**Program Testing**

Testing a program consists of providing the program with a set of test inputs (or test cases) and observing if the program behaves as expected. If the program fails to behave as expected, then the conditions under which failure occurs are noted for later debugging and correction.

Some commonly used terms associated with testing are:

- **Failure:** This is a manifestation of an error (or defect or bug). But, the mere presence of an error may not necessarily lead to a failure.
- **Test case:** This is the triplet [I,S,O], where I is the data input to the system, S is the state of the system at which the data is input, and O is the expected output of the system.
- **Test suite:** This is the set of all test cases with which a given software product is to be tested.

## ***Aim of Testing***

The aim of the testing process is to identify all defects existing in a software product. However for most practical systems, even after satisfactorily carrying out the testing phase, it is not possible to guarantee that the software is error free. This is because of the fact that the input data domain of most software products is very large. It is not practical to test the software exhaustively with respect to each value that the input data may assume. Even with this practical limitation of the testing process, the importance of testing should not be underestimated. It must be remembered that testing does expose many defects existing in a software product. Thus testing provides a practical way of reducing defects in a system and increasing the users' confidence in a developed system.

## ***Software verification and validation***

Verification and validation are the processes in which we check a product against its specifications and the expectations of the users who will be using it.

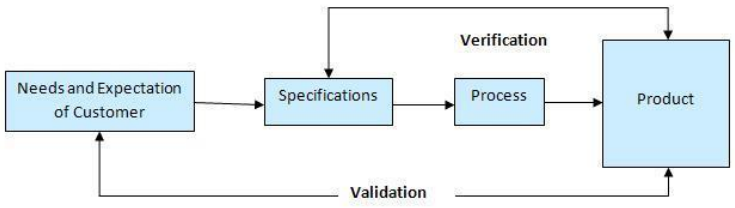
All assignment should validate and verify the software it produces. This is made by:

Verify that every software object meets particular necessities.

Verify all software objects it is used as a key to a new action.

Ensure that checking every software entities are done, as possible, by someone else other than the developer.

Ensure that the sum of validation and verification effort is enough to explain every software objects is appropriate for equipped use.




## Software Validation

Software validation checks that the software product satisfies or fits the intended use (high-level checking), i.e., the software meets the user requirements, not as specification artifacts or as needs of those who will operate the software only; but, as the needs of all the stakeholders (such as users, operators, administrators, managers, investors, etc.). There are two ways to perform software validation: internal and external. During internal software validation it is assumed that the goals of the stakeholders were correctly understood and that they were

expressed in the requirement artifacts precise and comprehensively. If the software meets the requirement specification, it has been internally validated. External validation happens when it is performed by asking the stakeholders if the software meets their needs. Different software development methodologies call for different levels of user and stakeholder involvement and feedback; so, external validation can be a discrete or a continuous event. Successful final external validation occurs when all the stakeholders accept the software product and express that it satisfies their needs. Such final external validation requires the use of an acceptance test which is a dynamic test.

Following points discuss about the validation process in software testing.

Validation determines is the system complies with the necessities and performs functions for which it is proposed and meets the organization's goals and user requirements.

It is prepared at the finish of the development method and starts after verifications are finished.

It is for building a product right.

It also checks for accessing the accurate data.

Validation is a High level action.

It is performed following an effort product is formed alongside recognized criteria ensures that the product integrates properly into the situation.

Also determines exactness of the ultimate software product by a advance project with respect to the consumer desires and necessities.

Activities involved in validation:

1. Black box testing
2. White box
3. Unit testing
4. Integration testing

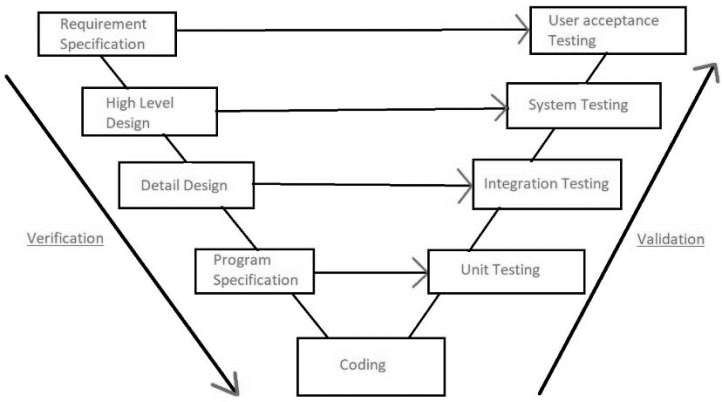
## Software Verification

Verification is a static practice of verifying documents, design, code and program. It includes all the activities associated with producing high quality software: inspection, design analysis and specification analysis. It is a relatively objective process.

Verification will help to determine whether the software is of high quality, but it will not ensure that the system is useful. Verification is concerned with whether the system is well-engineered and error-free.

- Verification makes ensure that the result is intended to give all functionality to the client.
- It is done at the initial of the development method. It includes reviews and meetings, walkthroughs, check, etc. to assess credentials, strategy, code, necessities and stipulation.
- It ensures for building a right product.
- It also checks for accessing the data correct in the correct place and in the correct way.
- Verification is a Low level action.
- It is performed at the time of development on walkthroughs, reviews and inspections, adviser comment, guidance, checklists and principles.
- Manifestation of reliability, wholeness, and accuracy of the software at every phase and among every phase of the development life cycle.

According to the Capability Maturity Model, we can also describe verification as the method of evaluating software to establish whether the yield of a particular development stage please the situation forced at the beginning of that stage.




## ***Design of Test Cases***

Exhaustive testing of almost any non-trivial system is impractical due to the fact that the domain of input data values to most practical software systems is either extremely large or infinite. Therefore, we must design an optional test suite that is of reasonable size and can uncover as many errors existing in the system as possible. Actually, if test cases are selected randomly, many of these randomly selected test cases do not contribute to the significance of the test suite, i.e. they do not detect any additional defects not already being detected by other test cases in the suite. Thus, the number of random test cases in a test suite is, in general, not an indication of the effectiveness of the testing. In other words, testing a system using a large collection of test cases that are selected at random does not guarantee that all (or even most) of the errors in the system will be uncovered. Consider the following example code segment which finds the greater of two integer values x and y. This code segment has a simple programming error.

**if (x>y)**

**max = x;**

**else**

**max = x;**

For the above code segment, the test suite, {(x=3,y=2);(x=2,y=3)} can detect the error, whereas a larger test suite {(x=3,y=2);(x=4,y=3);(x=5,y=1)} does not detect the error. So, it would be incorrect to say that a larger test suite would always detect more errors than a smaller one, unless of course the larger test suite has also been carefully designed. This implies that the test suite should be carefully designed than picked randomly. Therefore, systematic approaches should be followed to design an optimal test suite. In an optimal test suite, each test case is designed to detect different errors.