Mutation Analysis

Testing that tests your tests

Mitchell Gale - November 2022

Common Testing Metrics

- Branch coverage
- Line coverage
- Statement coverage

Mutation Analysis

A way to test the quality of your test suite.

Creates small changes (mutations) to your project to add bugs

Common Testing Metrics

- Branch coverage
- Line coverage
- Statement coverage

These are good… sometimes. High coverage does not mean we will catch all potential bugs.

What can we do to Ensure good testing?

- Intentionally add bugs into your program and test if a test fails as a result of them
- Do this across your entire project and count how many bugs it caught and how many it missed
- This measures the test suite’s ability to find bugs

- We’ll called bugs that it caught, bugs that were “killed”
- Bugs that were not caught will be have “survived”

Mutants

- One change to the program that should cause a bug.

- Some examples of mutants
  - a = b + c -> a = b - c
  - If (a == b) -> if (a != b)

Mutants

ORIGINAL

```
int max (int a, int b)
{
    if (a > b)
    {
        return a;
    }
    return b;
}
```

Mutant

```
int max (int a, int b)
{
    if (a < b)
    {
        return a;
    }
    return b;
}
```

Mutants

ORIGINAL

```
int max (int a, int b)
{
    if (a > b)
    {
        return a;
    }
    return b;
}
```

Mutant

```
int max (int a, int b)
{
    if (a < b)
    {
        return a;
    }
    return b;
}
```

Test that would kill the mutant

```
assertEquals(max(10, 20), 20);
```

Test that would survive

```
assertEquals(max(20, 20), 20);
```

Mutants

ORIGINAL

```
int sumIfAEven (int a, int b)
{
    if (a%2 == 0)
    {
        return a + b;
    }
    return 0;
}
```

Mutant

```
int sumIfAEven (int a, int b)
{
    if (a%2 == 0)
    {
        return a - b;
    }
    return 0;
}
```

Mutants

ORIGINAL

```
int sumIfAEven (int a, int b)
{
    if (a%2 == 0)
    {
        return a + b;
    }
    return 0;
}
```

Mutant

```
int sumIfAEven (int a, int b)
{
    if (a%2 == 0)
    {
        return a - b;
    }
    return 0;
}
```

Test that would kill the mutant

```
assertEquals(sumIfAEven(6, 10), 16);
```

Test that would Survive

```
assertEquals(sumIfAEven(11, 12), 0);
```

Mutants

ORIGINAL

```
bool andFunc (bool a, bool b)
{
    if (a && b)
    {
        return true;
    }
    return false;
}
```

Mutant

```
bool andFunc (bool a, bool b)
{
    if (a || b)
    {
        return true;
    }
    return false;
}
```

Mutants

ORIGINAL

```
bool andFunc (bool a, bool b)
{
    if (a && b)
    {
        return true;
    }
    return false;
}
```

Mutant

```
bool andFunc (bool a, bool b)
{
    if (a || b)
    {
        return true;
    }
    return false;
}
```

Test that would kill the mutant

```
assertEquals(andFunc(false, true), false);
```

Test that would survive

```
assertEquals(andFunc(true, true), true);
```

Equivalent Mutants

ORIGINAL

```
int max (int a, int b)
{
    int max = a
    if (a > b)
    {
        max =  b;
    }
    return max;
}
```

Mutant

```
int max (int a, int b)
{
    int max = a
    if (max > b)
    {
        max = b;
    }
    return max;
}
```

Types of mutants

- Arithmetic Operations
  - Replace operators (+, -, /, *, %) with each other
- Boolean Relations
  - Replace operators (>, >=, ==, !=, <=, <) with each other
- Statement Deletion
  - Remove a statement

And more!

Mutation Testing Tools

- PiTest - Java
- Stryker - Javascript/C#/Scala

PiTest

```
For a gradle project to add PiTist:

plugins {
    id 'info.solidsoft.pitest' version '1.9.0'
}
```

```
pitest {
    targetClasses = ['org.opensearch.sql.*']
    pitestVersion = '1.9.0'
    outputFormats = ['XML', 'HTML']
    junit5PluginVersion = '1.0.0'
}
```

PiTest on OpenSearch SQL

```
==================================================================
- Statistics
==================================================================
>> Line Coverage: 5429/5799 (94%)
>> Generated 2896 mutations Killed 2453 (85%)
>> Mutations with no coverage 118. Test strength 88%
>> Ran 26997 tests (9.32 tests per mutation)
```

Issues with Mutation Analysis

- Commonly, only one mutant is modified at a time
- Manual checking of survived mutants for equivalent mutants

Mutation Analysis Summary

All about testing the quality of your tests

Mutants are small modifications to your project to implement bugs, and see if your test suite can find the bugs

Thank You!

Any Questions?