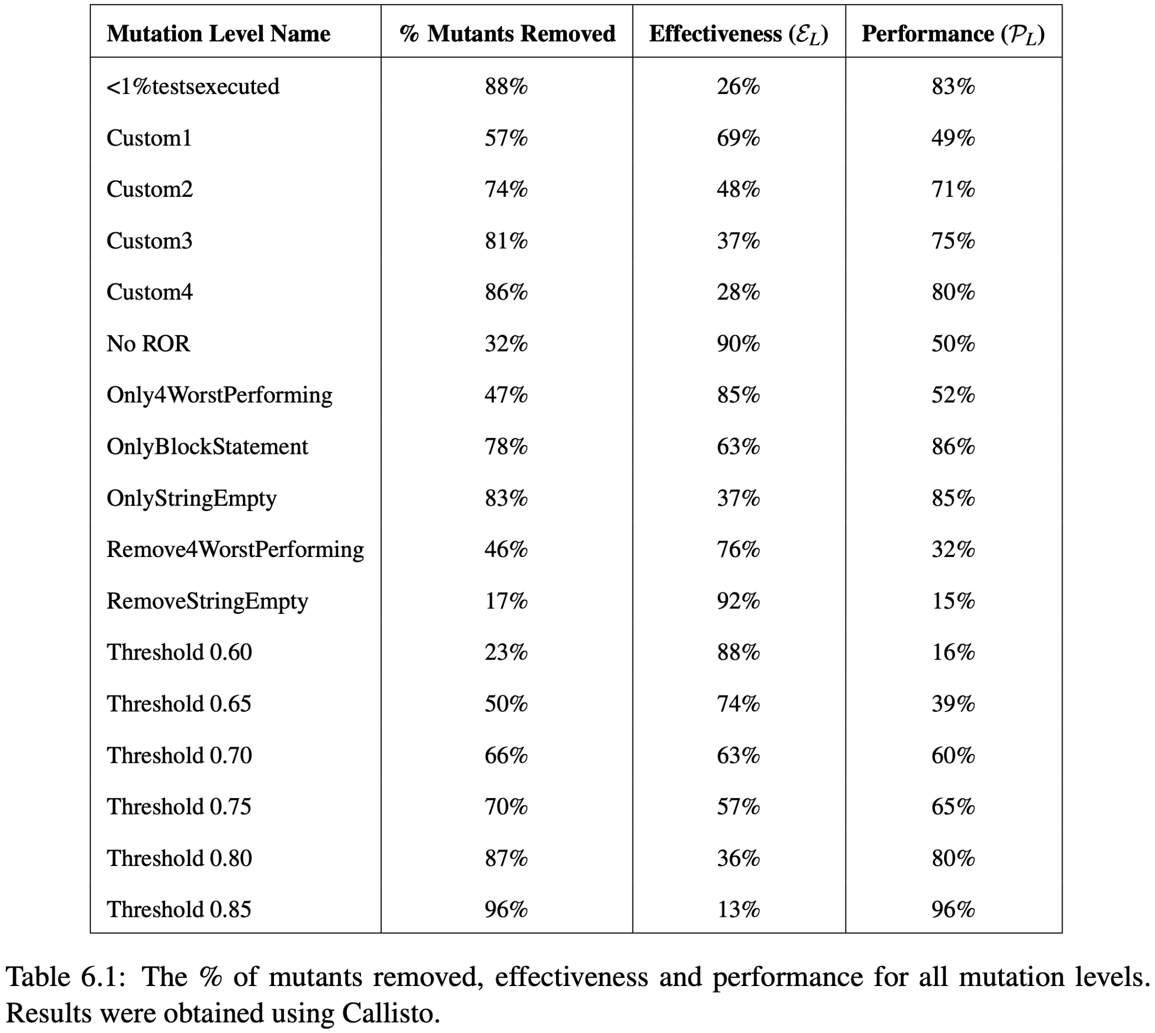
Selective mutation approach by Info Support's Jan Smits

- User choice depending on requirements
    - Type of project / domain
    - Pull request / nightly build

Mutation score not necessarily comparable!

Mutation levels: Callisto

- Full run of mutation testing as input
- Finds balance between accuracy and number of test executions



Mutation Level Name	% Mutants Removed	Effectiveness ($\mathcal{E}_L$)	Performance ($\mathcal{P}_L$)
<1%testsexecuted	88%	26%	83%
Custom1	57%	69%	49%
Custom2	74%	48%	71%
Custom3	81%	37%	75%
Custom4	86%	28%	80%
No ROR	32%	90%	50%
Only4WorstPerforming	47%	85%	52%
OnlyBlockStatement	78%	63%	86%
OnlyStringEmpty	83%	37%	85%
Remove4WorstPerforming	46%	76%	32%
RemoveStringEmpty	17%	92%	15%
Threshold 0.60	23%	88%	16%
Threshold 0.65	50%	74%	39%
Threshold 0.70	66%	63%	60%
Threshold 0.75	70%	57%	65%
Threshold 0.80	87%	36%	80%
Threshold 0.85	96%	13%	96%

Table 6.1: The % of mutants removed, effectiveness and performance for all mutation levels. Results were obtained using Callisto.

Smits, J. P. G. (2022). Callisto-Selecting Effective Mutation Operators for Mutation Testing (Master's thesis, University of Twente). Summary @ research.infosupport.com, Thesis @ utwente.nl.

Mutation levels: project Xavier

Mutation levels implementation in Stryker JS

- **Hot off the press**: implementation done, pull request #4686 open

Mutation levels: project Xavier

Mutation levels implementation in Stryker JS

- **Hot off the press**: implementation done, pull request #4686 open

Implemented by a project group of CS master students from the University of Twente

Mutation levels: project Xavier

Mutation levels implementation in Stryker JS

- **Hot off the press**: implementation done, pull request #4686 open

Implemented by a project group of CS master students from the University of Twente

- Documentation to follow...

Further reducing test runs 🧐

Analyze multiple mutants per test run

- Minimal number of test runs
- Combine mutants that do not influence each other
- No negative effects on accuracy

Further reducing test runs 🧐

Analyze multiple mutants per test run

- Minimal number of test runs
- Combine mutants that do not influence each other
- No negative effects on accuracy

Current graduation project of CS master student at Info Support

Time to test your tests!

In general

A lot of progress in 45 years

- Better hardware
- Lots of process improvements

In general

A lot of progress in 45 years

- Better hardware
- Lots of process improvements

We have production-ready tooling

- Integrates with build tool
- Uses information already provided by your tests
- Ability to run on CI pipeline

Mutation testing for your language of choice

Language	Framework
JavaScript & TypeScript	StrykerJS
Scala	Stryker4s
C#	Stryker.NET
Java	PIT
PHP	InfectionPHP
Ruby	Mutant
Python	Cosmic Ray
C/C++	Mull
Go	Gremlins
Swift	Muter

More options available: https://github.com/theofidry/awesome-mutation-testing (or search `${lang} mutation testing`)

Conclusion

- Mutation testing is testing the tests
- Don't rely on code coverage, use mutation score to check assertions
- A lot of research in performance improvements
  - Still open research questions
- Applicable now

Conclusion

- Mutation testing is testing the tests
- Don't rely on code coverage, use mutation score to check assertions
- A lot of research in performance improvements
  - Still open research questions
- Applicable now

Conclusion

- Mutation testing is testing the tests
- Don't rely on code coverage, use mutation score to check assertions
- A lot of research in performance improvements
  - Still open research questions
- Applicable now

Conclusion

- Mutation testing is testing the tests
- Don't rely on code coverage, use mutation score to check assertions
- A lot of research in performance improvements
  - Still open research questions
- Applicable now

Get started with StrykerJS

Who's testing the tests? Mutation testing with StrykerJS

Saturday, 18:30-19:00

Javascript devroom, H.1301 (Cornil)

- Watch back the slides and/or video online

Mutation testing framework

for JS/TS, C#, Scala, ~~Kotlin~~

stryker-mutator.io

infoSupport
Solid Innovator

Jan-Jelle Kester

Software Engineering Consultant
Trainer
Research Supervisor

jjkester
in jjkester

research.infosupport.com