IT-415 Software Testing and Quality Analysis

Prof. Saurabh Tiwari
DA-IICT

20^{th} **April 2017**

Subject : Mutation Testing

Group Members :

1. Akash Koradia - 201401021

2. Dhruv Koshiyar - 201401136

3. Dhaval Prajapati - 201401138

Mutation Testing

Index :

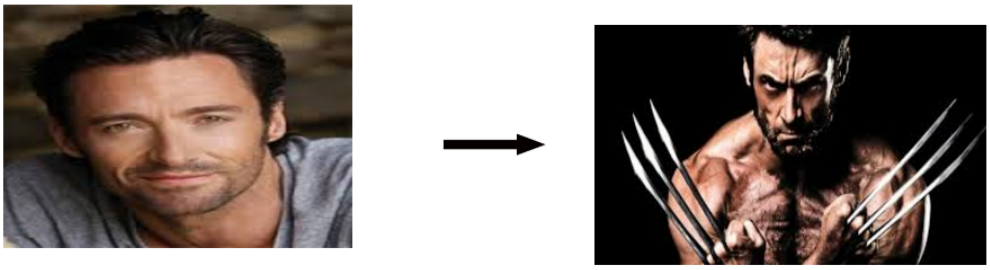
(b) Creation of Mutants
    i. First Order Mutants
    ii. Second Order Mutants
    iii. Third Order Mutants
    iv. Fourth Order Mutants
(c) Test Case and Test Suite Range for Testing
(d) Calculation of Mutant score for HOMs
(e) Execution Results of first order Mutants
(f) Result of Execution of Higher order Mutants
(g) Calculated Mutation Score for Each and Every Higher Order Mutants

9. Mutation Testing Tools
    (a) PIT
    (b) Ninja Turtles
    (c) Visual Mutator
    (d) Insure++
    (e) Mu-Java
10. Advantages and Disadvantages of Mutation testing
    (a) Advantages
    (b) Disadvantages
11. Conclusion
12. Contribution
13. Bibliography

Abstract :

Testing is the Important part of Software development Life Cycle(SDLC).There are different type of testing are used at different different phase of Software development life cycle.Here in this document we are going to study about the Mutation Testing.This document describes What is mutation testing, how can we implement it on any software or program,different types of mutation testing.We will take an example to explain it,which are different different tools available in market for Mutation testing and last we will talk about the pros and cons of mutation testing.So, let's jump in to the world of Mutation Testing.

What is Mutation ..?

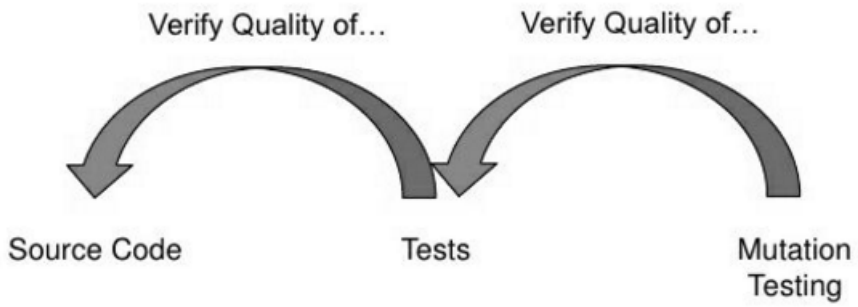


Normal Human Mutant

Mutation :

When some data or program is changed by making small amount of changes let say by applying insertion, deletion, duplication ,etc. in it and we can say it "Mutation", which could or could not change or affect the functionality of Program or Software or Application.By mutation we can add some programming errors in our Program or Software or Application. Each mutant program or code should contains at least one mutation.

What is Mutation Testing?



Testing :

It is the process of executing program or code for finding bugs.You can also say that it is the process of validating and verifying Program or code.

Mutation Testing :

Mutation testing is used in designing new software or Program.It evaluates the quality of existing software or Program tests. Basically this testing is about inserting defects (mutation) into original source code and check if our test suits can find them. Mutation Testing is different from testing technique like path or data-flow testing.It is somewhat related to white box testing, and we know that white box testing is used for unit testing. Through Mutation testing we can check the precision and efficiency of our code or program or software. It helps in evaluation or boosting the sufficiency of test suites.

The mutation testings aim is to create robust enough test cases which have the ability to kill the mutants. The other word for Mutation testing is **fault-based testing** , where we create mutants defining a new defect into the main program or software.

This technique is based on software faults it is aimed at finding out the completeness or acceptability of the tests by estimating the potential of a particular test suite in differentiating a particular program or mutant from another mutant. Normally,we can create mutant by defining a legal syntactic change in the software. Given a program "P" and corresponding test suite "T", the first step of the mutation testing is to construct the alternative programs - that is, mutants, (P). Then each mutant and the original program "P" has to be executed against each test case. If the output of mutant "Q" remains the same as the output of the original program "P", we consider "Q" as alive; otherwise, "Q" is considered to be dead or distinguished. "Q" remaining alive can be due to the consequence of two factors like:

1. insufficient test data
2. Equivalent Mutant

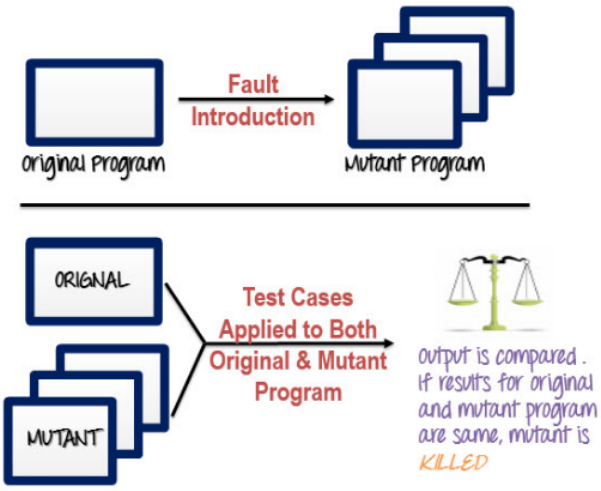
A mutant is said to be an equivalent mutant if it is equivalent to the original program then it will not contribute to the adequacy of "T" and should be eliminated. Then the mutants which are alive ,we have to create new test cases for that.The ratio of the killed mutants and remaining mutants is called mutation score:

$$\text{Mutation Score} = \frac{\text{(Dead Mutants)}}{\text{(Total Mutants)} - \text{(Equivalent Mutants)}}$$

- **Platform :**

  Mutation testing is used on many various platforms, including Java, C, C++,Python and Ruby.

- **How to execute mutation testing ?**



Following steps are for how to execute mutation testing:
Step 1 : First of all we will create faults in main source code of program which are called mutants.After that we will create mutant program by using these mutants(faults).Here our mutant program should be failed and demonstrating the effectiveness of the test cases which we applied.

Step 2 : We will apply the test case to the main program and mutant program. Applied test case should be sufficient and explicit.It is useful in finding mutants in a main source code.

Step 3 : We will check the results of main source code and mutant program and then

we will compare both results.

Step 4 : If we see the main program and mutant program produce different output,then mutant is killed.That means this test case detects the difference between Mutant Programs and main program.

Step 5 : If we see the original program and mutant programs producing the same output, That means mutant is alive(equivalent mutant).In this circumstance we have to make more effective test cases who can kill the mutants.

- **Mutation Operators :**

The operators who are applied on the main source code to create the mutants that are known as Mutation operators. Mutation operators can be classified into:

1. Method level mutation operators
2. Class - Level mutation operators

1. **Method- level Mutation Operators :**

   We can generate mutants by inserting, replacing or deleting using following mutation operators.
   Mainly there are 6 types of method- level mutation operators:

   - Arithmetic Operators
   - Relational Operators
   - Conditional Operators
   - Shift Operators
   - Logical Operators
   - Assignment Operators

   (a) **Arithmetic Operators :**

   Basically these operators perform mathematical computations on all floating-point numbers and integers.Following arithmetic operators supported in Java language are:

   - For binary: a + b (addition), a b (subtraction), a * b (multiplication), a / b (division) and a
   - For unary: +a (positive value), -a (negative value)
   - For short-cut: a++ (post-increment), ++op (pre-increment), a– (post-decrement), –a (pre-decrement)

(b) Relational Operators :

These operators compare the values of two operands. The relational operators supported in Java are:

- a> b(greater than), a>= b(greater than or equal to), a< b(less than), a<= b(less than or equal to), a== b(equal to) and a!= b(not equal to)

Two operators are required by these operators. So, only replacement of these operators is possible, not insertion, deletion.

(c) Conditional Operators (Bitwise operators) :

Conditional operators perform computations on the binary values of its operands. These operators exhibit short-circuit behavior. The conditional operators supported in Java are:

- For binary: a&& b(conditional AND), a|| b(conditional OR),
  a& b(bitwise AND), a| b(bitwise OR) and
  a^ b(bitwise XOR).
- For unary: !a (bitwise logical complement)

(d) Shift Operators :

Shift operators require two operands. They manipulate the bits of the first operand in the expression by either shifting them to the right or the left, second operand times. The shift operators supported in Java are:

- a>> b(signed right shift), a<< b(signed left shift) and
  a>>> b(unsigned right shift).

(e) Logical Operators :

Logical operators perform logical comparisons to produce a Boolean result for comparison statements. The logical operators supported in Java are:

- For binary: a& b(AND), a| b(OR) and a^ b(XOR).
- For unary: ~a (bitwise complement)

(f) **Assignment Operators :**

Assignment operators set the values of an operand. The shortcut assignment operators perform a computation on the right hand operand and then assign its value to the left hand operand. The assignment operators supported in Java are:

For shortcut:

- a+= b(addition assignment), a -= b (subtraction assignment), a*= b(multiplication assignment), a/= b(division assignment), a%= b(modulus assignment),
- a&= b(bitwise AND assignment), a|= b(bitwise OR assignment), a^= b(bitwise XOR assignment),
- a<<= b(right shift assignment), a>>= b(left shift assignment), a>>>= b(unsigned right shift assignment).

2. **Class - Level Mutation Operators**

We can change the program syntax by inserting, deleting or modifying the expressions under test using class-level operators.
Following are the 3 Class level mutation operators:

- Encapsulation
- Inheritance
- Polymorphism

(a) **Encapsulation :**

- Encapsulation is also known as data hiding.
- Basically Encapsulation is the property of an object which prevent the direct access of methods and data. In encapsulation, access levels can be defined to prevent the access of various objects by using access modifiers like public, private, protected, etc.
- In encapsulation, variables of any class will be hidden from another class (by using private) and they can only be accessed by their current class methods.
- Mutants are created through using operators like insert, delete or modify By using the variables and methods.

(b) **Inheritance :**

- By using Inheritance, same code can be reused.

- In inheritance, subclass can use all the attributes and methods of the superclass from which, it is derived.
- The mutants are produced by application of operators that deletes a hiding variable to check whether that variable is defined and that its accessibility in class and subclasses are correct.

(c) **Polymorphism :**

- In polymorphism , we have different methods with the same name but different parameters. Basically it allows objects to react differently to the same name method.
- We can create mutants using polymorphism operators to check whether the methods having the same name and number of parameters are accessible in a right way or not accessible.

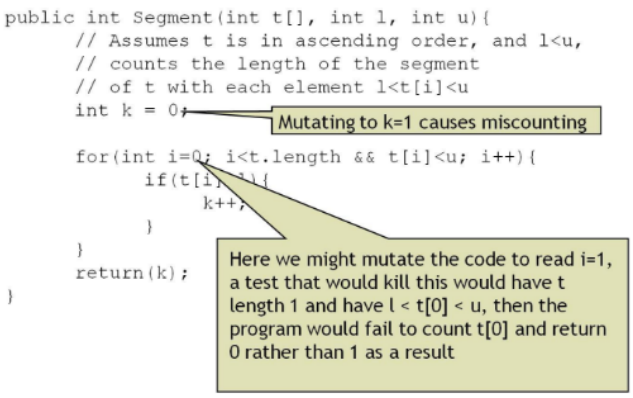
Mutation Testing Types :

There are basically three types of Mutation testing.

1. Value Mutation
2. Decision Mutation
3. Statement Mutation

1. Value Mutation :

   - It is obtained by changing values of variables in programs to find the errors in the programs.
   - Most commonly used examples are :
     (a) Modifying the values
     (b) Swapping values in initialisations. (accessing arrays i=0 to i=1).
   - The common approach is to increment or decrement constants, by one in an attempt to generate a one-off error (particularly common in checking loop bounds)

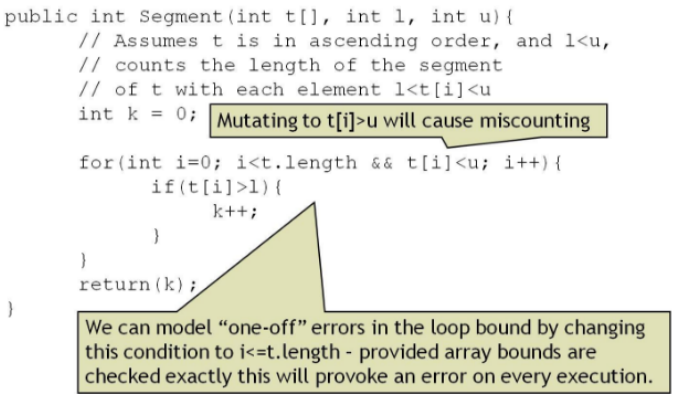


2. Decision Mutation :

   - It is obtained by changing the decision operators (Greater Than, less Than, less than and equal to, Greater Than and equal to, is not equal to, equal to equal to) to reflect the errors in the programs.

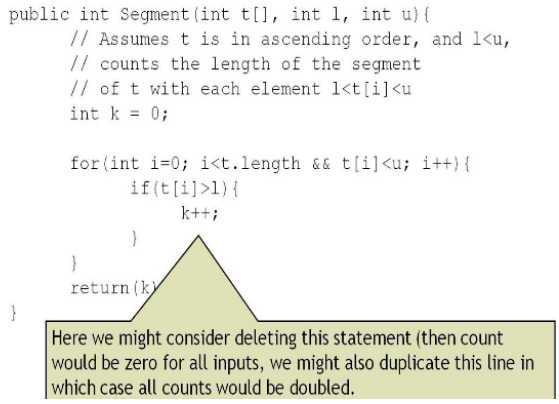
   - Typical examples are :

     (a) Generating one-off" errors by changing less than, greater than to less than or equal to,greater than or equal to respectively or Reverse way also. (this is common in checking loop bounds).
     (b) Generating confusion about larger and smaller, so changing greater than to less than or vice- versa. mistake in logical expressions e.g. mistaking precedence between AND and OR.



3. Statement Mutation :

- We can get this buy cut pasting or copy pasting to find the fault at line level. This may result in duplicate lines, deleted lines or change in the order of execution of a line of the code.
- Most commonly used example examples include :
  (a) Deleting a line of program or code
  (b) Duplicating a line of program or code
  (c) Modifying (permutation) the order of statements.



Types of Mutant :

There two types of Mutants.

1. First Order Mutants(FOMs)
2. Higher Order Mutants(HOMs)

These types of mutants are created by combining more than single faults (FOMs).

Original Program P	First Order Mutant (FOMs)	Higher Order Mutant(HOMs)
If(a>0 && b>0) return 1;	If(a>0 || b>0) return 1;	If (a<=0 || b>0) return 1;

Table 8.1: First Order and Higher Order Mutants

- FOMs are generated by changing only one operator. HOMs are generated by changing more than one operators (i.e. By applying more than one mutants). A combined HOM is harder to kill compared to the FOMs from which it is constructed.
- Higher Order Mutants decrease the number of created mutants, and also decrease the number of test cases require to kill these higher order Bugs. Thus, HOMs improve the efficiency of the mutation testing as well make the mutation testing practically possible.

Example for Mutation testing :

- Triangle program is considered.
- Here in input, we give the all sides' length of triangle and in output we check that by given lengths one can create triangle or not,if it valid then the answer would be equilateral, isosceles or scalene one.Here Program shows the source code of the program.
- The output is considered on the basis of its sides' lengths. Six different mutation positions are possible (program statements which are written in bold letters) which results in fifteen Second order mutants, ten Third Order Mutants, six Forth Order Mutants, four Fifth Order Mutants and one Sixth Order Mutant.

Program : Triangle :

Input: length of Three sides a, b, c
Output : Triangle Type

```
int triangle_type
if (a <= 0 || b <= 0 || c <= 0) then
        print INVALID
triangle_type = 0
if (a == b) then triangle_type = triangle_type + 1
if (a == c) then triangle_type = triangle_type + 2
if (b == c) then triangle_type = triangle_type + 3
if (triangle_type == 0) then
        if (a + b < c || a + c < b || b + c < a) then
                return INVALID
        else
                return SCALENE
if (triangle_type > 3) then
         return EQUILATERAL
if (triangle_type == 1 && a + b > c) then // Mutant position 1 and 2
        return ISOSCELES
else if (triangle_type == 2 && a + c > b) then // Mutant position 1 and 2
        return ISOSCELES
else if (triangle_type == 3 && b + c > a) then // Mutant position 5 and 6
        return ISOSCELES
return INVALID
```

Creation of Mutants :

1. First Order Mutants

```
if (triangle_type > 1 && a + b > c) then
        return ISOSCELES
else if (triangle_type == 2 && a + c > b) then
        return ISOSCELES
else if (triangle_type == 3 && b + c > a) then
        return ISOSCELES
```

2. Second Order Mutants

```
if (triangle_type > 1 && a + b <= c) then
        return ISOSCELES
else if (triangle_type == 2 && a + c > b) then
        return ISOSCELES
else if (triangle_type == 3 && b + c > a) then
        return ISOSCELES
```

3. Third Order Mutants

```
if (triangle_type > 1 && a + b > c) then
        return ISOSCELES
else if (triangle_type == 2 && a + c <= b) then
        return ISOSCELES
else if (triangle_type > 3 && b + c <= a) then
        return ISOSCELES
```

4. Fourth Order Mutants

```
if (triangle_type > 1 && a + b > c) then
        return ISOSCELES
else if (triangle_type == 2 && a + c <= b) then
        return ISOSCELES
else if (triangle_type > 3 && b + c <= a) then
        return ISOSCELES
```

Test Case and Test Suite Range for Testing :

Test Case	Test Case Range	Test Case value
TC1	a==b && a + b> c	(12, 12, 8)
TC2	a==c && a +b> c && a +c<=b	(12, 24, 12)
TC3	b==c && a +b>a && b+ c<=a	(24,12,12)
TC4	a==b&& a+ b>c	(12,12,6)

Calculation of Mutant score for HOMs :

For first order mutants, mutation score achieved is 1 as all the test cases are capable of detecting one or more mutants. So, the next step would be that test the Higher Order Mutants with same test cases and compute the mutant score for all Higher Order Mutants up to sixth order. No. of First Order Mutants (FOMs) killed = 1F1, 1F2, 1F3, 1F4, 1F5, 1F6 = 6

Execution Results of first order Mutants :

Mutants	TC1	TC2	TC3	TC4
$1F_1$	Killed	Killed	Killed	Survived
$1F_2$	Killed	Survived	Survived	Killed
$1F_3$	Survived	Survived	Killed	Survived
$1F_4$	Survived	Killed	Survived	Survived
1F5	Killed	Killed	Killed	Survived
$1F_6$	Killed	Killed	Killed	Survived

All HOMs are tested with the test cases. Some mutants are caught while remaining test cases shows the same output as the main program. Mutants who are caught , we can say that they are "Killed Mutants".And remaining mutants are called "Survived Mutants".

Result of Execution of Higher order Mutants :

Mutants	TC1	TC2	TC3	TC4
$2H_{12}$	Killed	Survived	Survived	Survived
$2H_{13}$	Killed	Killed	Killed	Survived
$2H_{14}$	Killed	Killed	Killed	Survived
$2H_{15}$	Killed	Killed	Killed	Survived
$2zH_{16}$	Killed	Killed	Killed	Survived
$2H_{23}$	Killed	Survived	Killed	Killed
$2H_{24}$	Killed	Killed	Survived	Killed
$2H_{25}$	Killed	Survived	Survived	Killed
$3H_{134}$	Killed	Killed	Killed	Survived
$3H_{135}$	Killed	Killed	Killed	Survived
$3H_{136}$	Killed	Killed	Killed	Survived
$3H_{145}$	Killed	Killed	Killed	Survived
$3H_{146}$	Killed	Killed	Killed	Survived
$3H_{156}$	Killed	Killed	Killed	Survived
$4H_{1234}$	Killed	Survived	Survived	Survived
$4H_{1235}$	Killed	Survived	Killed	Survived
$4H_{1236}$	Killed	Survived	Killed	Survived
$5H_{12346}$	Killed	Killed	Survived	Survived
$5H_{13456}$	Killed	Killed	Killed	Survived
$6H_{123456}$	Killed	Survived	Survived	Survived

Calculated Mutation Score for Each and Every Higher Order Mutants :

1. MSO (2) = No. of 2HOMsKilled / (TotalNo.of 2HOMs No. of Equivalent 2HOMs) = 13 /15 = 0.86
2. MSO (3) = No. of 3HOMsKilled/ (TotalNo.of 3HOMs No. of Equivalent 3HOMs) = 10 /10 = 1
3. MSO (4) = No. of 4HOMsKilled / (TotalNo.of 4HOMs No. of Equivalent 4HOMs) = 6 / 6 = 1
4. MSO (5) = No. of 5HOMsKilled / (TotalNo.of 5HOMs No. of Equivalent 5HOMs) = 3 / 3 = 1
5. MSO (6) = No. of 6HOMsKilled / (TotalNo.of 6HOMs No. of Equivalent 6HOMs) = 1/1 = 1

Mutation Testing Tools :

Mutation testing is very costly, time consuming and complicated. So it becomes a very tiresome process, when it comes to do manually.So, with the help of automation tools we can perform this process very easily. We can use different Mutation testing tools for various language.They have their own ability to test the higher level mutation testing. So, a brief Introduction about popular tools which user are using for mutation testing is given below.

1. PIT :-

   - PIT is the most popular mutation testing tool for java.
   - It focuses on Mutation testing and Coverage criteria both.It gives better reports compared to other java tools like Jester and it is twice faster than jumble.
   - It can also work with frameworks like mocking Frameworks like Mockito, JMock, EasyMock, PowerMock and JMockit which other tools like jumble and Javalanche can not do.

2. Ninja Turtles :-

   - Ninja Turtles uses .net framework for mutation testing.
   - Ninja turtle can support intra-method mutations like Statement deletion, Replacing branch points based on a boolean condition, Adjusting the inclusivity of boundary conditions, Replacing arithmetic, bitwise operators, Replacing reads and writes local variables of the same type with each other Replacing reads from parameters.

3. Visual Mutator :-

   - It is integrated as development processes of visual studio IDE.
   - In the process of mutation testing it provide us way to measure quality of test suits.
   - we can create First order mutants by using built-in and custom operators.
   - It has the ability to view modied code fragments in C and IL7 languages.

- It has the user interactive interfaces. By this you can view details about any mutant right after the beginning of the mutation testing process.
- you can see the mutation score and an information about passed and failed tests.
- It has the Option to write detailed results to an XML le.

4. Insure++ :-

   - Insure++ is commercial tool. It support c and c++ language for Quick Mutation testing.
   - It can help us to find defects in the code using the state-of-the-art.
   - Insure++ mainly focuses on finding potential equivalent mutants. The main reason behind doing this is that we assume that if our test cases are able to crack potential equivalent mutants then, they can identify normal faults in our Main Program.

5. Mu-Java :-

   - It is the oldest Mutation testing tool for java, predating even J-Unit.It is developed by two universities, KAIST in South Korea and GMU in USA.

There are also another good tools for different different language like for c there are tools like Milu and Nester.For python there is Pester.For java There are lots of tools are available in market.so this tools are basically targeted for Object oriented language particular. In which most popular are PIT,Jumble and Insurance.Which we have shown above.

Advantages and Disadvantages of Mutation Testing :

- Advantages :-
  - It is powerful technique for fulfill Coverage criteria of source code.
  - It can help the software developer by good level of error detection.
  - By mutation testing, you can uncovers ambiguities in the source code,
  - It has the capacity to detect all the faults in the program.
  - It is the most comprehensive technique for test a program.
  - It is the most reliable and stable technique, so customers are satisfied with it.
  - By this technique you can do effective test development.
  - Mutation testing improves program quality.
  - Debugging becomes more easier by this technique.
  - You can maintain the product easily by this technique.

- Disadvantage :-
  - Mutation testing generates lots of mutant programs,so it would become very costly and time consuming.
  - Since it is very tiresome and time consuming we must have to use automation tools.
  - This large number of mutants will have to tested against test suits of original program.
  - It is Difficult to implement complex mutations.
  - It requires skilled testers who must have knowledge of programming.
  - In this technique we do change in source code. So we can not apply Black box testing on it.

Conclusion :

Mutation word actually comes from biology.It suggest the changes in our Genes.Here we are using this word in technical term. We change some part of code and new program called Mutant.This processed called Mutation. That's why we can say that Mutation testing is a "Misnomer". we can say that it is an analytical method rather than testing method.

By Using Mutation Testing we can check the behavior of our application through test cases. Researches show that mutation score is better measure to detect real fault compare to code coverage. Quality of test cases can not be measured by complete branch and line coverage. If we blindly throw some test cases and they pass by luck.But it will not ensure that in future if some rough test cases come then our Application will survive or not. First step to increase the quality of code is that using Boundary value analysis, then Mutation testing comes into picture.So, at the end, mutation testing enforce us for thinking about different possibilities which can occur and by creating new test-cases we can ensure that our code will survive in any type of tough circumstances.

Contribution :

- Dhaval Prajapati :- Abstract, Mutation testing tools, Advantage and Disadvantage of Mutation Testing, Conclusion,Scribe Document Making.
- Dhruv Koshiyar :- What is the Mutation Testing?, Platform, How to execute Mutation Testing ?,Making of Scribe Document.
- Akash Koradiya :- Mutation Testing Operators, Mutation Testing Types, Type of Mutant, Example of Mutation Testing.

Bibliography :

- http://www.softwaretestingclass.com/what-is-mutation-testing-tools-testing-types-and-its-challenges/
- https://visualmutator.github.io/web/
- http://pitest.org/
- https://www.techopedia.com/definition/20905/mutation-testing
- http://www.guru99.com/mutation-testing.html
- http://www.exforsys.com/tutorials/testing-types/mutation-testing.html
- https://www.techopedia.com/definition/20905/mutation-testing
- http://www0.cs.ucl.ac.uk/staff/M.Harman/PastMScProjects2005/MaryumUmar.pdf
- https://pdfs.semanticscholar.org/625f/532cae2b4a2968165f713ec789e0d7bba5d2.pdf
- http://shodhganga.inflibnet.ac.in/bitstream/10603/8030/18/18$_chapter$