

# MUTATION TESTING

*“Who will watch the watchmen?”*  
– Juvenal, 115 AD

**José Cunha & Nelson Almeida**

TVVS – FEUP – 2019/2020  
Prof. Ana Paiva

# CODE COVERAGE

- Code coverage, by itself, is a poor test quality measure.
- Code coverage guarantees code is being tested - does not ensure it is done thoroughly.

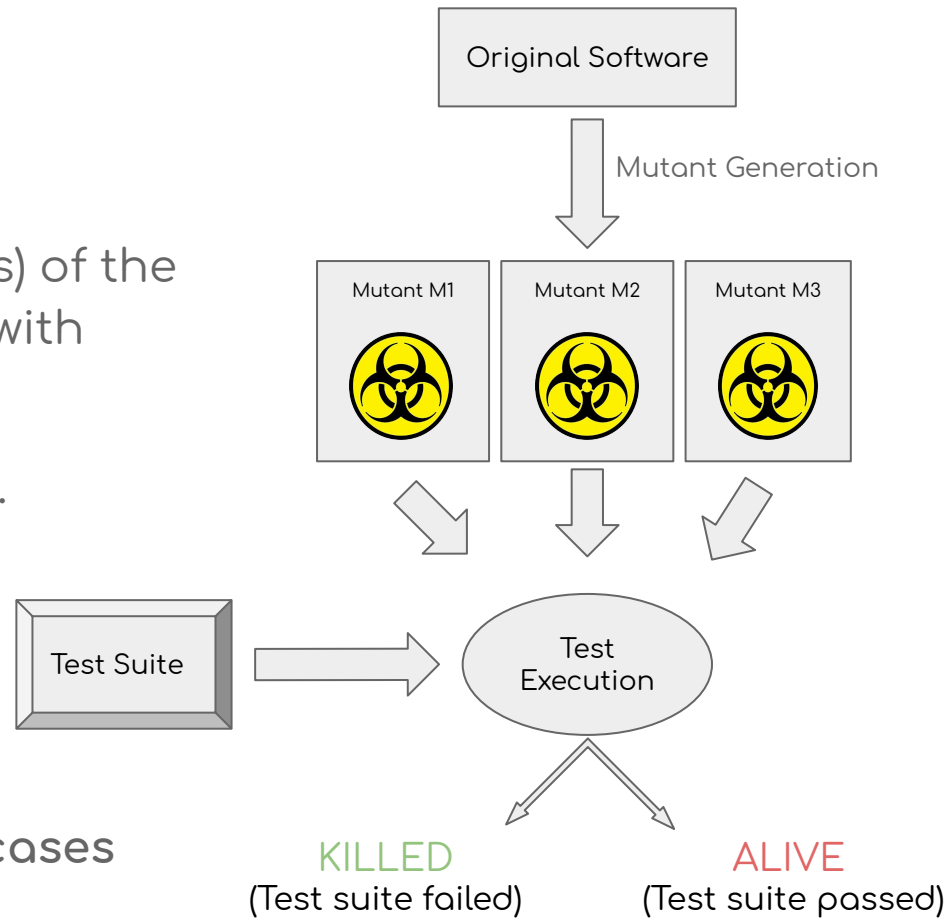
100% coverage means nothing.

# MUTATION TESTING

- Mutation testing consists in making **small changes** to the code, deliberately **introducing defects** in the program.
- Original test suite is then run on the modified program. If **tests are robust enough** they will **fail, detecting** the introduced **defects**.
- Code coverage is a good test quality indicator when combined with mutation testing

# PROCEDURE

- Different modified versions (Mutants) of the original code are created, inserted with typical programming errors.
- Test suite is applied to each mutant.
- A mutant is **killed** if it fails in any of the test cases. Which means the test suite is **robust** enough to detect the generated error.
- The mutant remains **alive** if all test cases pass. Declaring the test suite as **insufficient** to detect the given error.



# MUTATION OPERATORS

Operators are rules applied to a program, originating mutants.

## Traditional Operators

Original

Relational Operator Replacement

```
if(a > b)
    a = 2*b;
else
    b = 3;
```

```
if(a <= b)
    a = 2*b;
else
    b = 3;
```

Boolean Expressions

Original

Arithmetic Operator Replacement

```
if(a > b)
    a = 2*b;
else
    b = 3;
```

```
if(a > b)
    a = 2/b;
else
    b = 3;
```

Arithmetic Expressions

# TYPES OF MUTANTS

## First Order Mutants

Single syntactic change to the original program

Original	First Order Mutant
<pre>if(a &gt; b)     a = 2*b; else     b = 3;</pre>	<pre>if(a &lt;= b)     a = 2*b; else     b = 3;</pre>

## Higher Order Mutants

Multiple first order defects to simulate more complex faults

Original	Higher Order Mutant
<pre>if(a &gt; b)     a = 2*b; else     b = 3;</pre>	<pre>if(a &lt;= b)     a = 2/b; else     b = 3;</pre>

# MUTATION TESTING STRENGTH

Mutant kill conditions:

- 1) Test input causes different states for the mutant and the original program
- 2) Output values must also be different and caught by the test case

## Weak Mutation Testing

Requires only first condition to be met

Used for coverage measures

Less computation power

## Strong Mutation Testing

Requires both conditions to be met

Used to detect test suite flaws

Test case	
Input: (a = 5; b = 2)	Expected Output: 0
Original	Mutant
<pre>if(a &gt; b)   a = 4*b; //(a = 8) else   b = 3;  return a%b; //(8%2 = 0)</pre>	<pre>if(a &gt; b)   a = 4/b; //(a = 2) else   b = 3;  return a%b; //(2%2 = 0)</pre>
Weak Testing - Mutant <b>KILLED</b> (flow undetected)	
Strong Testing - Mutant <b>ALIVE</b>	

# MUTATION TESTING PROS & CONS

## Advantages

- Discovers flaws in your test cases, increasing test suite robustness.

## Disadvantages

- High computational cost
  - Hard to detect mutations killed by basic test cases (Trivial Mutants)
  - Too much time required to find mutants functionally similar to original program (Equivalent Mutants)
- Requires programming knowledge



# MUTATION TESTING TOOLS



# ConfigValidator.ts - Stryker report

All files / ConfigValidator.ts

File / Directory Mutation score  
ConfigValidator.ts 88.64%

# Killed # Survived # Timeout # No coverage # Runtime errors # Transpile errors Total detected Total undetected Total mutants  
39 5 0 0 2 39 5 46

Survived (5) Killed (39) TranspileError (2) Expand all

```
import { TestFramework } from 'stryker-api/test_framework';
import { MutationScoreThresholds } from 'stryker-api/core';
import { Config } from 'stryker-api/config';
import { getLogger } from 'log4js';

export default class ConfigValidator {

  private isValid = true;
  private readonly log = getLogger(ConfigValidator.name);

  constructor(private strykerConfig: Config, private testFramework: TestFramework | null) {
  }

  validate() {
    this.validateTestFramework();
    this.validateThresholds();
    this.downgradeCoverageAnalysisIfNeeded();
    this.crashIfNeeded();
  }

  private validateTestFramework() {
    if (this.strykerConfig.coverageAnalysis === 'perTest' && !this.testFramework) {
      this.invalidate('Configured coverage analysis "perTest" requires there to be a testFramework configured. Either configure a testFramework or change coverageAnalysis to "all"');
    }
  }
}
```

#	Mutator	State	Location	Original	Replacement
0	BooleanSubstitution	Killed	7 : 20		
1	Block	Killed	13 : 13	{ ...; }	{ }
2	Block	Killed	20 : 34	{ ...; }	{ }
3	IfStatement	Killed	21 : 8	. ... .	
4	IfStatement	Killed	21 : 8	. ... .	
5	BinaryExpression	Killed	21 : 58	&&	
6	BinaryExpression	Killed	21 : 44	---	!--
7	Mutator	Stryker	Stryker.NET	Stryker4s	
8					
9	Arithmetic Operator	✓	✓	✗	
10	Array Declaration	✓	✗	✗	
11	Assignment Expression	✗	✓	n/a	
12	Block Statement	✓	✗	✗	
13	Boolean Literal	✓	✓	✓	
14	Checked Statement	n/a	✓	n/a	
15	Conditional Expression	✓	✓	✓	
	Equality Operator	✓	✓	✓	
	Logical Operator	✓	✓	✓	
	Method Expression	✗	✓	✓	
	String Literal	✓	✓	✓	
	Unary Operator	✓	✓	✗	
	Update Operator	✓	✓	n/a	

## Supported languages

- JavaScript
- TypeScript
- C#
- Scala

## Advantages

- You can write your own mutation operators
- Increased speed with parallel test runner processes
- Open source
- Pretty HTML report
- Supports Angular, React and vuejs

## Disadvantages

- Not all traditional mutation operators are available

# PHP TOOLS

## Infection

### Positives

- Supports PHP 7.1+
- Applies mutations at AST level
- Supports PHPUnit and PhpSpec testing frameworks
- Displays Mutations Score metrics
- Checks mutation code coverage
- Extensive mutation operators

### Negatives

- Requires Xdebug/phpdbg installed



# PYTHON TOOLS

## mutmut

### Positives

- Supports Python 3
- Supports all test runners
- Can use coverage data to only mutate covered code
- Supports JUnit XML report
- Allows whitelisting code to prevent mutation in specific places

### Negatives

- No GUI

## mutpy

### Positives

- Supports Python 3.3+
- Applies mutation at the AST level
- Shows changes on source
- Supports standard unittest module and pytest
- Generates YAML/HTML reports
- Big number command-line arguments for customization
- Allows higher order mutations (HOM)

### Negatives

- No GUI

# C/C++ TOOLS

## Mu11

### Positives

- Supports C++ and C
- Bitcode mutation
- Shows mutations changes in source

### Negatives

- Requires the use of the LLVM compiler
- Requires use of another tool for HTML report generation
- Junk mutations

## Mutate++

### Positives

- Supports C++
- Source code mutation
- GUI (web app)
- Shows execution time
- Shows mutations changes

### Negatives

- Isn't mature yet and has some issues
- Requires a lot of manual configuration

# JAVA TOOLS

	MuJava	Judy	Jumble	Jester	PIT
Traditional & Class-level operators	Yes	Yes	Some traditional operators	No	Traditional operators
Eclipse plug-in	No	No	Yes	No	Yes
JUnit support	Yes	Yes	Yes	Yes	Yes
Control over operator selection	Yes	Yes	No	No	Yes
Code coverage	No	No	No	No	Yes
Total execution time	No	Yes	No	Yes	Yes
Actively supported or developed	Yes	No	No	No	Yes