**EXPERIMENT NO. 07**

**AIM:** To study various Software Testing Methods.

**THEORY:**

***Software Testing***

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the [software](https://en.wikipedia.org/wiki/Software) to allow the business to appreciate and understand the risks of software implementation. Test techniques include the process of executing a program or application with the intent of finding [software bugs](https://en.wikipedia.org/wiki/Software_bug) (errors or other defects).

Software testing involves the execution of a software component or system component to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test:

* meets the requirements that guided its design and development,
* responds correctly to all kinds of inputs,
* performs its functions within an acceptable time,
* is sufficiently usable,
* can be installed and run in its intended [environments](https://en.wikipedia.org/wiki/Operating_environment), and
* Achieves the general result its stakeholder’s desire.

As the number of possible tests for even simple software components is practically infinite, all software testing uses some strategy to select tests that are feasible for the available time and resources. As a result, software testing typically (but not exclusively) attempts to execute a program or application with the intent of finding [software bugs](https://en.wikipedia.org/wiki/Software_bug) (errors or other defects). The job of testing is an iterative process as when one bug is fixed; it can illuminate other, deeper bugs, or can even create new ones. Software testing can provide objective, independent information about the quality of software and risk of its failure to users and/or sponsors.

Software testing can be conducted as soon as executable software (even if partially complete) exists. The [overall approach to software development](https://en.wikipedia.org/wiki/Software_development_process) often determines when and how testing is conducted. For example, in a phased process, most testing occurs after system requirements have been defined and then implemented in testable programs. In contrast, under an [Agile approach](https://en.wikipedia.org/wiki/Agile_software_development), requirements, programming, and testing are often done concurrently. There are different methods that can be used for software testing. Here, we are going to know about Black Box Testing, White Box Testing and Some Special Testing.

**1. Black-Box Testing**

The technique of testing without having any knowledge of the interior workings of the application is called black-box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, while performing a black-box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

The following table lists the advantages and disadvantages of black-box testing.

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| * Well suited and efficient for large code segments. * Code access is not required. * Clearly separates user's perspective from the developer's perspective through visibly defined roles. * Large numbers of moderately skilled testers can test the application with no knowledge of implementation, programming language, or operating systems. | * Limited coverage, since only a selected number of test scenarios is actually performed. * Inefficient testing, due to the fact that the tester only has limited knowledge about an application. * Blind coverage, since the tester cannot target specific code segments or error-prone areas. * The test cases are difficult to design. |

***1.1 Equivalence Class Testing***

Equivalence Partitioning also called as equivalence class partitioning. It is abbreviated as ECP. It is a software testing technique that divides the input test data of the application under test into each partition at least once of equivalent data from which test cases can be derived.

An advantage of this approach is it reduces the time required for performing testing of software due to less number of test cases.

Example:

The Below example best describes the equivalence class Partitioning:

Assume that the application accepts an integer in the range 100 to 999

Valid Equivalence Class partition: 100 to 999 inclusive.

Non-valid Equivalence Class partitions: less than 100, more than 999, decimal numbers and alphabets/non-number

***1.2 Boundary Value Analysis Testing***

Boundary value analysis is often called as a part of stress and negative testing. It’s widely recognized that input values at the extreme ends of input domain cause more errors in system. More application **errors occur at the boundaries** of input domain. ‘Boundary value analysis’ testing technique is used to identify errors at boundaries rather than finding those exist in center of input domain.

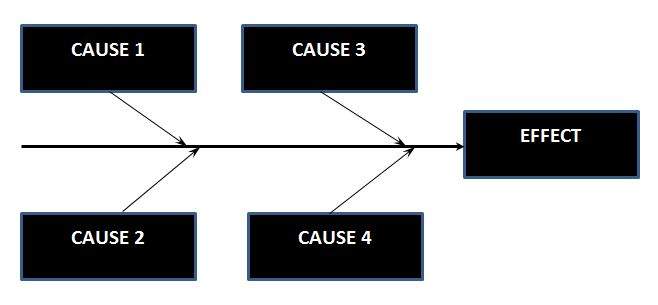
Boundary value analysis is a next part of Equivalence partitioning for designing test cases where test cases are selected at the edges of the equivalence classes.

* **Test cases for input box accepting numbers between 1 and 1000 using Boundary value analysis:**
* Test cases with test data exactly as the input boundaries of input domain i.e. values 1 and 1000 in our case.
* Test data with values just below the extreme edges of input domains i.e. values 0 and 999.
* Test data with values just above the extreme edges of input domain i.e. values 2 and 1001.

***1.3 Cause Effects Testing***

Cause Effect Graph is a black box testing technique that graphically illustrates the relationship between a given outcome and all the factors that influence the outcome.

It is also known as Ishikawa diagram as it was invented by Kaoru Ishikawa or fish bone diagram because of the way it looks.



## Circumstances - under which Cause-Effect Diagram used

* To identify the possible root causes the reasons for a specific effect, problem, or outcome.
* To relate the interactions of the system among the factors affecting a particular process or effect.
* To analyze the existing problems so that corrective action can be taken at the earliest.

## Benefits

* It helps us to determine the root causes of a problem or quality using a structured approach.
* It uses an orderly, easy-to-read format to diagram cause-and-effect relationships.
* It indicates possible causes of variation in a process.
* It identifies areas, where data should be collected for further study.
* It encourages team participation and utilizes the team knowledge of the process.
* It Increases knowledge of the process by helping everyone to learn more about the factors at work and how they relate.

## Steps for drawing cause-Effect Diagram:

* **Step 1:**Identify and Define the Effect
* **Step 2:**Fill in the Effect Box and Draw the Spine
* **Step 3:**Identify the main causes contributing to the effect being studied.
* **Step 4:**For each major branch, identify other specific factors which may be the causes of the EFFECT.
* **Step 5:**Categorize relative causes and provide detailed levels of causes.

***1.4 Orthogonal Array Testing***

Orthogonal array testing is a systematic and statistical way of a black box testing technique used when number of inputs to the application under test is small but too complex for an exhaustive testing.

## Orthogonal Array Testing Characteristics:

* OAT is a systematic and statistical approach to pair wise interactions.
* Executing a well-defined and a precise test is likely to uncover most of the defects.
* 100% Orthogonal Array Testing implies 100% pair wise testing.

## Example:

If we have 3 parameters, each can have 3 values then the possible Number of tests using conventional method is 3^3 = 27

While the same using OAT, it boils down to 9 test cases.

***1.5 Regression Testing***

Regression testing is the process of testing changes to computer programs to make sure that the older programming still works with the new changes. Regression testing is a normal part of the program development process and, in larger companies, is done by code testing specialists. Test department coders develop code test scenarios and exercises that will test new units of code after they have been written. These test cases form what becomes the test bucket. Before a new version of a software product is released, the old test cases are run against the new version to make sure that all the old capabilities still work. The reason they might not work is because changing or adding new code to a program can easily introduce errors into code that is not intended to be changed.Regression testing ensures that little changes don't break software.

**2. White-Box Testing**

White-box testing is the detailed investigation of internal logic and structure of the code. White-box testing is also called **glass testing** or **open-box testing**. In order to perform **white-box** testing on an application, a tester needs to know the internal workings of the code.

The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.

The following table lists the advantages and disadvantages of white-box testing.

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| * As the tester has knowledge of the source code, it becomes very easy to find out which type of data can help in testing the application effectively. * It helps in optimizing the code. * Extra lines of code can be removed which can bring in hidden defects. * Due to the tester's knowledge about the code, maximum coverage is attained during test scenario writing. | * Due to the fact that a skilled tester is needed to perform white-box testing, the costs are increased. * Sometimes it is impossible to look into every nook and corner to find out hidden errors that may create problems, as many paths will go untested. * It is difficult to maintain white-box testing, as it requires specialized tools like code analyzers and debugging tools. |

***2.1 Path Testing***

Path Testing is a structural testing method based on the source code or algorithm and NOT based on the specifications. It can be applied at different levels of granularity.

## Path Testing Assumptions:

* The Specifications are Accurate
* The Data is defined and accessed properly
* There are no defects that exist in the system other than those that affect control flow

## Path Testing Techniques:

* Control Flow Graph (CFG) **-**The Program is converted into Flow graphs by representing the code into nodes, regions and edges.
* Decision to Decision path (D-D) **-** The CFG can be broken into various Decision to Decision paths and then collapsed into individual nodes.
* Independent (basis) paths **-**Independent path is a path through a DD-path graph which cannot be reproduced from other paths by other methods.

***2.2 Condition Testing***

Condition coverage is also known as Predicate Coverage in which each one of the Boolean expression have been evaluated to both TRUE and FALSE.

## Example

if ((A || B) && C)

{

<< Few Statements >>

}

else

{

<< Few Statements >>

}

## Result

In order to ensure complete Condition coverage criteria for the above example, A, B and C should be evaluated at least once against "true" and "false".

So, in our example, the 3 following tests would be sufficient for 100% Condition coverage testing.

A = true | B = not equal | C = false

A = false | B = true | C = true

A = false | B = false | C = not equal

***2.3 Data Flow Testing***

Data flow testing is a family of test strategies based on selecting paths through the program's control flow in order to explore sequences of events related to the status of variables or data objects. Dataflow Testing focuses on the points at which variables receive values and the points at which these values are used.

## Advantages of Data Flow Testing:

Data Flow testing helps us to pinpoint any of the following issues:

* A variable that is declared but never used within the program.
* A variable that is used but never declared.
* A variable that is defined multiple times before it is used.
* Deal locating a variable before it is used.

***2.4 Loop Testing***

Loop testing a white box testing technique performed to validate the loops. There are four kinds of loops as mentioned below:

* Simple Loops
* Nested Loops
* Concatenated Loops
* Unstructured Loops

## What is tested in Loop Testing?

* Loops testing reveal loops initialization problems.
* By going through the loop once, the uninitialized variables in the loop can be determined.
* Testing can also fix loop repetition issues.
* Loops can also reveal capacity/performance bottlenecks.

**3. Smoke Testing**

Smoke Testing is a testing technique that is inspired from hardware testing, which checks for the smoke from the hardware components once the hardware's power is switched on. Similarly in Software testing context, smoke testing refers to testing the basic functionality of the build. If the Test fails, build is declared as unstable and it is NOT tested anymore until the smoke test of the build passes. Some of its features are:

* Identifying the business critical functionalities that a product must satisfy.
* Designing and executing the basic functionalities of the application.
* Ensuring that the smoke test passes each and every build in order to proceed with the testing.
* Smoke Tests enables uncovering obvious errors which saves time and effort of test team.
* Smoke Tests can be manual or automated.

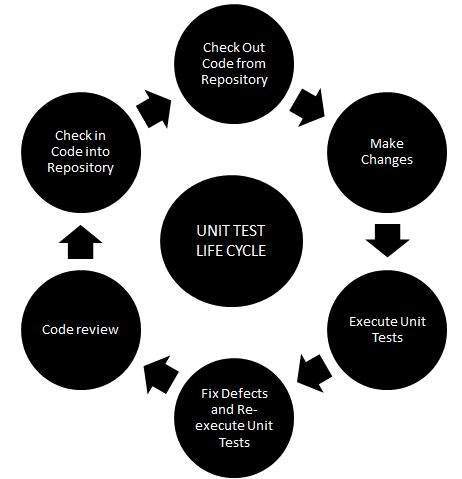
**4. Some Special Testing**

***4.1 Unit Testing***

Unit testing is a testing technique using which individual modules are tested to determine if there are any issues by the developer himself. It is concerned with functional correctness of the standalone modules. The main aim is to isolate each unit of the system to identify, analyze and fix the defects. Advantages are as follows.

* Reduces Defects in the newly developed features or reduces bugs when changing the existing functionality.
* Reduces Cost of Testing as defects are captured in very early phase.
* Improves design and allows better refactoring of code.
* Unit Tests, when integrated with build gives the quality of the build as well.

Unit Testing Life Cycle:

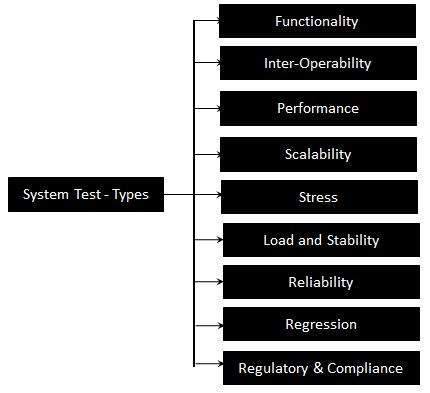


***4.2 System Testing***

System Testing (ST) is a black box testing technique performed to evaluate the complete system the system's compliance against specified requirements. In System testing, the functionalities of the system are tested from an end-to-end perspective.

System Testing is usually carried out by a team that is independent of the development team in order to measure the quality of the system unbiased. It includes both functional and Non-Functional testing.

Types of System Tests:



As a rule, system testing takes, as its input, all of the "integrated" software components that have passed [integration testing](https://en.wikipedia.org/wiki/Integration_testing) and also the software system itself integrated with any applicable hardware system(s). The purpose of integration testing is to detect any inconsistencies between the software units that are integrated together (called *assemblages*) or between any of the *assemblages* and the hardware. System testing is a more limited type of testing; it seeks to detect defects both within the "inter-assemblages" and also within the system as a whole.

***4.3 Integration Testing***

Upon completion of unit testing, the units or modules are to be integrated which gives raise to integration testing. The purpose of integration testing is to verify the functional, performance, and reliability between the modules that are integrated. Integration Strategies,

* Big-Bang Integration
* Top Down Integration
* Bottom Up Integration
* Hybrid Integration

In Big Bang approach, most of the developed modules are coupled together to form a complete software system or major part of the system and then used for integration testing. The Big Bang method is very effective for saving time in the integration testing process. However, if the test cases and their results are not recorded properly, the entire integration process will be more complicated and may prevent the testing team from achieving the goal of integration testing. A type of Big Bang Integration testing is called Usage Model testing. Usage Model testing can be used in both software and hardware integration testing.

Bottom-Up Testing is an approach to integrated testing where the lowest level components are tested first, then used to facilitate the testing of higher level components. The process is repeated until the component at the top of the hierarchy is tested.

All the bottom or low-level modules, procedures or functions are integrated and then tested. After the integration testing of lower level integrated modules, the next level of modules will be formed and can be used for integration testing. This approach is helpful only when all or most of the modules of the same development level are ready. This method also helps to determine the levels of software developed and makes it easier to report testing progress in the form of a percentage.

Top-Down Testing is an approach to integrated testing where the top integrated modules are tested and the branch of the module is tested step by step until the end of the related module.

Sandwich Testing is an approach to combine top down testing with bottom up testing.

The main advantage of the Bottom-Up approach is that bugs are more easily found. With Top-Down, it is easier to find a missing branch link.

Risky - hardest integration testing is an approach where the integration testing is performed starting with the risky and hardest software module first.

**CONCLUSION:** In this way, we have studied various Software Testing Techniques.