

Mutation Testing

Testing, Verification and
Validation of Software

Universidade do Porto
2021-2022

Hugo Almeida
Orlando Macedo

Who Tests the Tests?

What metrics should be used to measure the quality of the **Tests**?

Code Coverage

Its main purpose is to find how many lines and branches are **executed** and **covered** by the unit tests.

It does not check if the **Tests** are actually able to **detect faults** in the executed code.

It is therefore only able to **identify** code that is **definitely not tested**.



Mutation Testing

Form of white-box testing that evaluate the quality of unit tests by **changing** the source code or byte code (creating **mutants**) and running the tests.

As an example, changes (called **mutations**) can be accomplished by:

- Duplicating or deleting a statement
- Alter True or False expressions/variables



Types of Mutation Testing

```
int a = 75636737;  
int b = 3454;  
int mult = a * b;  
print(mult);
```



```
int a = 75;  
int b = 345466465;  
int mult = a * b;  
print(mult);
```

Decision Mutation

Control statements are changed

Value Mutation


Values of parameters are modified

```
if (a>b || b>c)  
{  
    print("yes");  
}  
else  
{  
    print ("No");  
}
```



```
if (a<b || b<c)  
{  
    print("yes");  
}  
else  
{  
    print ("No");  
}
```

Types of Mutation Testing




```
if (a > b)
{
    print("a is greater");
}
else
{
    print("b is greater");
}
```



```
if(a > b)
{
    // removing the statement
}
else
{
    print("b is greater");
}
```

Statement Mutation

Changes are made in the statements of code. These might include deleting, changing order, repeating, etc.



Traditional Mutation Operators



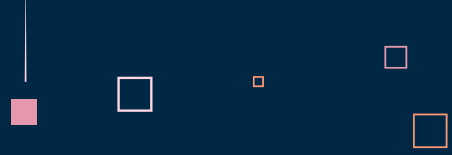
Replacement of a
variable with
another of the same
type



Replacement of
conditional expressions
(e.g. $< \rightarrow >$)



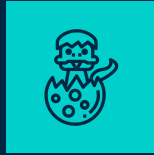
Removal of a
statement



How to execute Mutation Testing

Mutants Creation

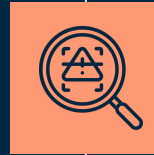
Introduction of faults into the source code



Test Cases

The original program and the mutants are tested

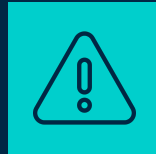
Test Results Comparison



The results from the tests are compared between the original and mutants programs

Same Output

Mutant is kept alive.
Test cases should be revisited

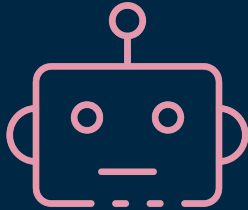


Different Output

Mutant is killed by
the test case

Types of Mutants

Survived/Live Mutants



Remain alive after the tests are ran. Test cases should be changed in order to kill them

Killed Mutants



Killed after the tests are ran. Tests results are different between the original and mutated variant of the source code

Equivalent Mutants



Like **Survived/Live Mutants** but the test cases can't be changed in order to kill them

Types of Mutants

First Order Mutants

Simple faults,
generated by a
single syntactic
change

Higher Order Mutants

Combination of
multiple first order
faults

Mutation Score

$$\text{Mutation Score} = \frac{\text{Killed Mutants}}{(\text{All Mutants} - \text{Number of Equivalent Mutants})} \times 100$$

- Test cases are mutation adequate if the score is 100%

Is It Worth It?

Advantages

Ability to identify **weak tests** or **code**.

Uncovers **ambiguities** in the source code.

Evaluate the **quality** of our test suite and adjust accordingly.

Disadvantages

Can be **time-consuming** and **expensive**.

Not applicable for Black Box Testing.

Need to counter check the surviving mutants, as some are invalid.



Tools



- Llvn-mutate
- Frama-C plugin
- Mutate_cpp
- MUCPP
- Accmut



- Stryker
- Testura
- Faultify
- VisualMutator



- PIT
- MuJava
- Judy
- Jumble
- Major



- Stryker
- Mutode



- Cosmic-ray
- Mumut
- MutPy
- Mutatest



- Infection

Java Mutation Test Tools

| | Updated | Mutant Generation Level | Traditional & object oriented Mutation Operators available | Mutant Code | Operator selection available | Output Generated |
|--------|---------|-------------------------|--|---------------------------------|------------------------------|---|
| MuJava | 2015 | Java Code and ByteCode | Both | Separate Source and Class Files | Yes | Live/Killed Mutants, Mutation Coverage |
| Judy | 2017 | Java Code | Both | In Memory | Yes | Live/Killed Mutants, Mutation Coverage |
| Jumble | 2015 | Byte Code | Just Traditional | In Memory | No | Mutation Coverage, Live Mutants |
| PIT | 2021 | Byte Code | Just Traditional | In Memory | Yes | Mutation Coverage, Live/Killed Mutants, Line Coverage |
| Major | 2019 | Java Code? | Just Traditional | In Memory, can be exported | Yes | Mutation Coverage, Live/Killed Mutants |

Why PIT?



Actively
developed
/supported

Compatible
with
Maven/Gradle

Easy to Use

IDE Plugins
(Eclipse,
IntelliJ, +)

Fast

PIT Mutation Operators (Mutators)

- Conditionals Boundary Mutator
- Increments Mutator
- Invert Negatives Mutator
- Math Mutator
- Negate Conditionals Mutator
- Return Values Mutator
- Void Method Calls Mutator
- Empty Returns
- False/True/Null Returns
- Remove Conditionals Mutator

```
if (a < b) {  
    // do something  
}
```



```
if (a <= b) {  
    // do something  
}
```

```
if (a == b) {  
    // do something  
}
```



```
if (a != b) {  
    // do something  
}
```

```
int a = b + c;
```



```
int a = b - c;
```

PIT Extra Features (Maven)

mutationsThreshold

Mutation Score Threshold At
Which To Fail Build

withHistory

Use Historical Files
To Speed Up
Analysis

maxMutationsPerClass

Specify Max Mutations On
A Class

targetClass

Specify Which
Classes Should Be
Mutated

threads

How Many
Threads To Use
When Making
Mutations
(default = 1)

mutators

Make Mutations
With List Of
Desired
Mutators

Demo



Questions?

CREDITS: This presentation template was created by [Slidesgo](#),
including icons by [Flaticon](#), and infographics & images by [Freepik](#)
Please keep this slide for attribution