Fault vs Failure

Feature	Fault	Failure
Definition	An imperfection or deficiency in a work product where it does not meet its requirements or specifications.	An event in which a component or system does not perform a required function within specified limits.
Reason	Irregularity in logic	System error
Prevention	Reviewing test documents	Reviewing requirement and specification

Testing vs Debugging

Testing	Debugging
Testing is the process to find bugs and errors.	Debugging is the process of correcting the bugs found during testing.
Testing is done by the tester.	Debugging is done by developer
Testing can be manual or automated	Debugging is always manual
Testing is a stage of the software development life cycle (SDLC)	Testing is not a stage of the software development life cycle (SDLC)
Testing includes validation and verification	Debugging includes matching symptoms with cause
Design knowledge is not needed in testing	Design knowledge is needed in debugging

Verification vs Validation

Verification	Validation
It includes checking documents, design, codes and programs.	It includes testing and validating the actual product.
Verification is the static testing.	Validation is the dynamic testing.
It does not include the execution of the code.	It includes the execution of the code.
Methods used in verification are reviews, walkthroughs, inspections, etc.	Methods used in validation are Black Box Testing, White Box, etc.
It can find the bugs in the early stage of the development.	It can only find the bugs that could not be found by the verification process.
It comes before validation.	It comes after verification.

Alpha and Beta testing

Alpha Testing	Beta Testing
Alpha testing involves both the white box and black box testing.	Beta testing involves black-box testing.
Alpha testing is performed at developer site.	Beta testing is performed at customer site
Alpha testing requires a testing environment	Beta testing doesn't require a testing environment
Reliability and security are not checked in alpha testing.	Reliability and security are checked in beta testing.
Alpha testing involves multiple test cycles	Beta testing involves few test cycles

Test case and Test suite

Feature	Test Case	Test Suite
Definition	Test case describes an input description and an expected output description.	The collection of test cases is called a test suite.
Function	Tests a single functionality	Tests multiple functionalities
Dependency	Test cases run independently of each other	It can be dependent on other Test Suites
Identification	Identified by a unique test case ID.	Identified by a unique test suite ID.

Domain testing

It is a Functional Testing technique in which the output of a system is tested with a minimal number of inputs to ensure that the system does not accept invalid and out of range input values. Domain specific knowledge is needed in order to test a software system.

GUI testing

GUI Testing is the process for ensuring proper functionality of the graphical user interface for a specific application. It evaluates the design of elements. It may be either manual or automatic and are often performed by third-party companies, rather than developers or end users.

Error and incident

Error: A human action that produces an incorrect result. eg. incorrect syntax, improper calculation of values, etc.

Incident: An incident is the symptom associated with a failure that alerts the user to the occurrence of a failure. e.g. hardware failure

Development and regression testing

S.No.	Development Testing	Regression Testing
1.	We write test cases.	We may use already available test cases.
2.	We want to test all portions of the source code.	We want to test only modified portion of the source code and the portion affected by the modifications.
3.	We do development testing just once in the lifetime of the software.	We may have to do regression testing many times in the lifetime of the software.
4.	We do development testing to obtain confidence about the correctness of the software.	We do regression testing to obtain confidence about the correctness of the modified portion of the software.
5.	Performed under the pressure of release date.	Performed in crisis situations, under greater time constraints.
6.	Separate allocation of budget and time.	Practically no time and generally no separate budget allocation.
7.	Focus is on the whole software with the objective of finding faults.	Focus is only on the modified portion and other affected portions with the objective of ensuring the correctness of the modifications.
8.	Time and effort consuming activity (40% to 70%).	Not much time and effort is consumed as compared to development testing.

Explain the boundary value analysis technique with a suitable example.

Boundary Value Analysis tests the boundary values of valid and invalid partitions. Basic idea is to use input variable values at their:

- Minimum
- Just above minimum
- A nominal value
- Just below their maximum
- Maximum

A function of n variables, boundary value analysis yields 4n+1 test cases

Consider a two input program to multiply numbers. x and y lies between the following intervals:

$100 \leq x \leq 300$

$100 \leq y \leq 300$

Test Case	X	Y	Expected Output
1	200	100	20000
2	200	101	20200
3	200	200	40000
4	200	299	59800
5	200	300	60000
6	100	200	20000
7	101	200	20200
8	299	200	59800
9	300	200	60000

What is slice based testing? How can it improve testing? Explain the concept with the help of an example and write. test cases accordingly.

Given a program P, a program graph G(P) in which statements are numbered, and a set V of variables in P, the slice on the variable set V at statement n, S(V, n) is defined as the set of node numbers of all statements fragments in P prior to n that contribute to the values of variables in V at statement fragment n.

Slicing improves testing as:

1. It acts as an aid to debugging tools
2. It helps in implementing information flow analysis
3. It decides which statements will execute in parallel

Example: Consider the program fragment

13. Input(locks)

14. While NOT(locks=-1)

15. Input(stocks, barrels)

16. totalLocks= totalLocks+ locks

17. totalStocks= totalStocks+ stocks

18. totalBarrels= totalBarrels+ barrels

19. Input(locks)

20. EndWhile

The slices on locks are:

S1: S(locks,13) = {13}

S2: S(locks,14) = {13,14,19,20}

S3: S(locks,16) = {13,14,19,20}

S4: S(locks,19) = {19}

What is mutation testing? What is the purpose of mutation score? Why are higher order mutants not preferred?

Mutation testing, also known as code mutation testing, is a form of white box testing in which testers change specific components of an application's source code to ensure a software test suite can detect the changes.

Mutation score is the score associated with a test suite and its mutants.

$$\text{Mutation Score} = \frac{\text{Number of mutants killed}}{\text{Total number of mutants}}$$

Where, total number of mutants is equal to number of killed mutant plus number of live mutants.

Higher order mutants are not preferred because they are difficult to manage, control and trace. They have increased complexity and decreased relevance.

Do we perform regression testing before the release of the software?

Yes, regression testing is performed before releasing software to ensure its stability and quality.

How is risk analysis used in testing? How can we prioritize test cases using risk factor?

Risk analysis is a process of identifying the potential problems and then assigning a ‘probability of occurrence of the problem’ value and ‘impact of that problem’ value for each identified problem. Both of these values are assigned on a scale of 1 (low) to 10 (high). A factor ‘risk exposure’ is calculated for every problem which is the product of ‘probability of occurrence of the problem’ value and ‘impact of that problem’ value. The risks may be ranked on the basis of its risk exposure. These values may be calculated on the basis of historical data, past experience, intuition and criticality of the problem.

Table 7.4. Risk analysis table of 'University Registration System'

S. No.	Potential Problems	Probability of occurrence of problem	Impact of that Problem	Risk Exposure
1.	Issued password not available	2	3	6
2.	Wrong entry in students detail form	6	2	12
3.	Wrong entry in scheme detail form	3	3	9
4.	Printing mistake in registration card	2	2	4
5.	Unauthorised access	1	10	10
6.	Database corrupted	2	9	18
7.	Ambiguous documentation	8	1	8
8.	Lists not in proper format	3	2	6
9.	Issued login-id is not in specified format	2	1	2
10.	School not available in the database	2	4	8

What are different types of structural testing techniques? Discuss any one technique with the help of example.

- Control Flow Testing
- Data Flow Testing
- Slice Based Testing
- Mutation Testing

Mutation Testing

The results of the program were affected by the change and any test case detects it. If this happens, the mutant is called a **killed mutant**.

The results of the program are not changed and any test case of test suit does not detect the mutation. The mutant is called a **live mutant**.

Mutation score is the score associated with a test suite and its mutants.

$$\text{Mutation Score} = \frac{\text{Number of mutants killed}}{\text{Total number of mutants}}$$

Where, total number of mutants is equal to number of killed mutant plus number of live mutants.

Example:

Consider the program to find largest of three numbers. The test suite selected by a testing technique is given as:

```
1   void main()
2   {
3   float A,B,C;
4   clrscr();
5   printf("Enter number  1:\n");
6   scanf("%f", &A);
7   printf("Enter number  2:\n");
8   scanf("%f", &B);
9   printf("Enter number  3:\n");
10  scanf("%f", &C);
11
12  if(A>C) {
13
                    printf("The largest number is: %f\n",A);
14                  }
15  else    {
16                  printf("The largest number is: %f\n",C);
17                  }
18  }
19  else   {
20  if(C>B) {
21                  printf("The largest number is: %f\n",C);
22                  }
23  else    {
24                  printf("The largest number is: %f\n",B);
25                  }
26  }
27  getch();
28  }
```

Test case	A	B	C	Expected output
1	6	10	2	10
2	10	6	2	10
3	6	2	10	10
4	6	10	20	20

Solution:

Mutant No.	Line no.	Original line	Modified Line
M_1	11	if(A>B)	if (A<B)
M_2	11	if(A>B)	if(A>(B+C))
M_3	12	if(A>C)	if(A<C)
M_4	20	if(C>B)	if(C=B)
M_5	16	printf("The Largest number is:%f\n",C);	printf("The Largest number is:%f\n",B);

Actual output of mutants 1 and 2 using given test suite

Test case	A	B	C	Expected output	Actual output
1	6	10	2	10	6
2	10	6	2	10	6
3	6	2	10	10	10
4	6	10	20	20	20

Test case	A	B	C	Expected output	Actual output
1	6	10	2	10	10
2	10	6	2	10	10
3	6	2	10	10	10
4	6	10	20	20	20

Actual output of mutant 3 and 4 using given test suite

Test case	A	B	C	Expected output	Actual output
1	6	10	2	10	10
2	10	6	2	10	2
3	6	2	10	10	6
4	6	10	20	20	20

Test case	A	B	C	Expected output	Actual output
1	6	10	2	10	10
2	10	6	2	10	10
3	6	2	10	10	10
4	6	10	20	20	10

Actual output of mutant 5 using given test suite

Test case	A	B	C	Expected output	Actual output
1	6	10	2	10	10
2	10	6	2	10	10
3	6	2	10	10	2
4	6	10	20	20	20

$$\text{Mutation Score} = \frac{\text{Number of mutants killed}}{\text{Total number of mutants}}$$

$$= \frac{4}{5}$$

$$= 0.8$$

The mutant M2 is live in the example. We may have to write a specific test case to kill this mutant.

Additional test case

Test case	A	B	C	Expected output
5	**10**	**5**	**6**	**10**

Revised Test suite

Test case	A	B	C	Expected output
1	6	10	2	10
2	10	6	2	10
3	6	2	10	10
4	6	10	20	20
5	**10**	**5**	**6**	**10**

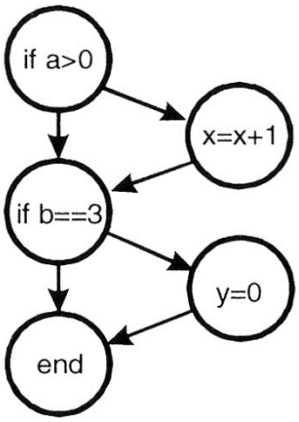
Show with the help of an example that a very high level of statement coverage does not mean that the program is defect-free.

Consider the following code snippet:

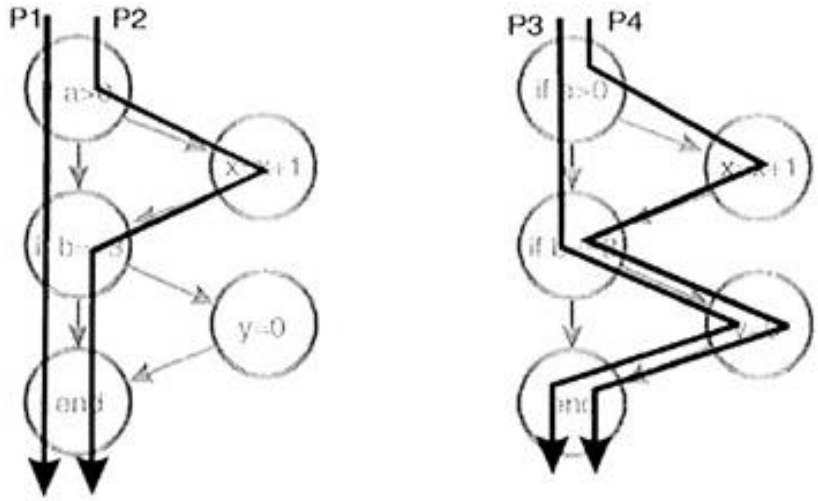
if (a>0) {x=x+1;}

if (b==3) {y=0;}

This code can be represented in graphical form as:



These two lines of code implement four different paths of execution:

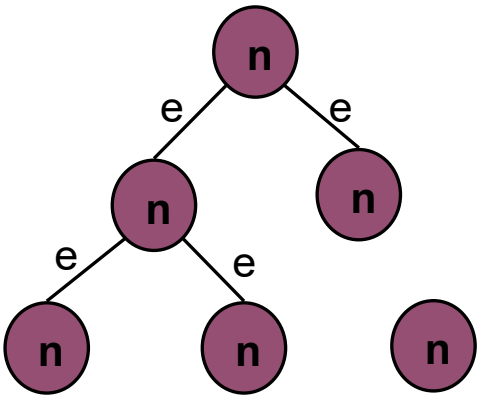


While a single test case is sufficient to test every line of code in this module (ex. a=6 and b=3), it is apparent that this level of coverage will miss testing many paths.

Explain the following with examples:

Incidence matrix

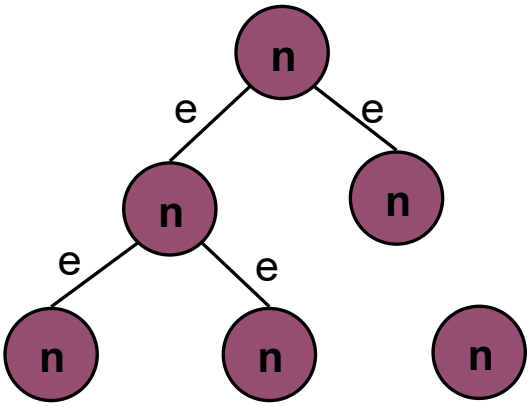
$$a(i,\ j) = \begin{cases} 1 \text{ if } j^{th} \text{ edge } e_j \text{ is incident on } i^{th} \text{ node } n_i \\ 0 \text{ otherwise} \end{cases}$$



$$\begin{array}{c} \\ n_1 \\ n_2 \\ n_3 \\ n_4 \\ n_5 \\ n_6 \end{array} \begin{array}{c} \begin{array}{cccc} e_1 & e_2 & e_3 & e_4 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Adjacency matrix

$$a(i,\ j) = \begin{cases} 1 \text{ if there is an edge between nodes } n_i \text{ and } n_j \\ 0 \text{ otherwise} \end{cases}$$

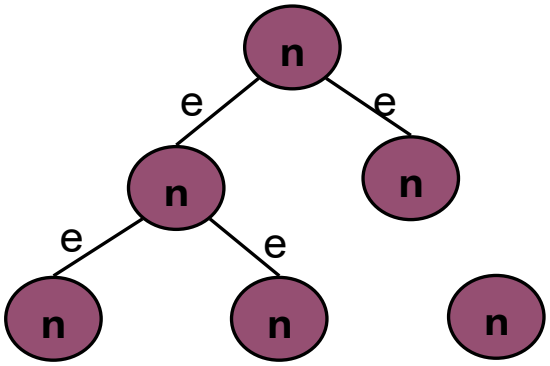


$$\begin{array}{c} \\ n_1 \\ n_2 \\ n_3 \\ n_4 \\ n_5 \\ n_6 \end{array} \begin{array}{c} \begin{array}{cccccc} n_1 & n_2 & n_3 & n_4 & n_5 & n_6 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Paths

A path in a graph is a sequence of adjacent nodes where nodes in sequence share a common edge or sequence of adjacent pair of edges where edges in sequence share a common node.

Consider the graph shown in below and find its path.



S.No.	Paths from initial node to final node	Sequence of nodes	Sequence of edges
1	n_1 to n_4	n_1, n_2, n_4	e_1, e_3
2	n_1 to n_5	n_1, n_2, n_5	e_1, e_4
3	n_1 to n_2	n_1, n_2	e_1
4	n_1 to n_3	n_1, n_3	e_2
5	n_2 to n_4	n_2, n_4	e_3
6	n_2 to n_5	n_2, n_5	e_4
7	n_2 to n_1	n_2, n_1	e_1
8	n_3 to n_1	n_3, n_1	e_2
9	n_4 to n_2	n_4, n_2	e_3
10	n_4 to n_1	n_4, n_2, n_1	e_3, e_1

S.No.	Paths from initial node to final node	Sequence of nodes	Sequence of edges
11	n_5 to n_2	n_5, n_2	e_4
12	n_5 to n_1	n_5, n_2, n_1	e_4, e_1
13	n_2 to n_3	n_2, n_1, n_3	e_1, e_2
14	n_3 to n_2	n_3, n_1, n_2	e_2, e_1
15	n_4 to n_3	n_4, n_2, n_1, n_3	e_3, e_1, e_2
16	n_3 to n_4	n_3, n_1, n_2, n_4	e_2, e_1, e_3
17	n_4 to n_5	n_4, n_2, n_5	e_3, e_4
18	n_5 to n_4	n_5, n_2, n_4	e_4, e_3
19	n_5 to n_3	n_5, n_2, n_1, n_3	e_4, e_1, e_2
20	n_3 to n_5	n_3, n_1, n_2, n_5	e_2, e_1, e_4

How path is it different from an independent path?

An independent path in a graph is a path that has at least one new node or edge in its sequence from the initial node to its final node.

Class Testing

Class testing includes activities associated with verifying that the implementation of a class corresponds exactly with the specification for that class. If an implementation is correct, then each of the class's instances should behave properly. The code for a class can be tested effectively by review or by executing test cases. Classes are usually tested by developing a test driver that creates instances of the class and sets up a suitable environment around those instances to run a test case.

Static Testing Tools

Static software testing tools perform analysis of the programs without executing them. They also find the source code which is hard to test and maintain. Types of static testing tools are: Complexity analysis tools, Syntax and semantic analysis tools, Flow graph generator tools, Code comprehension tools and Code inspectors

Dynamic Testing Tools

Dynamic software testing tools select test cases and execute the program to get the results. They also analyse the results and find reasons for failures of the program. They are used after the implementation of the program and also test non-functional requirements like efficiency, performance, reliability, etc. Types of dynamic testing tools are: Coverage analysis tools, Performance testing tools and Functional / Regression Testing Tools.

Characteristics of Modern Tools

1. It should use one or more testing strategy for performing testing on host as well as on target platform.
2. It should support GUI based test preparation.
3. It should provide complete code coverage and create test documentation in various formats
4. These tools should able to adopt the underlying hardware.
5. It should be easy to use.
6. It should provide a clear report on test case, steps, test case status