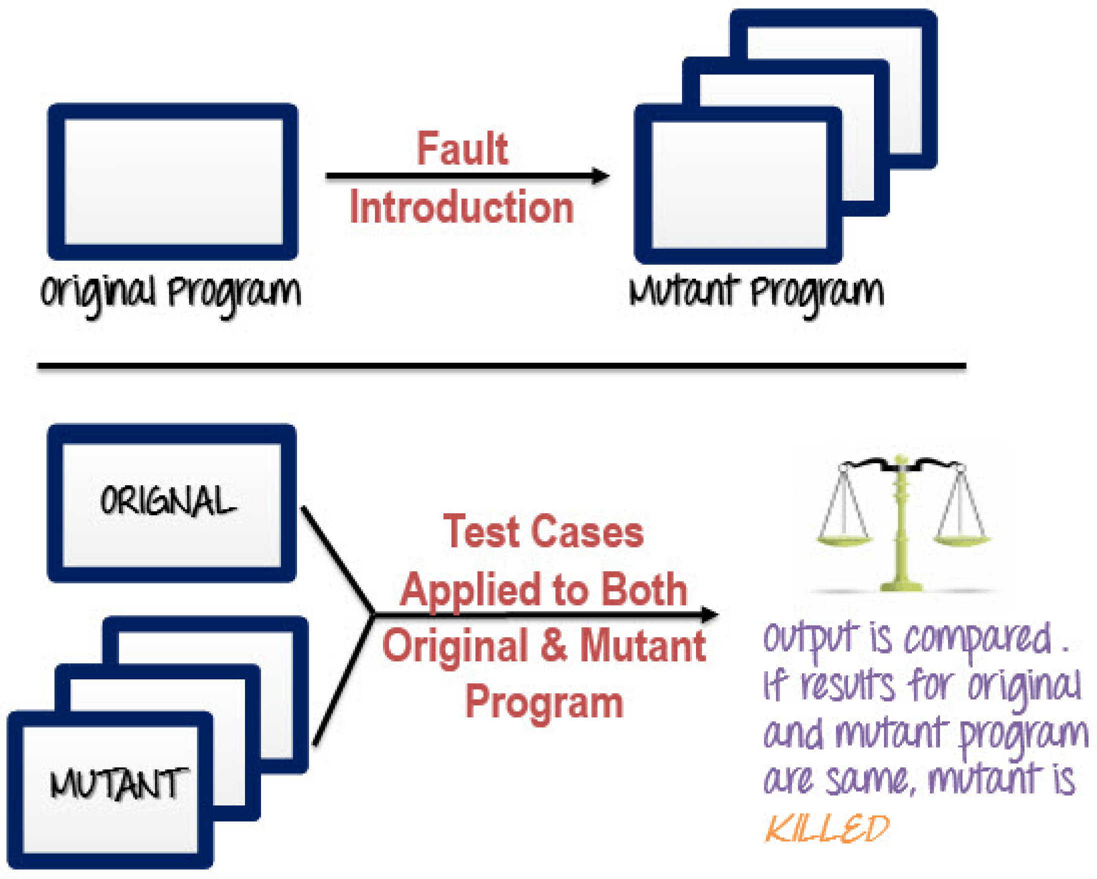
Unit- III
Mutation Testing

- Dr. Shivani Budhkar

Mutation Testing

- **Mutation Testing** is a type of software **testing** where we mutate (change) certain statements in the source code and check if the **test** cases are able to find the errors.
- It is a type of white box **testing** which is mainly used for unit **testing. T**he changes in mutant program are kept extremely small, so it does not affect the overall objective of the program.
- The goal of Mutation Testing is to assess the quality of the test cases which should be robust enough to fail mutant code. This method is also called as Fault based testing strategy as it involves creating fault in the program.
- Mutation was originally proposed in 1971 but lost fervor due to high costs involved. Now, again it has picked steam and is widely used for languages such as Java and XML.

How to execute mutation testing?



Steps to execute mutation testing:

- **Step 1**: Faults are introduced into the source code of the program by creating many versions called mutants. Each mutant should contain a single fault, and the goal is to cause the mutant version to fail which demonstrates the effectiveness of the test cases.
- **Step 2**: Test cases are applied to the original program and also to the mutant program. A test case should be adequate, and it is tweaked to detect faults in a program.
- **Step 3**: Compare the results of original and mutant program.
- **Step 4**: If the original program and mutant programs generate the same output, then that the mutant is killed by the test case. Hence the test case is good enough to detect the change between the original and the mutant program.
- **Step 5**: If the original program and mutant program generate different output, Mutant is kept alive. In such cases, more effective test cases need to be created that kill all mutants.

How to Create Mutant Programs?

- A mutation is nothing but a single syntactic change that is made to the program statement. Each mutant program should differ from the original program by one mutation.

Original Program	Mutant Program
If (x>y)	If(x<y)
Print "Hello"	Print "Hello"
Else	Else
Print "Hi"	Print "Hi"

What to change in a Mutant Program?

There are several techniques that could be used to generate mutant programs.

Operand replacement operators	Expression Modification Operators	Statement modification Operators
Replace the operand with another operand (x with y or y with x) or with the constant value.	Replace an operator or insertion of new operators in a program statement.	Programmatic statements are modified to create mutant programs.
Example- **If(x>y) replace x and y values** **If(5>y) replace x by constant 5**	Example- If(x==y) We can replace == into >= and have mutant program as If(x>=y) and inserting ++ in the statement If(x==++y)	Example- Delete the else part in an if-else statement Delete the entire if-else statement to check how program behaves Some of sample mutation operators: • GOTO label replacement • Return statement replacement • Statement deletion • Unary operator insertion (Like - and ++) • Logical connector replacement • Comparable array name replacement • Removing of else part in the if-else statement • Adding or replacement of operators • Statement replacement by changing the data • Data Modification for the variables • Modification of data types in the program

Automation of Mutation Testing:

- Mutation testing is extremely time consuming and complicated to execute manually. To speed up the process, it is advisable to go for automation tools. Automation tools reduce cost of testing as well.

- List of tools available -
- Ninja Turtles- .net mutation testing tool
- Mutagenesis- PHP mutation testing framework
- Jester- Mutation Testing Tool for Java

Types of Mutation Testing

- Mutation testing could be fundamentally categorized into 3 types– statement mutation, decision mutation, and value mutation
- **Statement Mutation** - developer cut and pastes a part of code of which the outcome may be removal of some lines
- **Value Mutation**- values of primary parameters are modified
- **Decision Mutation**- control statements are to be changed

Mutation Score:

- The mutation score is defined as the percentage of killed mutants with the total number of mutants.
- Mutation Score = (Killed Mutants / Total number of Mutants) * 100
- Test cases are mutation adequate if the score is 100%. Experimental results have shown that mutation testing is an effective approach for the measuring the adequacy of the test cases. But, the main drawback is that the high cost of generating the mutants and executing each test case against that mutant program.

Advantages and Disadvantages of Mutation Testing:

Advantages of Mutation Testing:

- It is a powerful approach to attain high coverage of the source program.
- This testing is capable comprehensively testing the mutant program.
- Mutation testing brings a good level of error detection to the software developer.
- This method uncovers ambiguities in the source code, and has the capacity to detect all the faults in the program.
- Customers are benefited from this testing by getting most reliable and stable system.

Disadvantages of Mutant testing:

- Mutation testing is extremely costly and time consuming since there are many mutant programs that need to be generated.
- Since its time consuming, it's fair to say that this testing cannot be done without an automation tool.
- Each mutation will have the same number of test cases than that of the original program. So, a large number of mutant programs may need to be tested against the original test suite.
- As this method involves source code changes, it is not at all applicable for black box testing.

Difference between Black Box Testing and White Box Testing.

#	Black Box Testing	White Box Testing
1	Black box testing is the Software testing method which is used to test the software without knowing the internal structure of code or program.	White box testing is the software testing method in which internal structure is being known to tester who is going to test the software.
2	This type of testing is carried out by testers.	Generally, this type of testing is carried out by software developers.
3	Implementation Knowledge is not required to carry out Black Box Testing.	Implementation Knowledge is required to carry out White Box Testing.
4	Programming Knowledge is not required to carry out Black Box Testing.	Programming Knowledge is required to carry out White Box Testing.
5	Testing is applicable on higher levels of testing like System Testing, Acceptance testing.	Testing is applicable on lower level of testing like Unit Testing, Integration testing.
6	Black box testing means functional test or external testing.	White box testing means structural test or interior testing.
7	In Black Box testing is primarily concentrate on the functionality of the system under test.	In White Box testing is primarily concentrate on the testing of program code of the system under test like code structure, branches, conditions, loops etc.
8	The main aim of this testing to check on what functionality is performing by the system under test.	The main aim of White Box testing to check on how System is performing.
9	Black Box testing can be started based on Requirement Specifications documents.	White Box testing can be started based on Detail Design documents.
10	The Functional testing, Behavior testing, Close box testing is carried out under Black Box testing, so there is no required of the programming knowledge.	The Structural testing, Logic testing, Path testing, Loop testing, Code coverage testing, Open box testing is carried out under White Box testing, so there is compulsory to know about programming knowledge.

Criteria	Black Box Testing	White Box Testing
Definition	Black Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is NOT known to the tester	White Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester.
Levels Applicable To	Mainly applicable to higher levels of testing : Acceptance Testing System Testing	Mainly applicable to lower levels of testing : Unit Testing Integration Testing
Responsibility	Generally, independent Software Testers	Generally, Software Developers
Programming Knowledge	Not Required	Required
Implementation Knowledge	Not Required	Required
Basis for Test Cases	Requirement Specifications	Detail Design