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

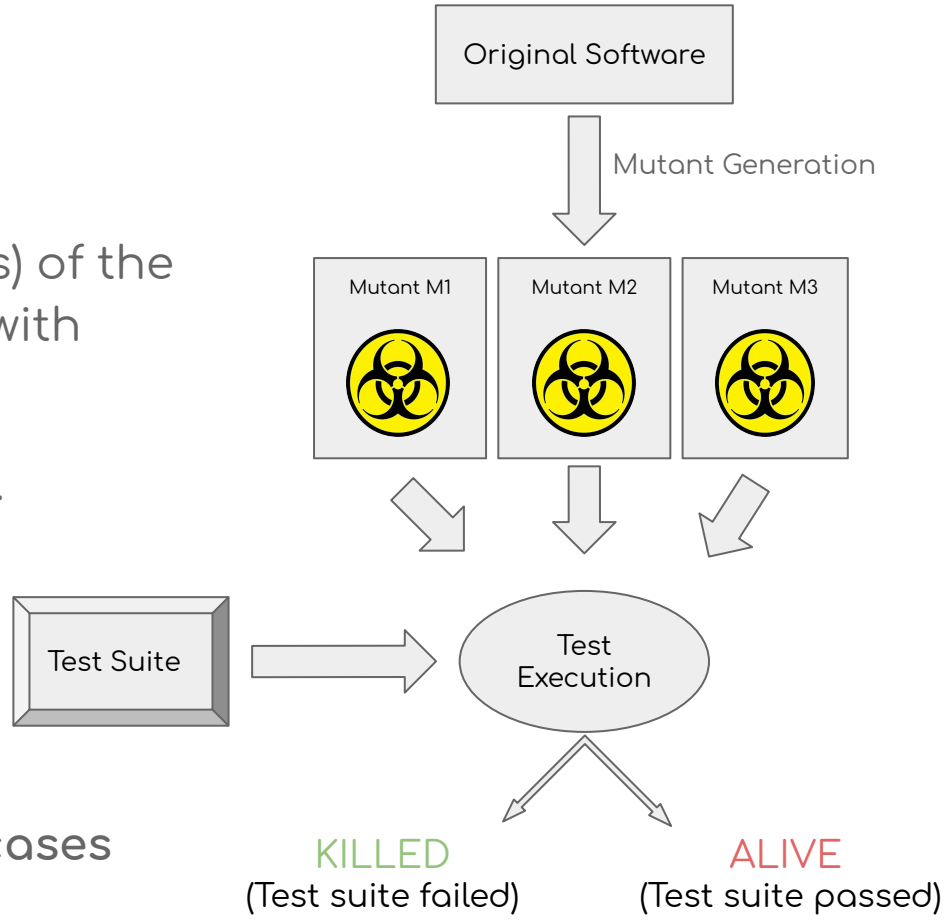
- Mutation testing consists in making **small changes** to the code, deliberately **introducing defects** in the program.

- 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

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

Relational Operator Replacement

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

Boolean Expressions

Original

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

Arithmetic Operator Replacement

```
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

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

First Order Mutant

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

Higher Order Mutants

Multiple first order defects to simulate more complex faults

Original

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

Higher Order Mutant

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

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

```
if(a > b)
    a = 4*b; //(a = 8)
else
    b = 3;

return a%b; //(8%2 = 0)
```

Mutant

```
if(a > b)
    a = 4/b; //(a = 2)
else
    b = 3;

return a%b; //(2%2 = 0)
```

Weak Testing - Mutant KILLED (flaw undetected)

Strong Testing - Mutant ALIVE

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

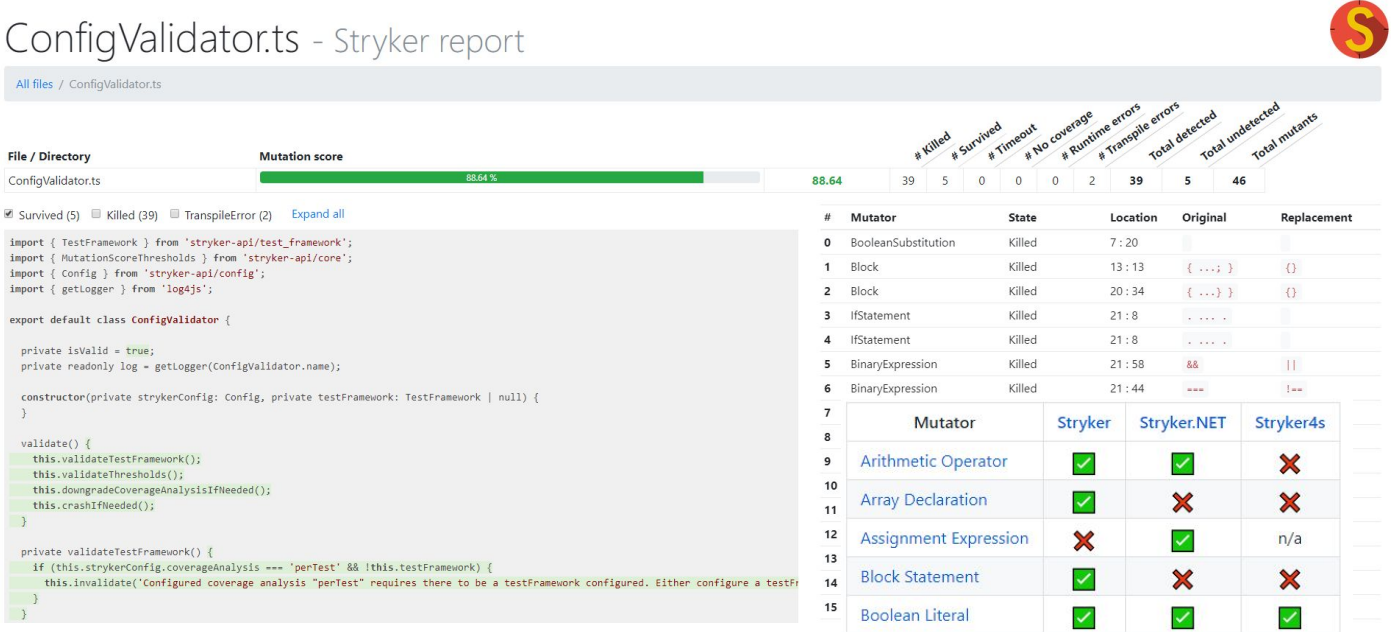
Stryker

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



Mutator	Stryker	Stryker.NET	Stryker4s
Arithmetic Operator	✅	✅	❌
Array Declaration	✅	❌	❌
Assignment Expression	❌	✅	n/a
Block Statement	✅	❌	❌
Boolean Literal	✅	✅	✅
Checked Statement	n/a	✅	n/a
Conditional Expression	✅	✅	✅
Equality Operator	✅	✅	✅
Logical Operator	✅	✅	✅
Method Expression	❌	✅	✅
String Literal	✅	✅	✅
Unary Operator	✅	✅	❌
Update Operator	✅	✅	n/a

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

Mull

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