

Mutation testing

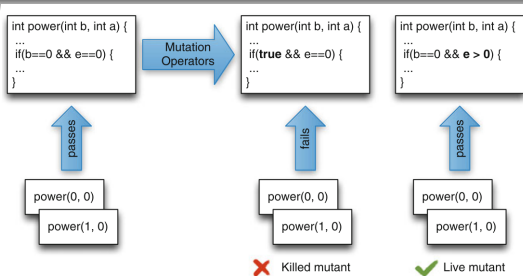
Angelo Gargantini

March 20, 2024

Mutation testing in brief

- 1 Mutation testing, also known as fault-based testing targets explicitly the software faults
- 2 To evaluate the tests, their quality is **NOT** measured in terms of coverage of structural elements
- 3 Instead, faults are injected in the code and tests are evaluated in terms of how many injected faults are detected

mutation process



Overview of the mutation testing process:

- Mutation operators are applied to the program under test to produce mutants.
- Tests are executed on all mutants; if a test fails on a mutant but passes on the original program, then the mutant is killed.
- If there is no test that kills the mutant, the mutant is alive, and likely reveals a weakness in the test suite.

An example - power method

```
1  int power(int b, int e){
2    if (e < 0) throw new Exception("Negative exponent")
3    if ((b == 0) && (e == 0)) throw new Exception("Undefined")
4    int r = 1;
5    while (e > 0){
6      r = r * b; e = e - 1;
7    }
8    return r;
9  }
10 @Test
11 public void testPowerOf2() {
12   int result = power(2, 2);
13   assertEquals(4, result);
14 }
```

mutant example

```
1  int power(int b, int e){
2    if (e < 0) throw new Exception("Negative exponent")
3    if ((true) && (e == 0)) throw new Exception("Undefined")
4    int r = 1;
5    while (e > 0){
6      r = r * b;
7      e = e - 1;
8    }
9    return r;
10 }
```

- Mutant by applying the COR operator (Conditional Operator Replacement) to line number 4
- The original test case `assertEquals(4, power(2, 2));` won't fail - the mutant is NOT killed - the fault is not found

Survived mutants

- 1 Survived mutants are a sign of weakness of the test suite (a fault that cannot be found)
- 2 New tests must be added
- 3 Note 1: (survived) mutants can be very many ...

mutant example

```
1 int power(int b, int e){
2   //... as before
3   if ((true) && (e == 0))
4     throw new Exception("Undefined");
5   //... as before
6 }
```

- To detect this fault we need a test in which we call power with $e = 0$ and $b \neq 0$. something like:

```
1 @Test public void test0PowerOf2 () {
2   int result = power(2, 0);
3   assertEquals(1, result);
4 }
```

- test0PowerOf2 will pass on the original code but it will fail with the mutant -> mutant is killed

Equivalent mutants

- 1 A limitation of mutation testing lies in the existence of equivalent mutants.
- 2 A mutant is equivalent when, although syntactically different, it is semantically equivalent to the original program.
- 3 There is NO test that kills an equivalent mutant - they will always survive
- 4 It is very difficult to say if a mutant has survived because a test is missing or because it is equivalent

equivalent mutants

```
int power(int b, int e){  
    if (e < 0)  
        throw new Exception("Negative exponent");  
    if ((b == 0) && (e == 0))  
        throw new Exception("Undefined");  
    int r = 1;  
    while (e != 0){  
        r = r * b; e = e - 1;  
    }  
    return r;  
}
```

- This mutant cannot be killed by any test since it is equivalent.

Tools for mutation testing

- There are many tools that perform mutation testing
- DEMO with PIT test: <https://pitest.org/>

Usare PIT con maven

- vedi <https://pitest.org/quickstart/maven/>
- aggiungi il jar nel pom e anche plugin
- fornisce due goal