**Software testing Techniques and Strategies**

Software Testing is evaluation of the software against requirements gathered from users and system specifications. Testing is conducted at the phase level in software development life cycle or at module level in program code. Software testing comprises of Validation and Verification.

**Software Validation**

Validation is process of examining whether or not the software satisfies the user requirements. It is carried out at the end of the SDLC. If the software matches requirements for which it was made, it is validated.

* Validation ensures the product under development is as per the user requirements.
* Validation answers the question – "Are we developing the product which attempts all user needs from this software?"
* Validation emphasizes on user requirements.
* “Are we building the right product?”

**Software Verification**

Verification is the process of confirming if the software is meeting the business requirements, and is developed adhering to the proper specifications and methodologies.

* Verification ensures the product being developed is according to design specifications.
* Verification answers the question– "Are we developing this product by firmly following all design specifications?"
* Verifications concentrate on the design and system specifications.
* “Are we building the product right?”

**Target of the test are** -

* **Errors** - These are actual coding mistakes made by developers. In addition, there is a difference in output of software and desired output, is considered as an error.
* **Fault** - When error exists fault occurs. A fault, also known as a bug, is a result of an error which can cause system to fail.
* **Failure**- failure is said to be the inability of the system to perform the desired task. Failure occurs when fault exists in the system. Failure occurs when fault exists in the system.

**TEST CASE**

A TEST CASE is a set of actions executed to verify a particular feature or functionality of your software application. A Test Case contains test steps, test data, precondition, postcondition developed for specific test scenario to verify any requirement. The test case includes specific variables or conditions, using which a testing engineer can compare expected and actual results to determine whether a software product is functioning as per the requirements of the customer.

For a [Test Scenario](https://www.guru99.com/test-scenario.html): Check Login Functionality there many possible test cases are:

* Test Case 1: Check results on entering valid User Id & Password
* Test Case 2: Check results on entering Invalid User ID & Password
* Test Case 3: Check response when a User ID is Empty & Login Button is pressed, and many more

**WHITE BOX TESTING**

WHITE BOX TESTING is testing of a software solution's internal structure, design, and coding. In this type of testing, the code is visible to the tester. It focuses primarily on verifying the flow of inputs and outputs through the application, improving design and usability, strengthening security. White box testing is also known as Clear Box testing, Open Box testing, Structural testing, Transparent Box testing, Code-Based testing, and Glass Box testing. It is usually performed by developers.

White box testing involves the testing of the software code for the following:

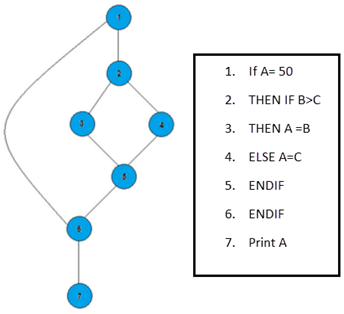
* Internal security holes
* Broken or poorly structured paths in the coding processes
* The flow of specific inputs through the code
* Expected output
* The functionality of conditional loops
* Testing of each statement, object, and function on an individual basis

**Basis path testing**

It is based on a [White Box Testing](https://www.guru99.com/white-box-testing.html) method, that defines test cases based on the flows or logical path that can be taken through the program. In software engineering, Basis path testing involves execution of all possible blocks in a program and achieves maximum path coverage with the least number of test cases. It is a hybrid of branch testing and path testing methods.

The objective behind basis path in software testing is that it defines the number of independent paths, thus the number of test cases needed can be defined explicitly (maximizes the coverage of each test case).

Here we will take a simple example, to get a better idea what is basis path testing include



In the above example, we can see there are few conditional statements that is executed depending on what condition it suffice. Here there are 3 paths or conditions that need to be tested to get the output,

* **Path 1**: 1,2,3,5,6, 7
* **Path 2**: 1,2,4,5,6, 7
* **Path 3**: 1, 6, 7

**Control structure testing**

Control structure testing is a group of white-box testing methods.

* Branch Testing
* Condition Testing
* Data Flow Testing
* Loop Testing

1) **Branch Testing**:- For every decision, each branch needs to be executed at least once also called decision testing.

Shortcoming - ignores implicit paths that result from compound conditionals.

Treats a compound conditional as a single statement. (We count each branch taken out of the decision, regardless which condition lead to the branch.)

**This example has two branches to be executed**:

IF ( a equals b) THEN statement 1

ELSE statement 2

END IF

2) **Condition Testing**:-

Condition testing is a test construction method that focuses on exercising the logical conditions in a program module.

**Errors in conditions can be due to**:

Boolean operator error

Boolean variable error

Boolean parenthesis error

Relational operator error

Arithmetic expression error

Definition: "For a compound condition C, the true and false branches of C and every simple condition in C need to be executed at least once."

Multiple-condition testing requires that all true-false combinations of simple conditions be exercised at least once.

Therefore, all statements, branches, and conditions are necessarily covered.

3) **Data Flow Testing**:-

Selects test paths according to the location of definitions and use of variables. This is a somewhat sophisticated technique and is not practical for extensive use. Its use should be targeted to modules with nested if and loop statements.

4) **Loop Testing**:-

Loops are fundamental to many algorithms and need thorough testing.

There are four different classes of loops: simple, concatenated, nested, and unstructured.

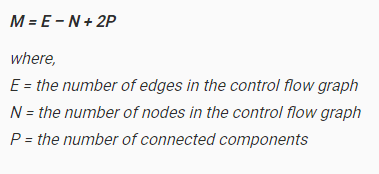
Check every type of loops with maximum allowable pass and in case of unstructured loop reconstructs it and in case of concatenated loop treat as a simple loop.

**Cyclomatic complexity**

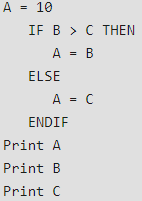
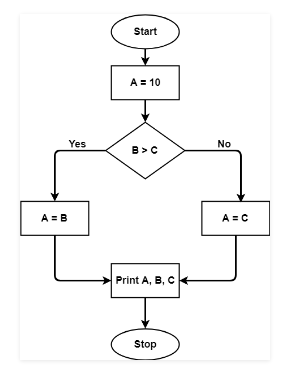
Cyclomatic complexity of a code section is the quantitative measure of the number of linearly independent paths in it. It is a software metric used to indicate the complexity of a program. It is computed using the Control Flow Graph of the program. The nodes in the graph indicate the smallest group of commands of a program, and a directed edge in it connects the two nodes i.e. if second command might immediately follow the first command.

For example, if source code contains no control flow statement then its cyclomatic complexity will be 1 and source code contains a single path in it. Similarly, if the source code contains one if condition then cyclomatic complexity will be 2 because there will be two paths one for true and the other for false.

Mathematically, for a structured program, the directed graph inside control flow is the edge joining two basic blocks of the program as control may pass from first to second. So, cyclomatic complexity M would be defined as,



Example:

**So M=E-N+2P**

**=7-7+2x1**

**=2**

**BLACK BOX TESTING**

BLACK BOX TESTING is defined as a testing technique in which functionality of the Application Under Test (AUT) is tested without looking at the internal code structure, implementation details and knowledge of internal paths of the software. This type of testing is based entirely on software requirements and specifications. In Black-Box Testing we just focus on inputs and output of the software system without bothering about internal knowledge of the software program**.**

****

The above Black-Box can be any software system you want to test. For Example, an operating system like Windows, a website like Google, a database like Oracle or even your own custom application. Under Black Box Testing, you can test these applications by just focusing on the inputs and outputs without knowing their internal code implementation.

**Generic steps followed to carry out any type of Black Box Testing.**

* Initially, the requirements and specifications of the system are examined.
* Tester chooses valid inputs (positive test scenario) to check whether SUT (software under test) processes them correctly. Also, some invalid inputs (negative test scenario) are chosen to verify that the SUT is able to detect them.
* Tester determines expected outputs for all those inputs.
* Software tester constructs test cases with the selected inputs.
* The test cases are executed.
* Software tester compares the actual outputs with the expected outputs.
* Defects if any are fixed and re-tested.

**Boundary Value Analysis (BVA)**

Boundary value analysis is based on testing at the boundaries between partitions. It includes maximum, minimum, inside or outside boundaries, typical values and error values.

It is generally seen that a large number of errors occur at the boundaries of the defined input values rather than the center. It is also known as BVA and gives a selection of test cases which exercise bounding values.

This black box testing technique complements equivalence partitioning. This software testing technique base on the principle that, if a system works well for these particular values then it will work perfectly well for all values which comes between the two boundary values.

**Guidelines for Boundary Value analysis**

* If an input condition is restricted between values x and y, then the test cases should be designed with values x and y as well as values which are above and below x and y.
* If an input condition is a large number of values, the test case should be developed which need to exercise the minimum and maximum numbers. Here, values above and below the minimum and maximum values are also tested.
* Apply guidelines 1 and 2 to output conditions. It gives an output which reflects the minimum and the maximum values expected. It also tests the below or above values.

**Example:**

Input condition is valid between 1 to 10

Boundary values 0, 1, 2 and 9, 10, 11

**Equivalence Class Partitioning**

Equivalent Class Partitioning allows you to divide set of test condition into a partition which should be considered the same. This software testing method divides the input domain of a program into classes of data from which test cases should be designed.

The concept behind this technique is that test case of a representative value of each class is equal to a test of any other value of the same class. It allows you to identify valid as well as invalid equivalence classes.

**Example:**

Input conditions are valid between

1 to 10 and 20 to 30

Hence there are five equivalence classes

--- to 0 (invalid)

1 to 10 (valid)

11 to 19 (invalid)

20 to 30 (valid)

31 to --- (invalid)

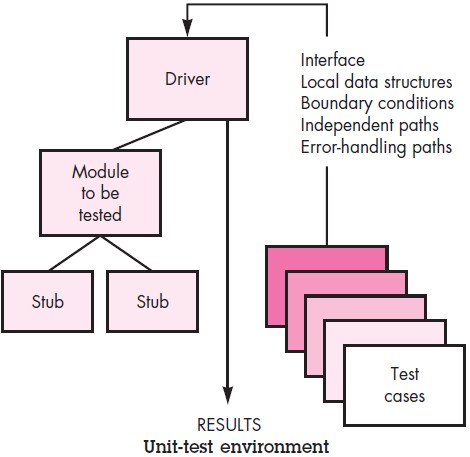
You select values from each class, i.e., -2, 3, 15, 25, 45

**UNIT TESTING**

UNIT TESTING is a type of software testing where individual units or components of a software are tested. The purpose is to validate that each unit of the software code performs as expected. Unit Testing is done during the development (coding phase) of an application by the developers. Driver and stub are software, written but not delivered with the final software product. Unit Tests isolate a section of code and verify its correctness. A unit may be an individual function, method, procedure, module, or object.

* Unit tests help to fix bugs early in the development cycle and save costs.
* It helps the developers to understand the code base and enables them to make changes quickly

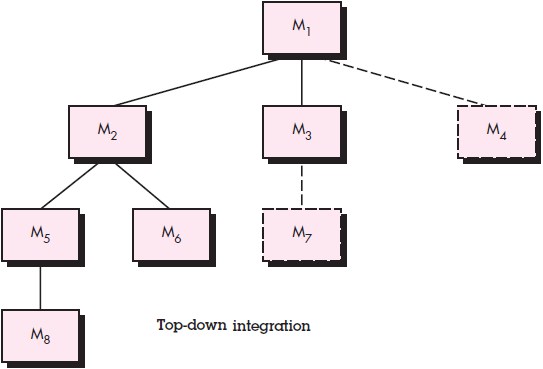
Unit testing makes heavy use of white-box testing techniques, exercising specific paths in a module's control structure to ensure complete coverage and maximum error detection in parallel for multiple components.

**The main considerations towards the unit test are**: 

* The *module interface* is tested to ensure that information properly flows into and out of the program unit under test.
* The *local data structure* is examined to ensure that data stored temporarily maintains its integrity during all steps in an algorithm's execution.
* *Boundary conditions* are tested to ensure that the module operates properly at boundaries established to limit or restrict processing.
* *Independent paths* (basis paths) through the control structure are exercised to ensure that all statements in a module have been executed at least once.
* All *error handling paths* are tested.

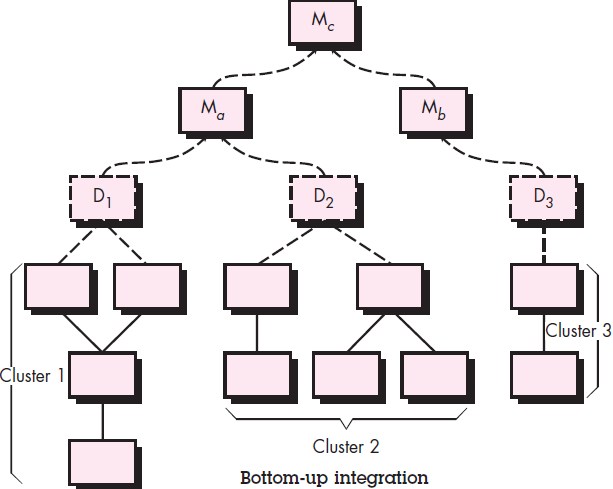
**INTEGRATION TESTING**

INTEGRATION TESTING is defined as a type of testing where software modules are integrated logically and tested as a group. A typical software project consists of multiple software modules, coded by different programmers. The purpose of this level of testing is to expose defects in the interaction between these software modules when they are integrated. Integration Testing focuses on checking data communication amongst these modules.

* 1. **Top-down integration testing**

It is an incremental approach to construction of the software architecture. Modules are integrated by moving downward through the control hierarchy, beginning with the main control module (main program). Modules subordinate (and ultimately subordinate) to the main control module are incorporated into the structure in either a depth-first or breadth-first manner and thus finally the whole programmed structure has been constructed.

* For depth first integration, selecting the left hand path, components M1, M2, M5 would be integrated first. Next, M8 or M6 would be integrated. Then, the central and right hand control paths are built.
* From breath first integration, components M2, M3, and M4 would be integrated first where M1 become the driver. The next control level, M5, M6 (M2 is driver component), and so on.



* 1. **Bottom-up integration testing**

This begins construction and testing with atomic modules (components at the lowest levels in the program structure). Because components are integrated from the bottom up, the functionality provided by components subordinate to a given level is always available and the need for stubs is eliminated. A bottom-up integration strategy may be implemented with the following steps:

* Low-level components are combined into clusters (sometimes called builds) that perform a specific software sub-function.
* A driver (a control program for testing) is written to coordinate test case input and output.
* The cluster is tested.
* Drivers are removed and clusters are combined moving upward in the program structure.
  1. **Regression testing**

Each time a new module is added as part of integration testing, the software changes. New data flow paths are established, new I/O may occur, and new control logic is invoked. These changes may cause problems with functions that previously worked flawlessly. In the context of an integration test strategy, regression testing is the re-execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects

**Validation Testing**

The process of evaluating software during the development process or at the end of the development process to determine whether it satisfies specified business requirements.

Validation Testing ensures that the product actually meets the client's needs. It can also be defined as to demonstrate that the product fulfills its intended use when deployed on appropriate environment.

It answers to the question, Are we building the right product?

When custom software is built for one customer, a series of acceptance tests are conducted to enable the customer to validate all requirements.

1. **Alpha test**

*Alpha test* is conducted at the developer's site by a representative group of end users. The software is used in a natural setting with the developer "looking over the shoulder" of the user and recording errors and usage problems. Alpha tests are conducted in a controlled environment.

1. **Beta test**

Beta test is conducted at one or more customer sites by the end-user of the software. Unlike alpha testing, the developer is generally not present. The Beta test is a "live" application of the software in an environment that cannot be controlled by the developer. The customer records all problems (real or imagined) that are encountered during beta testing and reports these to the developer at regular intervals

**SYSTEM TESTING**

System Testing is a level of testing that validates the complete and fully integrated software product. The purpose of a system test is to evaluate the end-to-end system specifications. Usually, the software is only one element of a larger computer-based system. Ultimately, the software is interfaced with other software/hardware systems. System Testing is actually a series of different tests whose sole purpose is to exercise the full computer-based system.

1. **Recovery Testing**

It verifies the system's ability to recover from points of failure like software/hardware crashes, network failures etc.

1. **Performance Testing**

It checks the speed, response time, reliability, resource usage, scalability of a software program under their expected workload.

1. **Security Testing**

It is a type of Software Testing that uncovers vulnerabilities, threats, risks in a software application and prevents malicious attacks from intruders.