Mutation Testing – A White Box Testing Technique

Dr. Durga Prasad Mohapatra

Dept. of Computer Science & Engineering

NIT Rourkela

White-box Testing

- Designing white-box test cases:
  - Requires knowledge about the internal structure of software.
  - White-box testing is also called structural testing.
  - In this unit we will study white-box testing.

White-Box Testing Methodologies

- There exist several popular white-box testing methodologies:
  - Statement coverage
  - Branch coverage
  - Condition coverage
  - MC/DC coverage
  - Path coverage
  - Data flow-based testing
  - Mutation testing

Introduction

- Mutation testing is the process of mutating some segment of code (putting some error in the code) and then, testing this mutated code with some data.
- If the test data is able to detect the mutations in the code, then the test data is quite good, otherwise we must focus on the quality of the test data.

Introduction cont...

- Mutation testing helps a user create test data by interacting with user to iteratively strengthen quality of test data.
- During mutation testing, faults are introduced into a program.
- Test data are used to execute these faulty programs with the goal of causing each faulty program to fail.
- Faulty programs are called mutants of the original program.

Introduction cont...

- A mutant is said to be killed when a test case causes it to fail.
- When this happens, the mutant is considered dead and no longer needs to remain in the testing process, since the faults represented by that mutant have been detected.

Introduction cont...

- The software is first tested:
  - using an initial testing method based on white-box strategies.

- After the initial testing is complete,
  - mutation testing is taken up.

Main Idea

- The idea behind mutation testing:
  - make a few arbitrary small changes to a program at a time.

Main Idea cont …

- Insert faults into a program:
  - Check whether the test suite is able to detect these.
  - This either validates or invalidates the test suite.

Main Idea cont …

- Each time the program is changed,
  - –it is called a mutated program
  - –the change is called a mutant.

Main Idea cont …

- A mutated program:
  - Tested against the full test suite of the program.
- If there exists at least one test case in the test suite for which:
  - A mutant gives an incorrect result,
  - Then the mutant is said to be dead.

Main Idea cont …

- If a mutant remains alive:
  - even after all test cases have been exhausted,
  - the test suite is enhanced to kill the mutant.
- The process of generation and killing of mutants:
  - can be automated by predefining a set of primitive changes that can be applied to the program.

Primitive changes

- The primitive changes can be:
  - Deleting a statement
  - Altering an arithmetic operator,
  - Changing the value of a constant,
  - Changing a data type, etc.

Traditional Mutation Operators

- Deletion of a statement
- Boolean:
  - Replacement of a statement with another

    eg. `==` and `>=`, `<` and `<=`
  - Replacement of boolean expressions with *true* or *false*

    eg. **`a || b`** with ***true***
- Replacement of arithmetic

  eg. `*` and `+`, `/` and `-`
- Replacement of a variable (ensuring same scope/type)

Underlying Hypotheses

- Mutation testing is based on the following two hypotheses:
  - **The Competent Programmer Hypothesis**
  - **The Coupling Effect**

Both of these were proposed by DeMillo *et al.*,1978

The Competent Programmer Hypothesis

- Programmers create programs that are close to being correct:
  - Differ from the correct program by some simple errors.

The Coupling Effect

- **Complex errors are caused due to several simple errors.**
- It therefore suffices to check for the presence of the simple errors

Types of mutants

- Primary Mutant
- Secondary Mutant

Primary Mutant

When the mutants are single modification of the initial program using some operators, they are called primary mutants.

Example

```
if(a>b)
        x=x + y;
else
        x = y;
Printf("%d", x);
……
```

We can consider following mutant for this example;

M1: x=x-y;
M2: x=x/y;
M3: x=x+1;
M4: printf("%d", y);

Example cont …

The results of the initial program and its mutant

Test Data	x	y	Initial program result	Mutant Result
TD1	2	2	4	0(M1)
TD2(x and y# 0)	4	3	7	1.4(M2)
TD3(y #1)	3	2	5	4(M3)
TD4(y #0)	5	2	7	2(M4)

Secondary Mutants

- When multiple levels of mutation are applied on the initial program, then, this class of mutant is called secondary Mutant.
- In this case, it is very difficult to identify the initial program form its mutants.

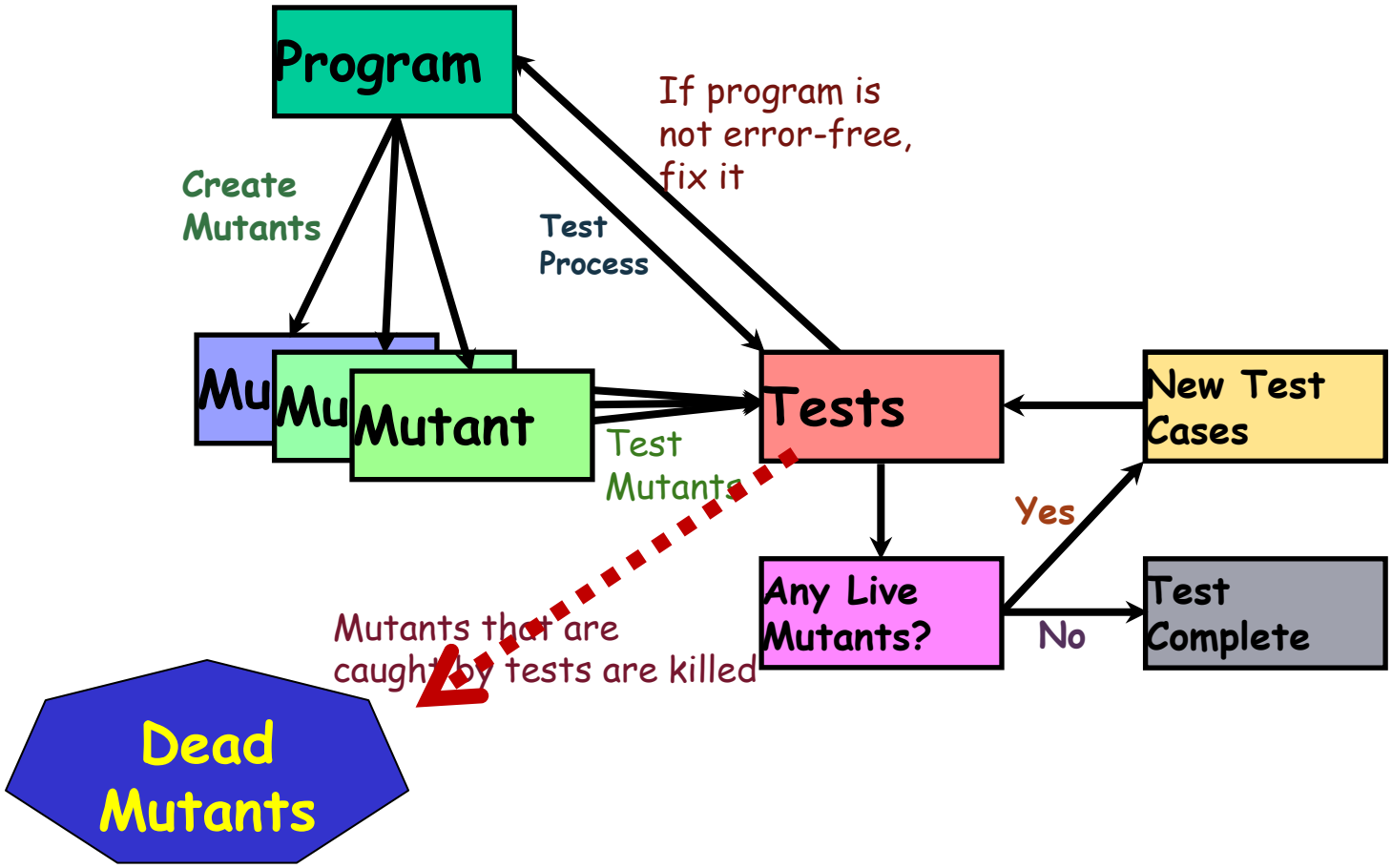
Example

```
If(a<b)
   c = a;
```

Mutants for this code may be as follows:

```
M1 : if(a<= b-1)
                c =a;
M2 : if(a+1<=b)
                c = a;
M3 : if(a == b)
          c =a+1;
```

Mutation Testing Process



Mutation Testing Process cont…

- Construct the mutants of a test program.
- Add test cases to the mutation system and check the output of the program on each test case to see if it is correct.
- If the output is incorrect, a fault has been found and the program must be modified and the process restarted.
- If the output is correct, that test case is executed against each live mutant.
- If the output of a mutant differs from that of the original program on the same test case, the mutant is assumed to be incorrect and is killed.

Mutation Testing Process cont…

- After each test case has been executed against each live mutant, each remaining mutant falls into one of the following two categories:

✓ One, the mutant is functionally equivalent to the original program. An equivalent mutant always produces the same output as the original program, so no test case can kill it.

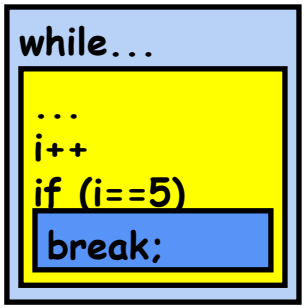
✓ Two, the mutant is killable, but the set of test cases is insufficient to kill it. In this case, new test cases need to be created, and the process iterates until the test set is strong enough to satisfy the tester.

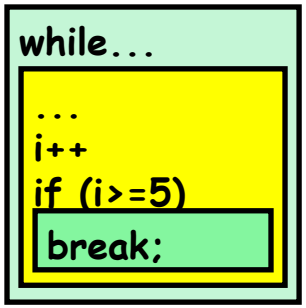
Mutation Testing Process cont…

- The mutation score for a set of test data is the percentage of non-equivalent mutants killed by that data.
- If the mutation score is 100%, then the test data is called mutation adequate.

Equivalent Mutants

- There may be surviving mutants that cannot be killed,
  - These are called Equivalent Mutants
- Although syntactically different:
  - These mutants are indistinguishable through testing.
- Therefore have to be checked 'by hand'





Disadvantages of Mutation Testing

- Equivalent mutants
- Computationally very expensive.
  - A large number of possible mutants can be generated.
- Certain types of faults are very difficult to inject.
  - Only simple syntactic faults introduced

Quiz 1

- Identify one advantage and one disadvantage of the mutation test technique.

Quiz 1: Solution

- Identify two advantages and two disadvantages of the mutation test technique.
- **Adv**:
  - Can be automated
  - Helps effectively strengthen black box and coverage-based test suite
- **Disadv:**
  - Equivalent mutants

Summary

- Presented basic concepts of mutation testing.
- Explained types of mutants.
- Discussed the mutation testing process.
- Presented some limitations of mutation testing.

31

References

1. Rajib Mall, Fundamentals of Software Engineering, (Chapter – 10), Fifth Edition, PHI Learning Pvt. Ltd., 2018.
2. Naresh Chauhan, Software Testing: Principles and Practices, (Chapter – 5), Second Edition, Oxford University Press, 2016.

Thank You