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

- Reliability: 0 Bugs (A)
- Maintainability: 0 Code Smells (A)
- Security: 0 Vulnerabilities (A)
- Security Review: 0 Security Hotspots (A)
- Coverage: 76% Coverage (21k New Lines to cover)
- Duplications: 0.0% Duplications (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

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 C. Sayward

Computer Science Department
Yale University
New Haven, CT 06520

ABSTRACT
"Recent" popularity

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

<table>
<thead>
<tr>
<th>Original</th>
<th>Mutated</th>
</tr>
</thead>
<tbody>
<tr>
<td>a + b</td>
<td>a - b</td>
</tr>
<tr>
<td>a / b</td>
<td>a * b</td>
</tr>
<tr>
<td>a &lt; b</td>
<td>a &gt; b</td>
</tr>
<tr>
<td>a == b</td>
<td>a != b</td>
</tr>
<tr>
<td>a &amp;&amp; b</td>
<td>a</td>
</tr>
<tr>
<td>&quot;Cola&quot;</td>
<td>&quot;&quot;&quot;</td>
</tr>
<tr>
<td>[1, 2, 3, 4]</td>
<td>[]</td>
</tr>
<tr>
<td>a &gt; b</td>
<td>true</td>
</tr>
<tr>
<td>{ ... }</td>
<td>{}</td>
</tr>
</tbody>
</table>
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?

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

\[ \text{mutationScore}(M) = \frac{M\checkmark + M\footnotesize\square}{M\checkmark + M\footnotesize\square + M\footnotesize\triangle + M\footnotesize\triangle\footnotesize\square} \times 100\% \]
Mutation score

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

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

\[
\text{coveredMutationScore}(M) = \frac{M\checkmark + M\square}{M\checkmark + M\square + M\triangle} \times 100\%
\]
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

```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

```javascript
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.

\[ t_m = |T| \quad m \in M \]
\[ t_M = \sum_{m \in M} |T| = |T| \times |M| \]
Performance

For every mutation we run the whole test suite once.

\[ t_m = |T| \quad m \in M \]

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

We need to be smarter: \( t_M < |T| \times |M| \)!
Performance

Three approaches to improving performance

- Do faster
- Do fewer
- Do smarter

Performance

Three approaches to improving performance

- Do faster: 27 studies
- Do fewer: 118 studies
- Do smarter: 75 studies

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

- Precise
- Fast
- Complicated (but manageable)

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

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

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

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

<table>
<thead>
<tr>
<th>Language</th>
<th>Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>JavaScript &amp; TypeScript</td>
<td>StrykerJS</td>
</tr>
<tr>
<td>Scala</td>
<td>Stryker4s</td>
</tr>
<tr>
<td>C#</td>
<td>Stryker.NET</td>
</tr>
<tr>
<td>Java</td>
<td>PIT</td>
</tr>
<tr>
<td>PHP</td>
<td>InfectionPHP</td>
</tr>
<tr>
<td>Ruby</td>
<td>Mutant</td>
</tr>
<tr>
<td>Python</td>
<td>Cosmic Ray</td>
</tr>
<tr>
<td>C/C++</td>
<td>Mull</td>
</tr>
<tr>
<td>Go</td>
<td>Gremlins</td>
</tr>
<tr>
<td>Swift</td>
<td>Muter</td>
</tr>
</tbody>
</table>

More options available: [https://github.com/theofidry/awesome-mutation-testing](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

Jan-Jelle Kester
Software Engineering Consultant
Trainer
Research Supervisor
jjkester
jjkester
research.infosupport.com