**5.1. Categories of Testing**

Testing is broadly categorised into two namely Static and Dynamic Testing. Dynamic Testing is again categorised into Black Box and White Box Testing.

**Static Testing**

Static Testing is testing of software work products either manually or by usage of tools. In this case, the work products are not executed.  It starts early in the life cycle and is a part of verification process.  It does not require a computer as there will not be execution of the software work products.

The following are the static testing techniques

* Reviews
* Walkthroughs
* Inspections

Of the above mentioned techniques, Peer reviews and Walkthroughs are informal reviews whereas Inspection is a formal review process.

**Formal Technical Review Objectives**

* Uncover errors in function, logic, or implementation for any representation of the software
* Verify that the software under review meets its requirements
* Ensure that the software has been represented according to predefined standards
* Achieve software that is developed in a uniform manner
* Make projects more manageable
* Serve as a training ground, enabling junior engineers to observe different approaches to software analysis, design, and implementation
* Serves to promote backup and continuity because a number of people become familiar with parts of the software that they may not have otherwise seen

**Formal Technical Reviews**

* Involves 3 to 5 people (including reviewers)
* Advance preparation (no more than 2 hours per person) required
* Duration of review meeting should be less than 2 hours
* Focus of review (walkthrough or inspection) is on a discrete work product
* Review leader organizes the review meeting at the producer's request
* Reviewers ask questions that enable the producer to discover his or her own error (the product is under review not the producer)
* Producer of the work product walks the reviewers through the product
* Recorder writes down any significant issues raised during the review
* Reviewers decide to accept or reject the work product and whether to require additional reviews of product or not

**Formal Technical Review Guidelines**

1. Review the product not the producer.
2. Set an agenda and maintain it.
3. Limit rebuttal and debate.
4. Enunciate problem area, but don’t attempt to solve every problem noted.
5. Take written notes.
6. Limit number of participants and insist on advance preparation.
7. Develop a checklist for each product that is likely to be reviewed.
8. Allocate resources and schedule time for all reviewers.
9. Conduct meaningful training for all reviewers.
10. Review your early reviews,

In a recap of above techniques ,

**Informal - Peer Review:**

* Peer Reviews are documented and uses a defect detection process    that has peers and technical specialist as part of the review process.
* The Review process does not involve management participation.
* It is usually led by trained moderator who is NOT the author.
* The report is prepared with the list of issues that needs to be addressed.

**Informal - Walkthrough:**

* It is not a formal process. It is led by the authors.
* Author guides the participants through the document according to his or her thought process to achieve a common understanding and to gather feedback.
* Useful for the people if they are not from the software discipline, who are not used to or cannot easily understand software development process.
* Especially useful for higher level documents like requirement specification, etc.

**Formal – Inspection:**

* It is the most formal review type. It is led by the trained moderators.
* During inspection the documents are prepared and checked thoroughly    by the reviewers before the meeting.
* A separate preparation is carried out during which the product is examined and the defects are found.
* The defects found are documented in a logging list or issue log.
* A formal follow-up is carried out by the moderator applying exit criteria.

External reference - <http://www.cs.toronto.edu/~sme/CSC444F/slides/L09-Inspections.pdf>

**Dynamic Testing**

Testing the software by dynamically executing the application is termed as dynamic testing. Dynamic testing is a method of assessing the feasibility of a software program by giving input and examining output (I/O).  The dynamic method requires the code be compiled and run.  It comes under the validation.

Dynamic Testing Consists of Two Types:

1) White-Box testing

2) Black-Box testing

**White Box Testing**

White Box testing can be outlined as below.

* Testing that takes into account the internal mechanism of a system or component.
* Checks out the paths and statements, branches that are executed
* Requires knowledge of the internal code.

For example, Unit testing can be termed as a White box testing where in the developer would test the paths, branches that are part of code is conforming to the functionality the program is supposed to do.***‎***

**Key Note**:  White box tests can be designed only after Component –level design ( source code ) is available.  The logical details of the program must be available

**White-Box Testing Questions**

The following questions need to be asked by testers to themselves before designing the white box test cases.

* Can it be guaranteed that all independent paths within a module will be executed at least once?
* Can all logical decisions be exercised on their true and false branches?
* Will all loops execute at their boundaries and within their operational bounds?
* Can internal data structures be exercised to ensure their validity?

**Black Box Testing**

Black box testing is the testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to the selected inputs and execution conditions. It is conducted to evaluate the compliance of a system or component with specified functional requirements. Requirements are the only test basis and knowledge of the internal code is not required.

Some of the Black box testing types:

* Integration Testing
* System Testing
* Acceptance Testing

These testing levels will be detailed later in the course.

**Black-Box Testing Questions**

Following would be some of the questions that a black box tester to consider before designing the test cases.

* How is functional validity tested?
* How is system behavior and performance tested?
* What classes of input will make good test cases?
* Is the system particularly sensitive to certain input values?
* How are the boundaries of a data class isolated?
* What data rates and data volume can the system tolerate?
* What effect will specific combinations of data have on system operation?

**5.2. Testing Techniques**

**What is Testing technique**

Testing technique is a procedure for selecting or designing tests. It is based on the functional or structural model of the software. The techniques have a high success rate of finding faults. It is a way of deriving the test cases which also helps in measuring the testing effort quantitatively.

**Why to use Testing Techniques**

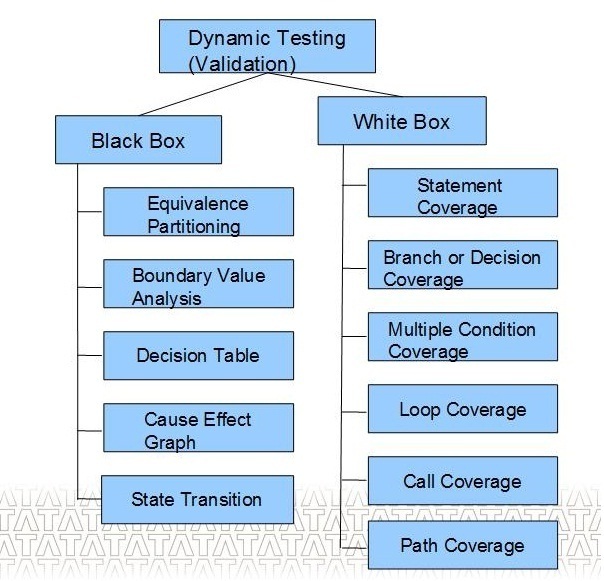
To ease the development of test cases, we use different testing techniques that have evolved as per different business scenarios. The reason why testing techniques are used is because there would be large number of test scenarios and finite time which would make it impossible to test.

Hence the usage of software testing techniques reduce the number of tests to be run whilst providing the sufficient coverage of system under test.

**Categories of Testing Techniques**

There are two types of test design techniques:

* Black Box techniques
* White Box techniques



**Black Box Testing Techniques**

In black box testing, the inner structure and design of the test object is unknown or not considered. The test cases are derived from the specification, or they are already available as part of the specification (“specification by example”). Black boxtechniques are also called specification based because they are based on specifications (of requirements). A test with all possible input data combinations would be a complete test, but this is unrealistic due to the enormous number of combinations. During test design, a reasonable subset of all possible test cases must be selected. There are several methods to do that, and they will be shown in the following sections.

**Equivalence partitioning technique**

Equivalence partitioning technique is to partitioning the input domain of a program into a finite number of classes [sets], to identify a minimal set of well selected test cases to represent these classes.

There are two types of input equivalence classes, valid and invalid.

Equivalence class technique can significantly reduce the number of test cases that must be created and executed

Illustration for EP

EP may be best explained with an example of a function which has the pass parameter "month" of a date. The valid range for the month is 1 to 12, standing for January to December. This valid range is called a partition. In this  example there are two further partitions of invalid ranges.

The first invalid partition would be <= 0 and the second

invalid partition would be >= 13.



**Boundary Value Analysis technique**

A selection technique in which test data are chosen to lie along "boundaries“ of the   input domain [or output range] classes, data structures, procedure    parameters is known as Boundary Value Analysis Technique.

Choices often include maximum, minimum, and trivial Values .This technique mainly focus on the boundaries of the input.

Illustration for BVA

If the same example of a function which has the pass parameter "month" of a date

Valid Class is  1<= month <=12

Invalid Class 1 is  month <1

Invalid Class 2 is month >12

When compared to EP, which says select any test case within a range and any on either side of it , in BVA the emphasis is on the ‘edges’.

1 and 12 for the ‘edges’ of the Valid class

0 and 13 for the Invalid class

**White Box Testing Techniques**

**Statement coverage technique**

It is a technique to satisfy the criterion that eachstatement in a program to be executed at least onceduring program testing. Coverage is 100 percentage whena set of test cases causes every program statement to beexecuted at least once.

The chief disadvantage of statement coverage is that it isinsensitive to some control structures.

Example

1    int select ( int a[], int n, int x)  
2    {  
3          int i = 0;  
4          while ( i < n && a[i] < x )  
5          {  
6                if (a[i] < 0)  
7                   a[i] = - a[i];  
8                    i++;  
9           }  
10           return 1;  
11        }

One test case n=1, a[0]=-7, x=9 covers everything ,

Flow 1 - > 2 - > 3 - > 4 - > 5 - > 6 - > 7 - > 8 - > 9 - > 10 - > 11

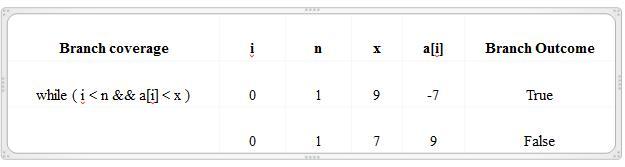
**Branch Coverage technique**

In Branch Coverage Technique execution of decisions is considered rather than execution of each statement. The result of decisions determines which statement will be executed next.

Test every Decision Output both Then and Else part of IF, all possibilities for a CASE statement, loops i.e execution of the loop body, bypassing of the loop body and return to the beginning of the loop.  
Example:

1    int select ( int a[], int n, int x)  
2    {  
3          int i = 0;  
4          while ( i < n && a[i] < x )  
5          {  
6                if (a[i] < 0)  
7                   a[i] = - a[i];  
8                    i++;  
9           }  
10           return 1;  
11    }

Test Data:



Flow A : 1 - > 2 - > 3 - > 4 - > 5 - > 6 - > 7 - > 8 - > 9 - > 10 - > 11

Flow B : 1 - > 2 - > 3 - > 4 - > 10 - > 11

**Multiple Condition Coverage technique**

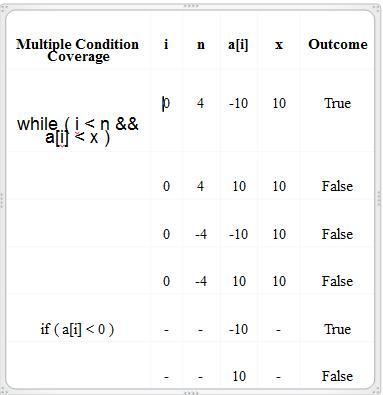
A test coverage criteria which requires enough test cases such that all possible combinations of condition outcomes in each decision, and all points of entry, are invoked at least once.

A large number of test cases may be required for full multiple condition coverage.

Example:

1    int select ( int a[], int n, int x)  
2    {  
3          int i = 0;  
4          while ( i < n && a[i] < x )  
5          {  
6                if (a[i] < 0)  
7                   a[i] = - a[i];  
8                    i++;  
9           }  
10           return 1;  
11    }

Test Data:



**Loop Coverage technique**

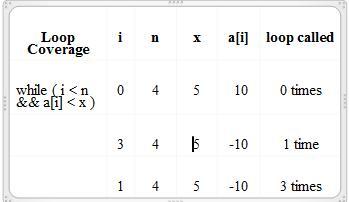
A test coverage criteria which checks whether loop body executed zero times, exactly once or more than once.

Example:

main ( )  
{

  int i, n, a[10],x;  
  printf (“Enter the values”);  
  scanf (“%d %d %d %d”, &i, &n, &a[i], &x);  
   while ( i < n && a[i] < x )  
   {  
      if (a[i] < 0)  
      a[i] = - a[i];  
      i++;  
    }  
    printf (“%d” , a[i] );  
}

Test Data:



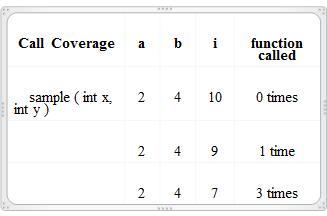
**Call Coverage technique**

A test coverage criteria which checks whether function called zero times, exactly once or more than once. Since probability of failure is more in function calls, each function call is executed.

Example:

    main ( )  
    {  
    int a, b, i ;  
    printf (“Enter the value of a, b, i”);  
    scanf (“ %d %d %d “, &a ,&b, &i);  
    if ( i  < 10 )  
    {  
    sample ( a, b);  
    i = i + 1;  
    }  
    }  
    sample ( int x , int y )  
    {  
    If ( x > 10 )  
        x = x + y ; break ;   
    if  ( y > 10 )  
        y = y + x ; break ;  
    }

Test Data:



**Path Coverage technique**

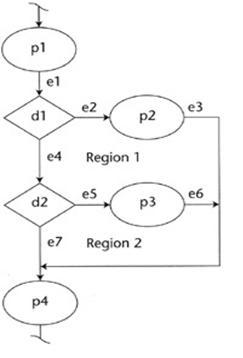
Testing to satisfy coverage criteria that each logical path through the program be tested. Often paths through the program are grouped into a finite set of classes. One path from each class is then tested

General coverage requires executing all paths, number of paths may be infinite if there are loops

Example:

Linear Independent Paths

Path 1 -> p1 – d1 – d2 – p4  
Path 2 -> p1 – d1 – p2 – p4  
Path 3 -> p1 – d1 – d2 – p3 – p4



Sample Program:

1    sample ( int x , int y )

    {  
2        If ( x > 10 )  
3        x = x + y ; break ;   
4        if  ( y > 10 )  
5         y = y + x ; break ;  
    }  
6    printf (“%d %d”, x , y);

