T. Y. B. Sc.
COMPUTER SCIENCE
SEMESTER-VI

NEW SYLLABUS
CBCS PATTERN

SOFTWARE TESTING

Dr. Ms. MANISHA BHARAMBE
NIKET TAJNE

NIRALI PRAKASHAN
ADVANCEMENT OF KNOWLEDGE

Syllabus ...

1. Introduction to Software Testing (5 Lectures)

- Basics of Software Testing – Faults, Errors and Failures
- Testing Objectives
- Principles of Testing
- Testing and Debugging
- Testing Metrics and Measurements
- Verification and Validation
- Testing Life Cycle

2. Software Testing Strategies and Techniques (10 Lectures)

- Testability – Characteristics Lead to Testable software
- Test Characteristics
- Test Case Design for Desktop, Mobile, Web Application using Excel
- White Box Testing – Basis Path Testing, Control Structure Testing
- Black Box Testing – Boundary Value Analysis, Equivalence Partitioning
- Differences between BBT and WBT

3. Levels of Testing (10 Lectures)

- A Strategic Approach to Software Testing
- Test Strategies for Conventional Software
- Unit Testing
- Integration Testing – Top-Down, Bottom-up Integration
- System Testing – Acceptance, Performance, Regression, Load/Stress Testing, Security Testing, Internationalization Testing
- Alpha, Beta Testing
- Usability and Accessibility Testing
- Configuration, Compatibility Testing

4. Testing Web Applications (6 Lectures)

- Dimension of Quality, Error within a WebApp Environment
- Testing Strategy for WebApp
- Test Planning
- The Testing Process – an Overview

5. Agile Testing (5 Lectures)

- Agile Testing,
- Difference between Traditional and Agile testing
- Agile Principles and Values
- Agile Testing Quadrants
- Automated Tests

■■■

CHAPTER 1

Introduction to Software Testing

Objectives...

- To understand Basic Concept of Software Testing
- To learn Principles and Life Cycle of Testing
- To understand Testing Metrics and Measurements
- To study Verification and Validation

1.0 INTRODUCTION

- The 21st century is referred to as a knowledge era which is strongly driven by Information Technology (IT).
- IT brought in revolutionary transformation in the way we collect, assimilate, process and distribute information.
- Also, specific information is availed to the right users across the world anytime and anywhere in a required form so as to drive every critical transaction or decision.
- These all could happen because of Information Technology (IT) which is a result of four Cs (Computers, Communication, Connectivity, Content and software).
- Thus, software is not only a major contributor in IT revolution but also, gifted the world an un-precedent growth.
- So in this IT world, success of any system or organization depends majorly on software which is part of the system or organization.
- We need to ensure that software shall work properly. Making software to work properly is not an easy work but is a very challenging and a highly creative work.
- In this endeavor, we need to address the problem at software development level and bugs level.
- Software testing plays an important role at both these levels and is a means of ensuring product quality and reduced rework.
- Also, testing increasing number of organizations and clients understand the value of testing and are now appreciating its contribution.

- If software has to work properly, it has to be syntactically and semantically right; meet client and user requirements; fit to the operational environment; and also fulfill requirements like efficiency, reliability, usability, portability and maintainability.
- The purpose of software testing is to analyze a particular item (hardware, software, process) of a software product to determine the differences between its requirements and actual behavior and to evaluate the features and functionality of the given item to ensure it meets the needs of the user.
- In addition, testing evaluates for fulfillment of requirements like efficiency, reliability, usability, portability and maintainability.

What is Testing?

- Testing is the process of executing software or program or system with the intent of finding/detecting errors.
- Basically, the purpose of testing is to detect and combat all factors which can lead to software or system failures before delivering the final product to the end users.
- Another purpose of testing is to assurance of quality of software or system as per desired outcomes/requirements and preventing of defects and overall software/system failure.
- Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not.
- In simple words, testing is executing a system in order to identify any errors or missing requirements in contrary to the actual requirements.
- The purpose of testing is to show that a system performs its intended functions correctly.
- Testing is the process of demonstrating that errors (bugs) are not present in the software or system.
- Testing can be defined as, "the process of evaluating a system by manual or automated means to verify that it satisfies specified requirements."
- The general testing process is the creation of a test strategy (which sometimes includes the creation of test cases), creation of a test plan/design (which usually includes test cases and test procedures) and the execution of tests.
- The purpose of the test process is to provide information to assure the quality of the product, decisions and the processes for a testing assignment.
- Fig. 1.1 shows process of testing.
- Testing is the process consisting of all life cycle activities; static and dynamic, concerned with planning, preparation and evaluation of software products and related work products to determine that they satisfy specified requirements, to demonstrate that they are fit for purpose and to detect defects.

- The main intent of software testing is to detect failures of the application so that failures can be discovered and corrected.

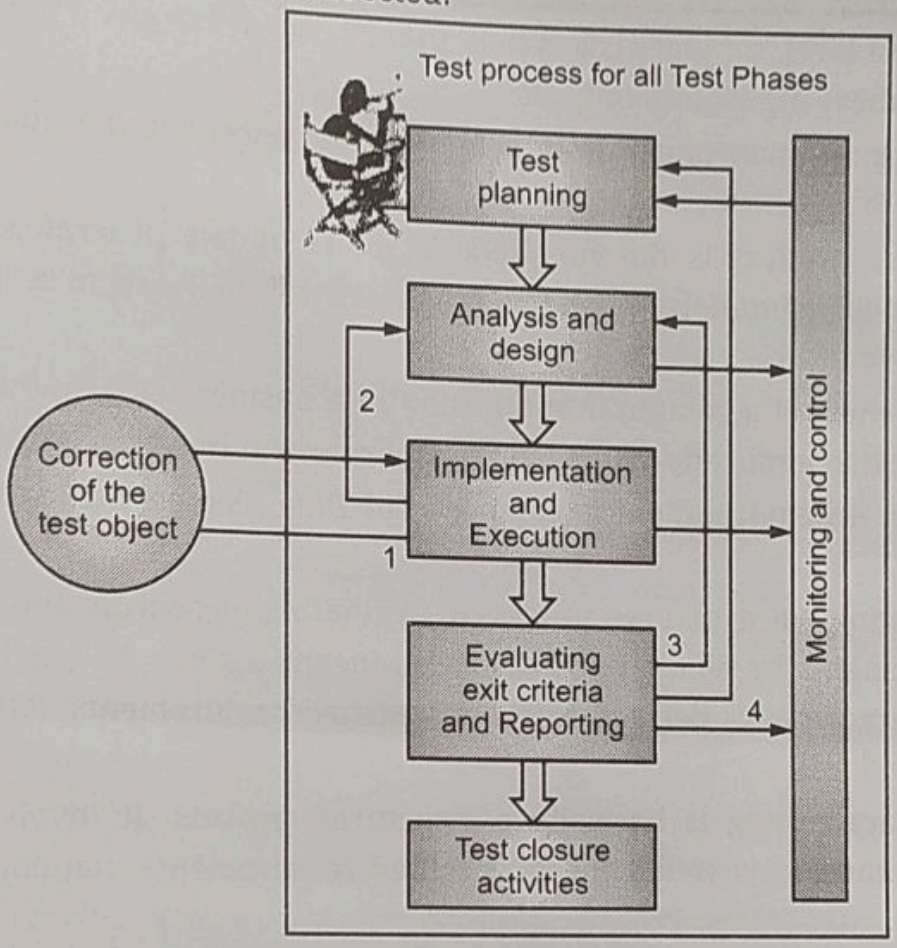

Fig. 1.1: Testing Process

- Software testing is more difficult, complex and time consuming because of the vast array of programming languages, operating systems and hardware platforms that have evolved in the intervening decades.
- Software testing is defined as, "a systematic process of executing a program or system with the intent of finding errors."

Advantages of Software Testing:

1. Software testing ensures that the software is being developed according to user requirements/expectations.
2. Software testing finds and removes errors, which prevent the software from producing output according to user requirements/expectations.
3. Software testing improves the quality of the software by removing maximum possible errors from it.
4. Software testing removes errors that lead to software failure.
5. Software testing ensures that the software conforms to business as well as user needs.

1.1 BASICS OF SOFTWARE TESTING

- At the highest level, software is a system that operates on a given inputs, processes and transforms these inputs into outputs.
- Testing plays an important role in achieving and assessing the quality of a software product.
- The software product is the complete set of computer programs, procedures and associated documentation and data designated for delivery to a customer (or end user).
- The effectiveness of a computer application in a business environment is determined by how well that application fits in to the environment in which it operates.
- Usually such an environment is composed of Data, People, Structure and Rules and Standards.
- Software testing is a structured process of evaluating measuring and rating the system or its components by manual or automated means against its quality characteristics and sub-characteristics derived from its specified requirements, latent requirements and also standards.
- Thus, software testing is basically a structured process. It involves evaluation of software to know how well it meets specified requirements, standards and fitness to cost and use.
- This makes us to perform software testing efficiently and effectively by binding people, methodology and tools and technology to work with improved capability to produce predictable results.
- Testing is mandatory because it will be a dangerous situation if the software fails any of time due to lack of testing. So, without testing software cannot be deployed/delivered to the end user.
- Software testing is intended to find errors in the software or system. Software testing is an investigation/observation conducted to provide stakeholders with information about the quality of the software product or service under test.
- Software testing can also provide an objective, independent view of the 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 detecting failures and verifying that the software product is fit for use.
- Software testing can provide objective, independent information about the quality of software and risk of its failure to users.
- A primary purpose of testing is to detect software failures so that defects may be discovered and corrected.

- Software testing involves the execution of the software or the system to evaluate one or more properties of interest. These properties indicate the extent to which the component or system under test:
  1. Is responds correctly to all kinds of inputs?
  2. Is system sufficiently usable?
  3. Is meets the requirements that guided its design and development?
  4. Is performs its functions within an acceptable time?
  5. Is the system or software achieves the general result its stakeholder's desire.
  6. Is the system or software can be installed and run in its intended environments?
- Types of testing that may be used to test a software includes:
  - Manual testing includes testing a software manually, i.e., without using any automated tool or any script.
  - In manual testing, the tester takes over the role of an end-user and tests the software to identify any unexpected behavior (or bug/error).
  - Automation testing (or test automation) is a software testing technique in which testing achieved by writing test scripts or using any automation testing tools like LoadRunner, WinRunner, TestComplete, LambdaTest, AvoAssure, Kobiton, Ranorex Studio and so on.
- Some testing roles are given below:
  1. **Test Lead or Test Manager:** The role of Test Lead/Manager is to effectively and efficiently lead the testing team. To fulfill this role the Leader must understand the discipline of testing and how to effectively implement a testing process while fulfilling the traditional leadership roles of a manager.
  2. **Test Architect:** The role of the Test Architect is to formulate an integrated test architecture that supports the testing process and create test estimates and define/build reusable testing assets for large/complex projects. To fulfill this role the Test Architect must have a clear understanding of the short-term as well as long-term goals/objectives of the organization, the resources available to the organization and a clear vision on how to most effectively deploy these assets to form integrated test architecture.
  3. **Test Designer or Tester:** The role of the Test Designer/Tester is to design test cases, execute tests, record test results, document defects and perform test coverage analysis. To fulfill this role the Test Designer/Tester must be able to apply the most appropriate testing techniques to test the application as efficiently and effectively as possible while meeting the test organizations testing mandate.
  4. **Test Automation Engineer:** The role of the Test Automation Engineer is to create automated test case scripts that perform the tests as designed by the Test Designer/Tester. To fulfill this role the Test Automation Engineer must develop and maintain an effective and efficient test automation infrastructure using the tools and techniques available to the testing organization/firm.

5. **Test Methodologist or Methodology Specialist:** The role of the Test Methodologist/Methodology Specialist is to provide the test organization/firm with resources on testing methodologies. To fulfill this role the Test Methodologist/ Methodology Specialist works with Quality Assurance (QA) to facilitate continuous quality improvement within the testing methodology and the testing organization/firm as a whole. To this end the Test Methodologist/Methodology Specialist evaluates the test strategy, provides testing frameworks and templates, and ensures effective implementation of the appropriate testing techniques.
6. **Software Tester:** Software tester is a expert who conducts prescribed tests on software programs and applications prior to their implementation to ensure quality, design integrity and proper functionality. The tester's job is to find out defects so that they will be eventually fixed by developers before the software product goes to the user/customer. A software tester must have complete knowledge about testing as a discipline. The goal of a software tester is to find bugs, find them as early as possible, and make sure they get fixed.

1.1.1 Goals of Software Testing

- The main goal of software testing is to find errors or bugs as early as possible and to fix bugs and to make sure that the software is bug free.
- The goals of software testing may be classified in to three major categories as shown in Fig. 1.2.

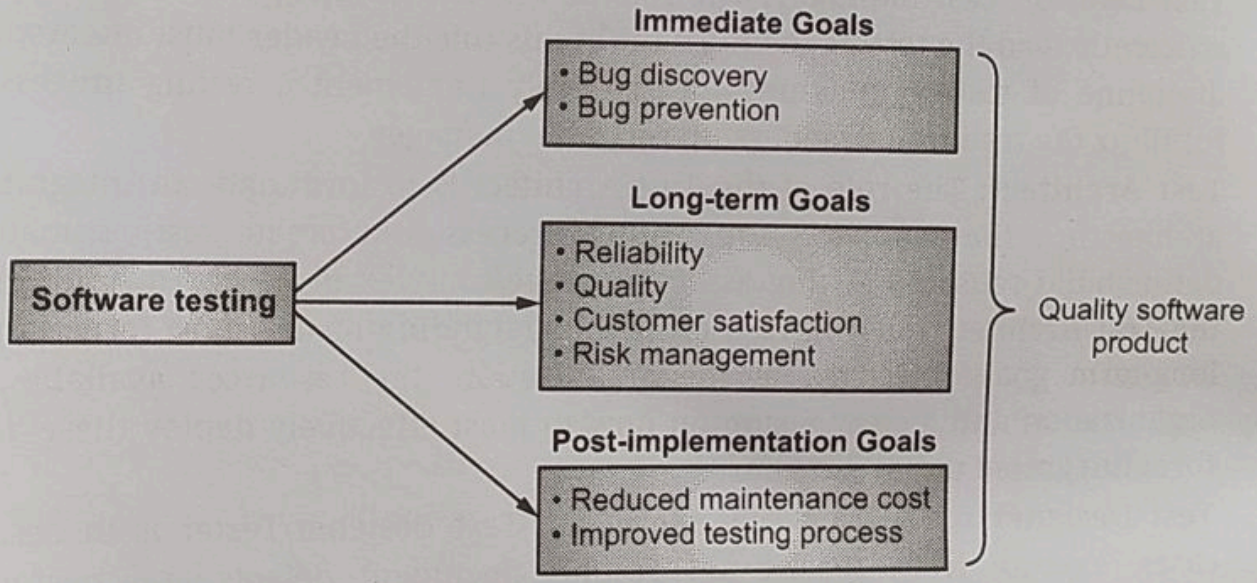

Fig. 1.2: Goals of Software Testing

- Fig. 1.2 shows following goals of software testing:
  1. **Immediate Goals:** These goals are also called as short term goals. These testing goals are the immediate result after testing. These goals contain:
     - **(i) Bug Discovery:** Is the immediate goal of software testing to find errors at any stage of software development. Numbers of the bugs are discovered in early stage of testing.
     - **(ii) Bug Prevention:** Is the consequent action of bug discovery.

2. **Long Term Goals:** These testing goals affect the software product quality in the long term. These include:
   (i) **Quality:** These goals enhance quality of the software product.
   (ii) **Customer Satisfaction:** This goal verifies the customers satisfaction for developed software product.
   (iii) **Reliability:** It is a matter of confidence that the software will not fail. In short reliability means to gain the confidence of the customers by providing them a quality product.
   (iv) **Risk Management:** Risk must be done to reduce the failure of product and to manage risk in different situations.
3. **Post Implemented Goals:** These goals are important after the software product released. Some of them are listed below:
   (i) **Reduce Maintenance Cost:** Post-released errors are costlier to fix and difficult to identify.
   (ii) **Improved Software Testing Process:** These goals improve the testing process for future use or software projects. These goals are known as post-implementation goals.

1.1.2 Model for Software Testing

- The Fig. 1.3 shows model for software testing.

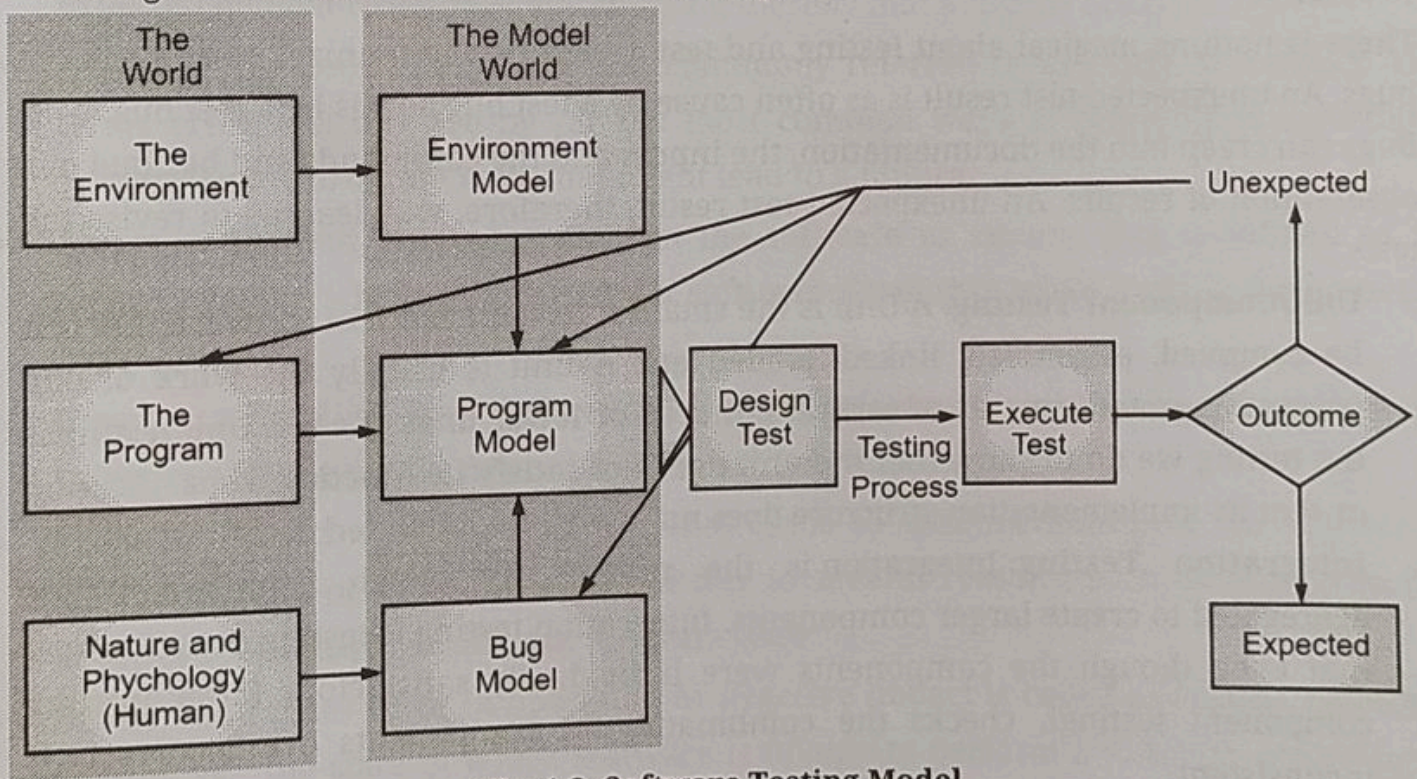

Fig. 1.3: Software Testing Model

- The software is basically a part of a system/environment for which it is being developed or produced. Systems consist of hardware and software to make the software product run.

- Program is a combination of code and data. A program's environment is the hardware and software required to make it run.
- If the program is built using a structured programming concepts with correctness, functionality, modularity and description then it is easy to test. If the code is highly complex then it is difficult to test.
- If we build the model of the program simple with appropriate level of description then testing becomes easy.
- Errors made by human beings in software development produce bugs in software work products like design or code or system. If a code or system with bug is executed, the system may experience a failure.
- Entire bug model is built around nature and psychology of programmers and testers who use bad beliefs or bad assumptions while developing the software.
- Number of the bugs appears because of these bad beliefs or bad assumptions of programmers and testers.
- Any unexpected result reported during testing may force programmers and testers to change their bad beliefs and thereby change the bug model.
- Tests are formal procedures in which inputs must be prepared, outcomes predicted, tests documented, commands executed and results observed; all these steps are subject to error.
- There is nothing magical about testing and test design that immunizes testers against bugs. An unexpected test result is as often cause by a test bug as it is by a rest bug.
- Bugs can creep into the documentation, the inputs and the commands and becloud our observation of results. An unexpected test result, therefore, way lead us to revise the tests.
    - **Unit/Component Testing:** A Unit is the smallest testable piece of software that can be compiled, assembled, linked, loaded etc. A unit is usually the work of one programmer and consists of several hundred or fewer lines of code. Unit testing is the testing we do to show that the unit does not satisfy its functional specification or that its implementation structure does not match the intended design structure.
    - **Integration Testing:** Integration is the process by which components are aggregated to create larger components. Integration testing is testing done to show that even though the components were individually satisfactory (after passing component testing), checks the combination of components are incorrect or inconsistent.
    - **System Testing:** A system is a big component. System testing is aimed at revealing bugs that cannot be attributed to components. It includes testing for performance, security, accountability, configuration sensitivity, startup and recovery.

1.1.3 Faults, Errors and Failures

- Errors are a part of our daily life. Humans make errors in their thoughts, in their actions and in the products that might result from their actions.
- Errors occur almost everywhere. To determine whether there are any errors in our thought, actions and the products generated we resort to the process of testing.
- The primary goal of testing is to determine if the thoughts, actions, and products are as desired i.e., they conform to the requirements.
- Testing of thoughts is usually designed to determine if a concept or method has been understood satisfactorily.
- Fig. 1.4 shows one class of meanings for the terms error, fault and failure.
- A programmer writes a program. An error occurs in the process of writing a program. A fault is the manifestation of one or more errors.
- A failure occurs when a faulty piece of program code is executed leading to an incorrect state that propagates to the program's output. The computer programmer might misinterpret the requirements and consequently write incorrect program code.
- Upon execution, the program might display behavior that does not match with the expected behavior/requirement implying thereby that a failure has occurred.
- A fault in the program code is also commonly referred to as a bug or a defect. The terms error and bug are by far the most common ways of referring to something "wrong" in the program code that might lead to a failure.
- Errors, faults and failures present in the software or system. Bug is defined as, "a logical mistake which is caused by a software developer while writing the software code".
- Error is defined as, "the difference/variance between the outputs produced by the software and the output desired by the user (expected output)".
- Fault is defined as, "the condition that leads to malfunctioning of the software". Malfunctioning of software is caused due to several reasons, such as change in the design, architecture or software program code.
- Defect that causes error in operation or negative impact is called as failure. Failure is defined as, "the state in which software is unable to perform a function according to user requirements/expectations".
- Bugs, errors, faults and failures prevent software from performing efficiently and hence, cause the software to produce unexpected results/outputs.

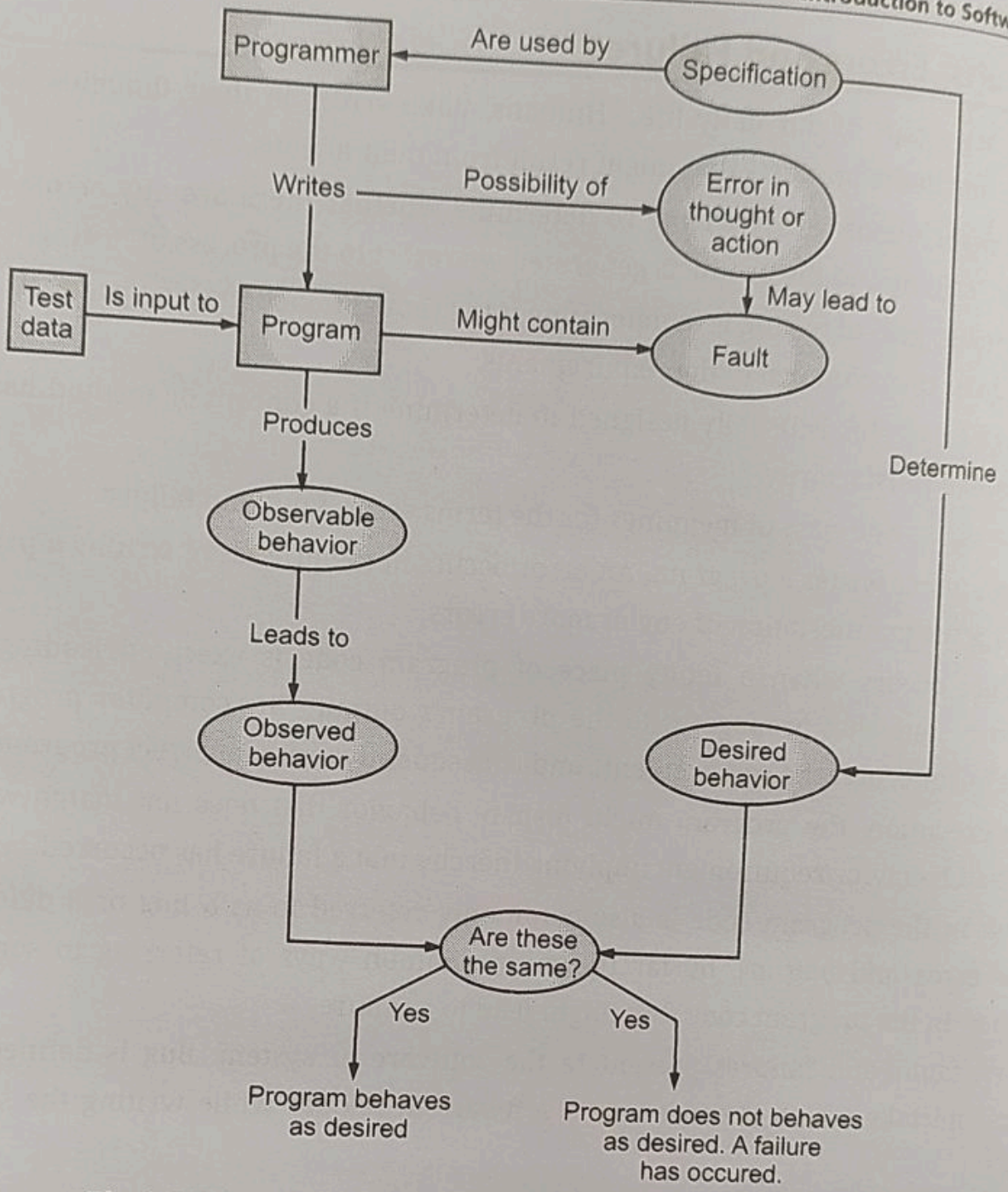

Fig. 1.4: Errors, Faults and Failures in the Process of Testing

- In short,
  - Defect or bug leads to deviation from the expected results Bug leads to faults. Faults can be viewed as incorrect results/steps occurred during the system/ program execution.
  - Faults further lead to failures. Failures are the lack of ability of a system to deliver its required function or results within specified performance requirements.

1.1.3.1 Faults

- Fault is a condition that causes the software or system to fail in performing its required function. A fault is the result of an error.

- A fault is an incorrect step or process in a computer program which causes the program to perform in an unintended or unanticipated manner.
- A fault (or a defect) is a missing or incorrect statement in a program resulting from an error is a fault. So, a fault is the representation of an error.
- A computer programmer makes an error (mistake), which results in a fault (defect, bug) in the software source code. A single fault in software or system may result in a wide range of failure symptoms.
- Faults are introduced when the program code is being developed by computer software programmers/developers.
- The computer software programmers/developers may introduce the faults during original design or when they are adding new features, making design changes or repairing faults that have been identified.
- A fault is the basic reason for software malfunction. Fault can be defined as, "the condition that leads to malfunctioning of the change in the design architecture or software code".
- A fault is also commonly called a bug. A bug is a fault in a program which causes the program to perform in an unintended behavior.
- Bug can be defined as, "a logical mistake, which is caused by a software developer/programmer while writing the software code".
- A software bug is termed as error, flaw, failure or fault in computer program or system and in terms gives incorrect and unexpected results in operations.
- In simple words, a defect is an error in coding or logic that causes a program to fail or to produce incorrect/unexpected results.

Examples of Faults:

1. Program does not work.
2. Program delivers incorrect results.
3. Programs are not written as per the requirements.
4. System does not function as per the requirements.
5. The wrong data is picked up for the execution.
6. System response is poor in the case of large volume of data or number of users accessing the system.

1.1.3.2 Errors

- Error is a discrepancy between actual value of the output given by the software and the specified correct value of the output for that given input.
- The error refers to the difference between the actual output of software and the correct output.
- We can define an error as, "the measure of deviation of the outputs given by the software from the outputs expected by the user".

- The mistake made by programmer is known as an 'Error'. Mistakes are defined as, errors of thoughts in which a person's cognitive activities lead to actions or decisions that are contrary to what was intended.
- Error also refers to human actions those results in software containing a defect or fault.

Examples of Errors:

1. Syntax errors in the code.
2. Improper logic.
3. Improper understanding of requirements
4. Incorrect system designs (even though requirements have understood!).
5. Wrong assessment of system or data load.

- Errors can be classified into following two categories:
  1. **Syntax Error:** A syntax error is a program statement that violates one or more rules of the language in which it is written.
  2. **Logic Error:** A logic error deals with incorrect data fields, out of range terms and invalid combinations.

1.1.3.3 Failures

- Failure is the inability of a system or software to perform required function according to its specification.
- In other words, a failure is the state or condition of not meeting a desirable or intended objective/goal.
- A failure occurs when a fault executes. The manifested inability of a system or software to perform a required/expected function within specified specification is known as a failure.
- A failure in a system or a software is evidenced by incorrect output, abnormal termination or unmet time and space constraints.
- Failure is defined as, "that stage of software or system under which it is unable to perform functions according to the user requirements".
- A software failure occurs when the behavior of software is different from the required behavior. A failure is produced only when there is a fault in the system.

Examples of Failures:

1. Incorrect programming/design specifications.
2. Software installation fails.
3. Software generates misleading results, thus fails to get accepted.
4. Customer refuses to accept the software and hence it fails to deliver.

Differences between Bug, Fault and Failure:

Sr. No.	Bugs	Faults	Failures
1.	All sorts of syntax errors in programming code.	Non-working of the programs.	Failure during software installation.
2.	Logic misappropriates.	Delivering not accurate.	Generates misleading results and overall failure in the software operation.
3.	Requirements improperly understood.	Programs are not as per the client requirements.	End user refuses to take delivery of the software and hence entire software failure occurs.
4.	System design incorrect.	System does not perform its functions as per the requirement.	End user implementation failure.
5.	Incorrect documentation.	Operating instructions for the software are not mentioned properly in the manual.	User not able to operate the systems correctly.
6.	Not proper data provided as input data.	Picked up wrong data for execution.	Results are incorrect due to software coding bugs.
7.	Poor system security.	System vulnerable to threats from outside world.	System not fully secured and customers not rely on.

1.2 TESTING OBJECTIVES

- Software testing not only ensures that the product functions properly under all conditions but also ensures that the product does not function properly under specific conditions.
- Software testing is the process of exercising software to verify that it satisfies requirements and to detect errors.
- The **objectives of testing** are listed below:
  1. **Finding Error:** Testing is a process of finding an error in a program or a system. A successful test is one that uncovers an undiscovered error.
  2. **Creating Good Test Cases:** A good test case is one that has a high probability of finding undiscovered error/bugs.
  3. **Quality Improvement:** Testing should systematically uncover different classes of errors in a minimum amount of time and with a minimum amount of effort.

4. **Satisfying Customer Requirements:** A secondary benefit of testing is that it demonstrates that the software appears to be working as stated in the specifications.
5. **Reliability and Quality:** The data collected through testing can also provide an indication of the software's reliability and quality. But, testing cannot show the absence of defect -- it can only show that software defects are present.

1.3 PRINCIPLES OF TESTING

- There are certain principles that are followed during software testing. These principles act as a standard to test software and make testing more effective and efficient.
- The commonly used software testing principles are explained below:

1. **Principle #1: Testing shows Presence of Defects:** Testing can show that defects are present in the system or software, but cannot provide that there are no defects. Testing reduces the probability of undiscovered defects remaining in the software or system but, even if no defects are found, it is not a proof of correctness.
2. **Principle #2: Exhaustive Testing is Impossible:** Testing everything such as all combinations of inputs and pre-conditions is not feasible except for trivial cases. Instead of exhaustive testing, we use risks and priorities to focus testing efforts.
3. **Principle #3: Early Testing:** Testing activities should start as early as possible in the software or system development life cycle and should be focused on defined objectives or goals.
4. **Principle #4: Defect Clustering:** A small number of modules contain most of the defects discovered during pre-release testing or show the most operational failures.
5. **Principle #5: Pesticide Paradox:** If the same tests are repeated again and again, eventually the same set of test cases will no longer find any new bugs. To overcome this 'pesticide paradox', the test cases need to be regularly reviewed and revised and new and different tests need to be written to exercise different parts of the software or system to potentially find more defects.
6. **Principle #6: Testing is Context Dependent:** Testing is done differently in different contexts in different environments. For example, safety-critical software is tested differently from an e-commerce Web site.
7. **Principle #7: Absence-of-Errors Fallacy:** Finding (or detecting) and fixing defects does not help if the system built is unusable and does not fulfill the users' needs and expectations.
8. **Principle #8: Tests should be as per Client Requirements:** In order to solve any sorts of defects/errors in software, proper tests cases should be developed and

hardcore testing should be performed manually or either with automation testing tools for making the software usable and bug free to meet the client's requirements.

9. **Principle #9: Define the Expected Output:** When programs are executed during testing, they may or may not produce the expected or required outputs due to different types of errors present in the software or system. To avoid this, it is necessary to define the expected output or result before software testing begins. Without knowledge of the expected/required results, testers may fail to detect an erroneous output.
10. **Principle #10: Inspect Output of each Test Completely:** Software testing should be performed once the software is complete in order to check its performance and functionality. Also, testing should be performed to find the errors that occur in various phases of software development.
11. **Principle #11: Pareto Principle to Software Testing:** Pareto principle, states that 80% of the errors are uncovered during testing procedures and 20% is likely to be traced. The problem is to isolate all sorts of errors with thorough testing of source code to make the overall source code error free.
12. **Principle #12: Testing should Begin "in small" and Progress toward Testing "in large":** The smallest programming units (or modules) should be tested first and then expanded to other parts of the system.

1.4 TESTING AND DEBUGGING

- Testing and Debugging are the most important steps during the development and after the development of any software or software application developed in any computer programming language.
- Testing involves identifying bug/error/defect/failure in the software without correcting it while Debugging involves identifying, isolating, and fixing the problems/bugs/errors.
- Testing is the process of detecting an error. It is a proactive process where we intend to find defects in the system or software.
- Debugging is the process of identifying the cause of the error once the testing/running of the software detects an error/defect.
- Software testing is a process of identifying defects in the software product and is performed to validate the behavior of the software or the application compared to requirements.
- Debugging means identifying, locating and correcting the bugs usually by running the program. The bugs in debugging are usually logical errors.
- Software testing can be defined as "a set of activities conducted with the intent of finding errors in software" or "a process of verifying that a set of programs collectively functions correctly".

- Debugging is defined as, "the process of identifying, locating and correcting the errors/bugs usually by running the program".
- Debugging is a technique to find out an error(s) and remove them in a program or set of programs; otherwise, they will lead to failure of these programs.

Process of Debugging:

- Debugging is the process of analyzing causes behind the bugs and removing them. Debugging starts with a list of reported bugs with unknown initial conditions.
- In debugging it is not possible to plan and estimate schedule and effort for debugging. Debugging is a problem solving involving deduction detailed design knowledge is of great help in good debugging.
- Debugging is done by the development team and hence is an insider's work. Also, automation of debugging is not in place.
- The goal of debugging is to find the cause of the software error and correct it. Fig. 1.5 shows the process of debugging.
- Debugging attempts to match symptom with cause, thereby, leading to error correction. Debugging will always have following one of two outcomes:
  1. The cause will be found and corrected and removed or
  2. The cause will not be found.

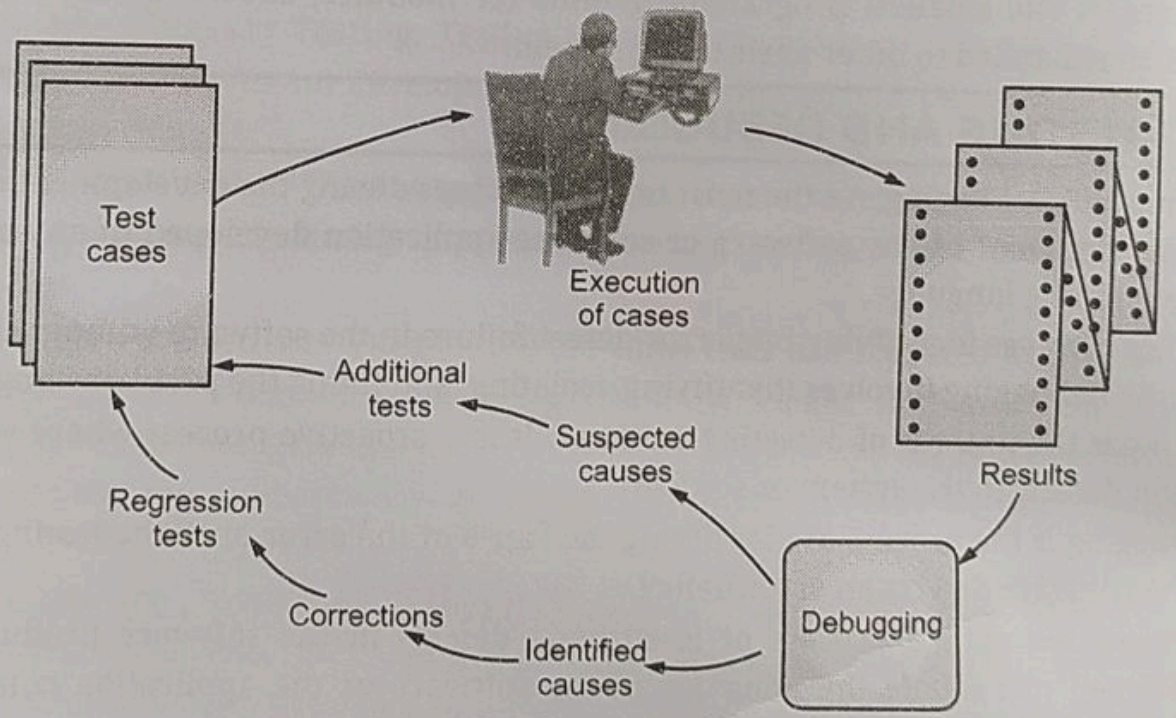

Fig. 1.5: Debugging Process

Stages involved in Debugging:

- Number of peoples thinks that program testing and debugging are the same thing. Though closely related, they are two distinct processes.
- Testing establishes the presence of errors in the program while Debugging is the locating of those errors and correcting them.

- Fig. 1.6 shows various stages in debugging. Debugging depends on the output of testing which tells the software developer about the presence or absence of errors.
- In debugging, the incorrect parts of the program code are located and the program is modified to meet its requirements.
- After repairing, the program is tested again to ensure that the errors have been corrected or removed. Debugging can be viewed as a problem-solving process.

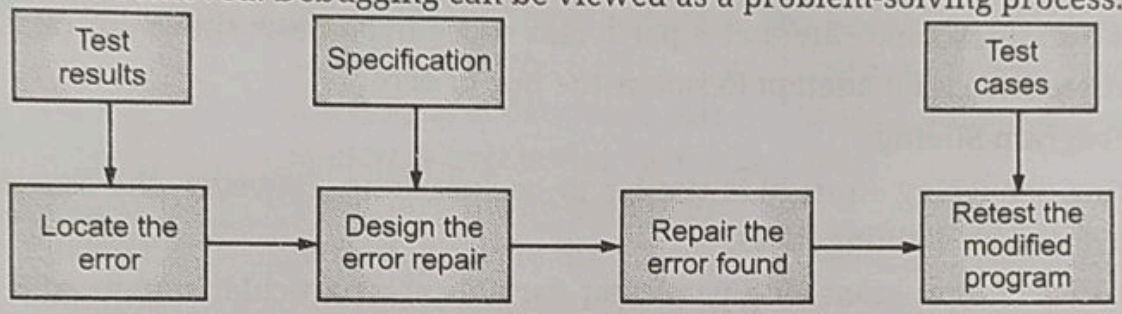

Fig. 1.6: Stages in Debugging

- There is no standard method to teach how to debug a program. The debugger must be a skilled person who can easily understand the errors by viewing the output.
- The debugger must have knowledge of common errors, which occur very often in a program.
- After errors have been discovered, then correct the error. If the error is a coding error, then that error can be corrected easily.
- But, if the error is some design mistake, then it may require effort and time. Program listings and the hard copy of the output can be an aid in debugging.

Debugging Strategies:

- Debugging is a systematic process of fixing the number of bugs or defects, in a piece of software so that the software is behaving as expected.
- Various debugging strategies are explained below:

1. **Backtracking:**
   - Backtracking is an effective debugging technique commonly used in small programs.
   - In Backtracking debugging technique, the programmer backtracks from the statement which gives the error symptoms for the first time and from this place, all the statements are checked for possible cause of errors.
   - Unfortunately, as the number of program source code lines increases, the number of potential backward paths may become unmanageably large.
   - Backtracking is done by analyzing the appearance of the error, determining the location of the symptoms and manually following the program's control flow' back to tracking the source program code until the cause of the error is found.

2. **Cause Elimination:**
   - This debugging technique is manifested by induction or deduction and introduces the concept of binary partitioning. In Cause Elimination debugging strategy, the data related to the error occurrence are organized to isolate potential causes.
   - In Cause Elimination debugging strategy, a list of all possible causes is developed and tests are conducted to eliminate each.
   - If initial tests indicate that a particular cause hypothesis shows promise, the data are refined is an attempt to isolate the bug or error.
3. **Program Slicing:**
   - This debugging strategy is similar to backtracking. However, the search space is reduced by defining slices.
   - A slice of a program for a particular variable at a particular statement is the set of source lines preceding this statement that can influence the value of that variable.
4. **Brute-force Debugging:**
   - In this debugging strategy, the programmer appends the print or write statement which, when executed, displays the value of a variable.
   - The computer software developer may trace the value printed and locate the statement containing the error. Earlier when the item for execution was quite high, the software developer had to use the core dumps (referred to as the static image of the memory and this may be scanned to identify the bug).

- For debugging we can apply wide variety of debugging tools. Debugging tool is a computer program that is used to test and debug other programs.
- Examples of automated debugging tools include Radare2, WinDbg, Valgrind and so on.

Difference between Testing and Debugging:

Sr. No.	Key Terms	Testing	Debugging
1.	Definition	Testing is a process which intent is identifying the defects.	Debugging is the activity performed by developers to fix the bug found in the system.
2.	Purpose	The purpose of testing is to show that the program or set of programs has a bug or an error. The correction phase is left to the set of developers or designers.	The purpose of debugging is to find out an error or a bug that occurs during the execution of the program and then to correct the bug.

contd. ...

3.	Objective	Main objective of testing is to find bugs and errors in an application which get missed during the unit testing by the developer.	The main objective of debugging is to find the exact root cause at code level to fix the errors and bugs found during the testing.
4.	Perform	As testing is mainly to find out the errors and bugs is mainly performed by the testers. Also if testing is at developer end known as unit testing then it is performed by the developer.	Debugging is to find the missing or de-faulty code in an application hence major performed by the developers only.
5.	Knowledge Required	As testing covers the functional and behavioral flow of an application so only functional knowledge is required for the tester to perform the testing.	Debugging is to find the error at code level so technical and code level knowledge is required for the developer to perform debugging.
6.	Automation	Testing can be manual or made automated with the help of different tools.	Debugging can't be get automated it is always be the manual.
7.	Level	Testing on basis of level of performing is at different level i.e., unit testing, integration testing, system testing, etc.	No such level of debugging.
8.	Programmer's View	Testing proves a programmer's failure.	Debugging is the programmer's vindication (justification).
9.	Process	After getting the errors, testing delivers the errors to the group of programmers to resolve them.	Debugging resolves all the errors which are provided by testing users.

1.5 TESTING METRICS AND MEASUREMENTS

- Metrics and measurement are necessary aspects of managing a software development project.
- For effective monitoring, the management needs to get information about the project such as how far it has progressed, how much development has taken place, how far behind schedule it is, what is the quality of the development? Based on this information, decisions can be made about the project.

- Without proper metrics to quantify the required information, subjective opinion would have to be used, but this is unreliable and goes against the fundamental goals of engineering.
- Software metrics are quantifiable measures that could be measure different characteristics of software systems (and/or the software development process).
- Metric is extremely important to measure the quality, cost and effectiveness of the project and the processes. Without measuring these, a project cannot head towards successful completion.
- Software metrics are a measure of some property of a piece of software systems or its specifications.
- Software testing metric is defined as, "a quantitative measure that helps to estimate the progress and quality of a software testing process".
- A software testing metric is a quantitative measure of the degree to which a system, component or process possesses a given attribute.
- Software testing metrics can be classified into following categories:

1. **Product Metrics:** A product metric is a metric used to measure the characteristics of the product such as size, complexity, design features, performance, and quality level. Product metrics in software are used to quantify characteristics of the product being developed, i.e. the software. Product metrics can be further classified as:
   - (i) **Project Metrics:** A set of metrics that indicates how the project is planned and executed. Project Metrics describe the project characteristics and execution. Examples include the number of software developers, the staffing pattern over the life cycle of the software, cost, schedule, and productivity.
   - (ii) **Progress Metrics:** A set of metrics that tracks how the different activities of the project are progressing. The activities include both development activities and testing activities. Progress metrics is monitored during testing phase. Progress metrics, is classified into test defect metrics and development defect metrics.
   - (iii) **Productivity Metrics:** A set of metrics that takes into account various productivity numbers that can be collected and used for planning and tracking testing activities. These metrics help in planning and estimating of testing activities.
2. **Process Metrics**: A process metric is a metric used to measure characteristics of the methods, techniques and tools employed in developing, implementing and maintaining the software product or system. A process metric can be used to improve the development and maintenance activities of the software. Process metrics in software are used to quantify characteristics of the environment or the

process being employed to develop the software. Process metrics aim to measure such considerations as productivity, cost and resource requirements, and the effect of development techniques and tools.

- Measurement plays a significant role in software testing. To access the quality of the engineered software product or system and to better understand the models that are created, some measures are used.
- These measures are collected throughout the software development life cycle with an intension to improve the software process on a continuous basis.
- Measurement help in estimation, quality control, productivity assessment and project control throughout a software project.
- Measurement is the action of measuring something. Measurement is the assignment of a number to a characteristic of an object or event, which can be compared with other objects or events.
- Actual measurement must be performed on a given software system in order to use metrics for quantifying characteristics of the given software.
- The LoC (Lines of Code) is the most widely used metrics for measurement of size of a program.
- Measure can be defined as, "quantitative indication of amount, dimension, capacity, or size of product and process attributes". Measurement can be defined as, "the process or determining the measure".
- Metrics can be defined as, "quantitative measures that allow software engineers to identify the efficiency and improve the quality of software process, project and product".
- A measurement is an indication of the size, quantity, amount or dimension of a particular attribute of a product or process. For example the number of errors in a system is a measurement.
- A metric is a measurement of the degree that any attribute belongs to a system, product or process. For example the number of errors per person hours would be a metric.

1.5.1 Metric

- The term "metric" refers to a standard of measurement. A metric is a quantitative measure of that system or software. Software metrics are used to measure the quality of software.
- To achieve an accurate schedule and cost estimate, better quality products, and higher productivity, an effective software management is required, which in turn can be attained through the use of software metrics.
- The IEEE defines metric as, "a quantitative measure of the degree to which a system, component or process possesses a given attribute".

- The goal of software metrics is to identify and control essential parameters that affect software development.
- The **objectives of using software metrics** are given below:
  1. Measuring the size of the software quantitatively.
  2. Assessing the level of complexity involved.
  3. Estimating cost of resources and project schedule.
  4. Assessing the strength of the module by measuring coupling.
  5. Assessing the testing techniques.
  6. Specifying when to stop testing.
  7. Determining the date of release of the software.
- In testing, metric is a unit used for describing or measuring an attribute. Test metrics are the means by which the software product quality can be measured.
- Test metrics provides the visibility into the readiness of the software product and gives clear measurement of the quality and completeness of the software product.
- Metric is a standard unit of measurement that quantifies results and used for evaluating the software processes, products and services is termed as software metrics.
- Fig. 1.7 (a) shows metrics life cycle. The steps/stages in life cycle metrics of metrics are given below:
  1. **Analysis:** In this stage, developers identify the required metrics and define them.

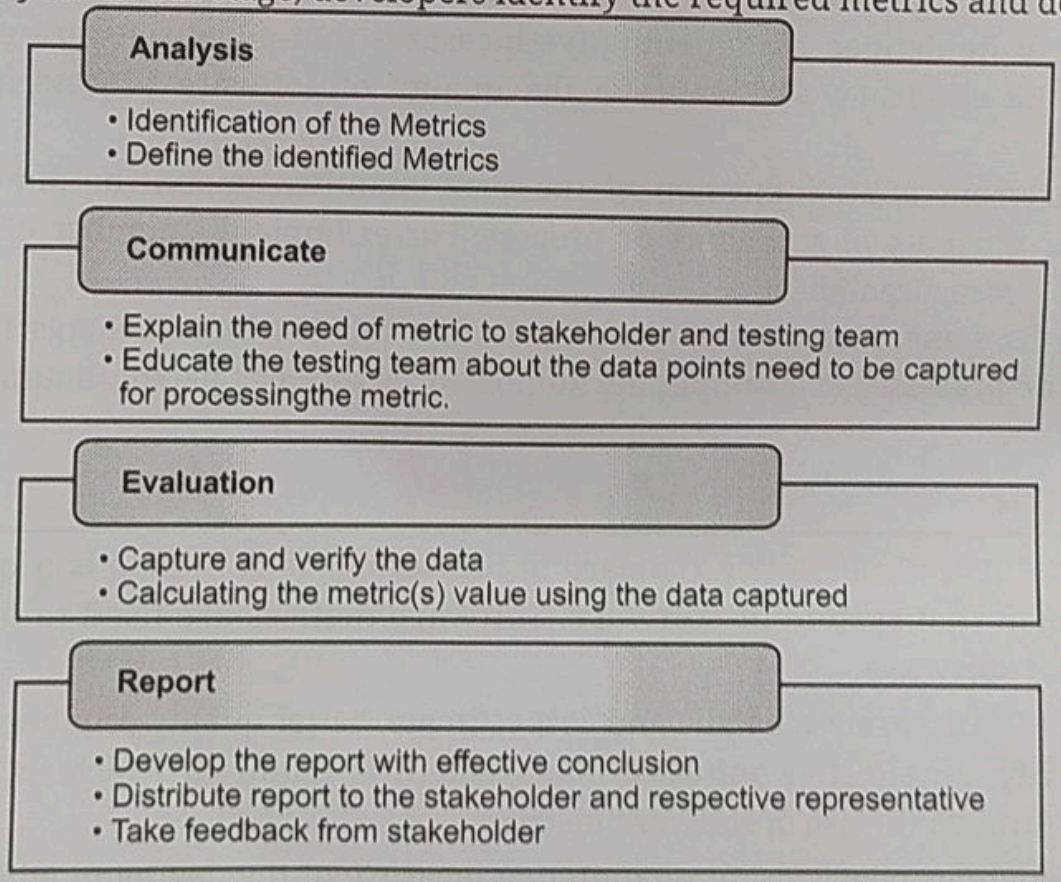

Fig. 1.7 (a)

2. **Communicate:** Once metrics are identified, developers have to explain their importance to stakeholders and the testing team.
3. **Evaluation:** This stage includes quantifying and verifying the data. Then testers have to use the data to calculate the value of the metric.
4. **Report:** Once the evaluation process is finished, the development team needs to create a report including a detailed summary of the conclusion. Then the report is distributed among stakeholders and relevant representatives. The stakeholders then give their feedback after reading the information carefully.

- A software metrics is a measurement based technique which is applied to processes, products and services to supply engineering and management information.
- A software metrics working on the information supplied to improve processes, products and services, if required.
- A test metric measures some aspect of the test process. Test metrics could be at various levels such as at the level of an organization, a project, a process or a product.

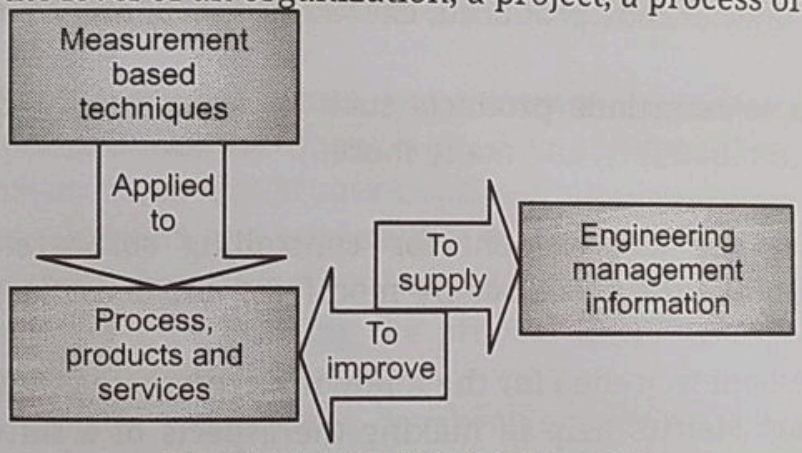

Fig. 1.7 (b): Test Metric Process

- Various characteristics of software Metrics:
  1. **Economical:** Computation of metrics should be economical.
  2. **Quantitative:** Metrics must possess quantitative nature. It means metrics can be expressed in values.
  3. **Language Independent:** Metrics should not depend on any programming language.
  4. **Understandable:** Metric computation should be easily understood ,the method of computing metric should be clearly defined.
  5. **Applicability:** Metrics should be applicable in the initial phases of development of the software.
  6. **Repeatable:** The metric values should be same when measured repeatedly and consistent in nature.

Example of Test Metrics:

1. How many defects are existed within the module?
2. How many test cases are executed per person?
3. What is the Test coverage %?

1.5.2 Measurement

- Measurement is the quantitative indication of extent, amount, dimension, capacity or size of some attribute of a software product or software process.
- The software measurement helps in estimation, quality control, productivity assessment and project control throughout a software project.
- In software measurement, software is measured to find its efficiency and accuracy. The functionality of software product is totally dependent on the measurement process.
- If the software gives correct output or result in the software measurement process, then it can be sure that software will give meaningful and valuable information within a defined time.
- Software measurements are of two categories namely, direct measures arid indirect measures.
  1. **Direct measures** include software processes such as cost and effort applied and products like lines of code produced, execution speed, and other defects that have been reported.
  2. **Indirect measures** include products such as functionality, quality, complexity, reliability, maintainability, and many more.

Need of Software Measurement:

- Measurements are the key element for controlling software project processes. Measurements improve the processes by modifying the activities based on different measures.
- Software measurement is needed for the following activities as shown in Fig. 1.8.
  1. **Understanding:** Metrics help in making the aspects of a software process more visible, thereby giving a better understanding of the relationships among the activities and entities they affect.
  2. **Control:** Using baselines, goals/objectives and an understanding of the relationships, we can predict what is likely to happen and correspondingly, make appropriate changes in the process to help meet the goals.
  3. **Improvement:** By taking corrective actions and making appropriate changes, we can improve a software product. Similarly, based on the analysis of a project, a process can also be improved.

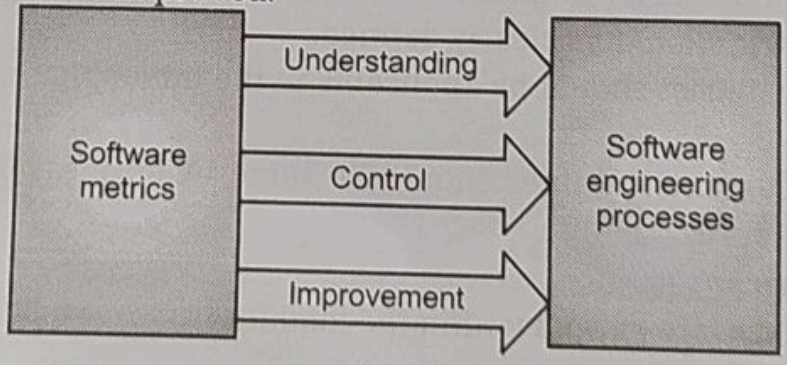

Fig. 1.8

- Metrics are derived from measurements using appropriate formulae or calculations.

1.6 VERIFICATION AND VALIDATION

- The idea of catching defects within each phase, without letting them reach the testing phase, leads to define Verification and Validation.
- Verification and Validation (V&V) activities are two branches of software testing. They are complementary to each other and not substitutes of each other. Verification and validation are completely dependent on each other.
- According to Stauffer and Fuji (1986), software V&V is a systems engineering process employing a rigorous methodology for evaluating the correctness and quality of software product through the software life cycle."
  - Verification refers to the process of ensuring that the software is being developed according to its specifications. For verification, techniques such as a reviews, analyses, inspections and walkthroughs are commonly used.
  - Validation refers to the process of checking that the developed software meets the requirements specified by the user.
- Software verification and validation activities check the software against its specification.
- During requirement gathering phase requirements are collected. The SRS (Software Requirements Specification) document is product of requirement phase. It is verified from the customer to check that proper requirements are captured.
- The design phase takes SRS and implements the design which is useful during coding. The requirement team checks whether the SRS is mapped to design phase properly.
- Verification means "Are we building the product right?" and Validation means "Are we building the right product?"
- The V&V is the generic name given to checking processes which ensure that the software conforms to its specification and meets the needs of the customer.
  - **Verification:** It involves checking that the program conforms to its specification.
  - **Validation:** It involves checking that the program as implemented meets the expectations of the customer.
- The system should be verified and validated at each stage of the software development process using documents produced in earlier stages.
- Verification and validation thus starts with requirements reviews and continues through design and code reviews to product testing.
- Fig. 1.9 shows software verification and validation.

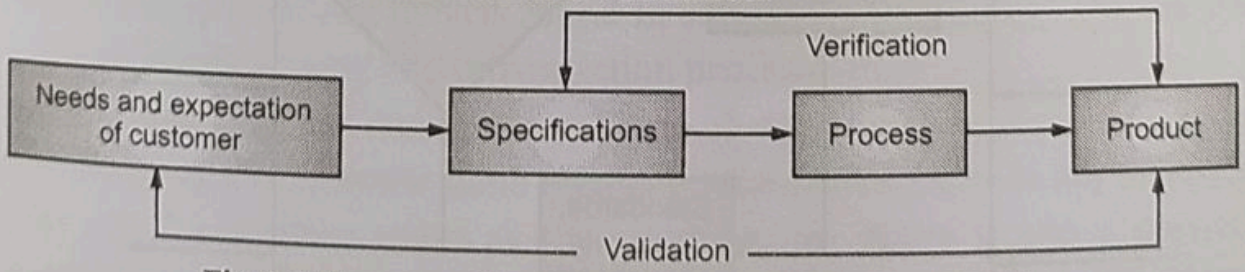

Fig. 1.9: Process of Verification and Validation (V&V)

- Requirements review, design review, code review etc. are examples of verification. Unit testing, Integration testing, system testing etc. are examples of validation.

1.6.1 Verification

- Verification is a disciplined approach to evaluate whether a software product fulfills the requirements or conditions imposed on them by the standards or processes.
- Verification is done to ensure that the processes and procedures defined by the customer and/or organization for development and testing are followed correctly.
- Verification is also called 'static technique' as it does not involve execution of any code, program or work product.
- Verification refers to, "confirmation by examination and provisions of objective evidence that specified requirements/expectations have been fulfilled".
- A test bench or testing workbench is a virtual environment used to verify the correctness of software product.
- A test bench has following components:
  1. **Input:** The entrance criteria or deliverables needed to perform work.
  2. **Procedures to Do:** The task or processes that will transform the input into the output.
  3. **Procedures to Check:** The processes that determine that the output meets the standards.
  4. **Output:** The exit criteria or deliverables produced from the workbench.

Workbench of Verification:

- Verification is the process of checking or verifying the credentials, data or information to confirm their credibility and accuracy.
- A verification workbench is where verification activities are conducted and may be a physical or virtual entity.
- A workbench is a way of documenting how a specific activity has to be performed. Fig. 1.10 shows verification workbench.

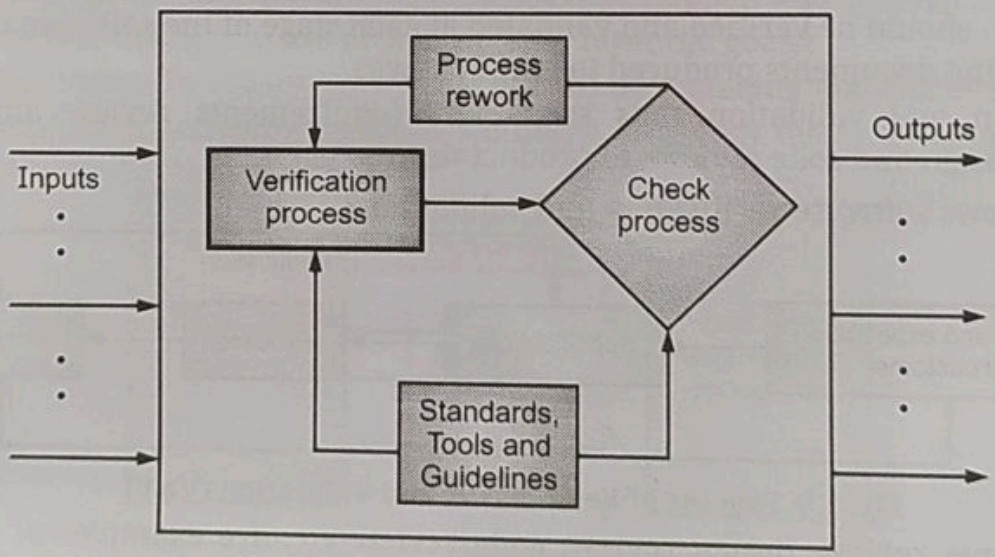

Fig. 1.10: Workbench of Verification

- For every workbench the following basic things are required:
  1. **Inputs:** Must be some entry criteria definition when inputs are entering the work bench. This definition should match with the output criteria for the earlier work bench. For example, if we are verifying a code file, then we must have a written and compatible code as an input to the work bench along with verification plan, verification artifacts, etc.
  2. **Output:** Must be some exit criteria from work bench which should match with input criteria for the next work bench. Outputs may include review comments and the work product with defects if any.
  3. **Verification Process:** Must describe step-by-step activities to be conducted in a work bench. It must also describe the activities done while verifying the work product under review.
  4. **Check Process:** Must describe how the verification process has been checked. It is not a verification of the work product but a verification of the process used for verification. Quality plan must define the objectives/goals to be achieved and check processes must verify that the objectives have been really achieved.
  5. **Standards, Tools and Guidelines:** May be termed 'the tools' available for conducting verification activities. There may be coding guidelines or testing standards available for verifications. Sometimes, checklists are used for doing verification.

Methods of Verification:

- There are many methods available of software work products. Some of them are given below:
  1. **Walkthrough:** It is a semi-formal type of a review as it involves larger teams along with the author reviewing a work product. It may involve a project team or part of a project team doing a review jointly as the case may be.
  2. **Self Review:** They may not be referred as an official way of review in most of the software verification descriptions, as it is assumed that everybody does a self check before giving work product for further verification. One must capture the self review records and defects found in self review to improve the process. It is basically a self-learning and retrospection process.
  3. **Peer Review:** It is the most informal type of review where an author and a peer are involved. It is a review done by a peer and review records are maintained. A peer may be a fellow tester as the case may be. There is also a possibility of superior review where peer is a supervisor with better knowledge and experience.
  4. **Audits:** It is a formal review based on samples. They are conducted by auditors who may or may not be experts in the given work product.

5. **Inspection (Formal Review):** Where people external to the team may be involved as inspectors. They are 'subject matter experts' who review the work product. It is also termed 'Fagan's inspection'.

Advantages of Verification:

1. Verification reduces the chances of failures in the software application or product.
2. Verification helps in lowering down the count of the defect in the later stages of software development.
3. Verifying or checking the software product at the starting phase of the development will help in understanding the software product in a better way.
4. It helps in building the software product as per the customer specifications, requirements and needs.
5. Verification can confirm that the work product has followed the processes correctly as defined by an organization or customer.
6. Verification can reduce the cost of finding and fixing defects as each work product is reviewed and corrected faster. Sometimes, defects are fixed as and when they are found.
7. Verification can locate the defect easily as the work product under review is yet to the integrated with other work products. Cost of fixing defect is less as there is no/minimum impact on other parts of software.

Disadvantages of Verification:

1. Actual working software may not be accessed by verification as it does not cover any kind of execution of a work product. 'Fit for use' cannot be assessed in verification.
2. Verification cannot show whether the developed software is correct or not. Rather, it shows whether the processes have been followed or not. If processes are not capable, then the outcome may not be good.

1.6.2 Validation

- Validation is a disciplined approach to evaluate whether the final built software product fulfills its specific intended use.
- It is meant to validate the requirements as defined in requirement specification, ensure that the application as developed matches with the requirements defined and is fit for the intended or future use.
- Validation is also called 'dynamic testing' as the application is executed during validation, with the intention to find defects.
- Software Validation is a process of evaluating software product, so as to ensure that the software meets the pre-defined and specified business requirements as well as the end users/customers' demands and expectations.

- Validation can be defined as, "the process of evaluating as system or component during or at the end of the development process to determine whether it satisfies specified requirements".

Validation Workbench:

- A validation work bench is a place where validation activities are conducted on the work products and may be a physical or virtual entity.
- Fig. 1.11 shows a validation workbench.

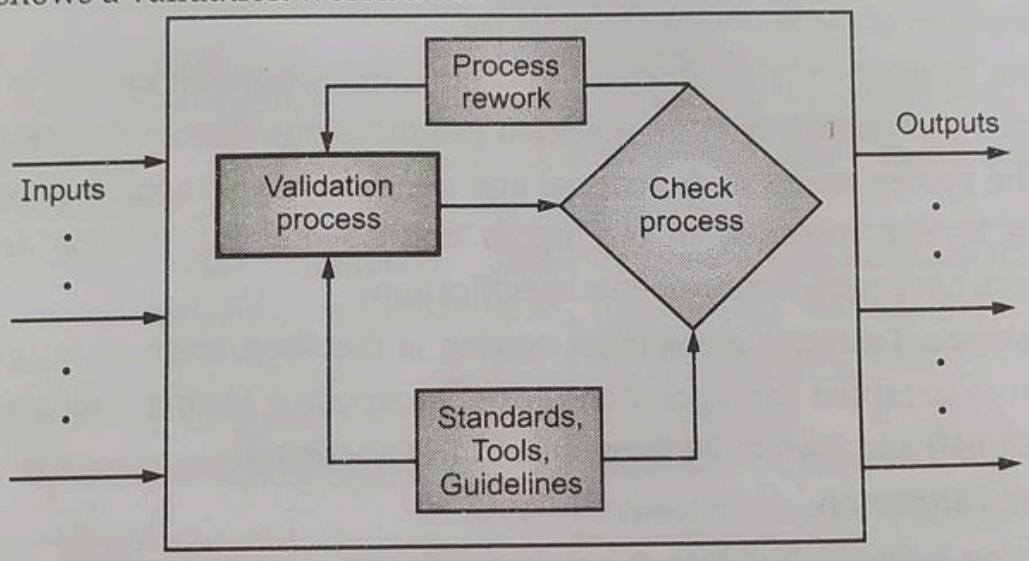

Fig. 1.11: Workbench of Validation

- Fig. 1.11 shows following basic things in workbench in validation:
  1. **Inputs:** It must be some entry criteria definition when inputs are entering the work bench. This definition should match with the output criteria of the earlier work bench. For example, if we are validating an application or part of it, then we must have a written and compliable code as an input to the work bench along with test plan and test cases/test data as per definition of input.
  2. **Outputs:** They are exit criteria from work bench which should match with input criteria for the next work bench. Outputs may include validated work products, defects and test logs.
  3. **Validation Process:** It must describe step-by-step activities to be conducted in a work bench. It must also describe the activities done while validating the work product under testing.
  4. **Check Process:** It must describe how the validation process has been checked. It is not a validation of the work product but a process. Test plan must define the objectives/goals to be achieved during validation and check processes must verify that the objectives have been really achieved.
  5. **Standards, Tools and Guidelines:** May be termed "the tools" available for validation. There may be testing guidelines or testing standards available. Sometimes, checklists are used for doing validation.

Levels of Validation:

1. **Unit Testing:** In this testing the smallest piece of software that can be tested in isolation. It is procedure used to validate that individual unit of source code is working properly.
2. **Integration Testing:** It starts at module level when various modules are integrated with each other to form a sub-system or system. Focuses on design and construction of the software architecture.
3. **System Testing:** It is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. The system testing validates that the system meets its functional and non-functional requirements. The system testing is also intended to test up to and beyond the bounds defined in the software/hardware requirements specifications.
4. **Acceptance Testing:** Acceptance testing is the final stage of testing before the system is accepted for operational use. Acceptance testing validates User needs (Functional) and System Performance (Non-Functional).

Advantages of Validation:

1. Validation helps in building the right product as per the customer's requirement and helps in satisfying their needs.
2. During verification if some defects cannot be find or missed then during validation process it can be caught as failures.
3. If during verification some specification of the software product is misunderstood and development had happened then during validation process while executing that functionality the difference between the actual result and expected result can be understood.
4. Validation is done during testing such as feature testing, integration testing, system testing, load testing, compatibility testing, stress testing, etc.
5. Validation is generally independent of the platform of development, database or any technical aspect related to software development.

Disadvantages of Validation:

1. Sometimes, it may result into redundant testing as the tester is not aware of internal structure and same part of the code gets executed again and again.
2. No amount of testing can prove that software does not have defects. If defects are not found by conducting the defined test cases, we can only conclude that there is no defect in the set of transactions actually executed. There can be many more defects not captured by these test cases.

Difference between Verification and Validation:

Sr. No.	Verification	Validation
1.	Verification means "are we building the product right?"	Validation means "are we building the right product?"
2.	Verification is a static practice of verifying documents, design, code and program.	Validation is a dynamic mechanism of validating and testing the actual product.
3.	It does not involve executing the code.	It always involves executing the code.
4.	It is human based checking of documents and files.	It is computer based execution of program.
5.	Verification uses methods like inspections, reviews, walkthroughs, and desk-checking etc.	Validation uses methods like black box (functional) testing, gray box testing, and white box (structural) testing etc.
6.	Verification is to check whether the software conforms to specifications.	Validation is to check whether software meets the customer expectations and requirements.
7.	It can catch errors that validation cannot catch. It is low level exercise.	It can catch errors that verification cannot catch. It is high level exercise.
8.	Verification is done by testing team to ensure that the software is as per the specifications in the SRS document.	Validation is carried out with the involvement of testing team.
9.	It generally comes first-done before validation.	It generally follows after verification.

1.7 SOFTWARE TESTING LIFE CYCLE

- Software testing is regarded as careful investigation done by software testing professionals on the source code/every module of the source code to detect errors and to enhance the quality of the software product.
- Software Testing Life Cycle (STLC) is the testing process which is executed in systematic and planned manner. In STLC process, different activities are carried out to improve the quality of the software product.
- STLC is a group of circularly arranged testing activities, in a specific sequence to understand and test the software in a structured way.
- The fundamental model called the 'Waterfall model' is for STLC is discussed in this section. The waterfall model is strictly sequential means it is a series of phases with each phase having a distinct beginning and ending.

- Each phase in Waterfall model enhances development, resulting in production of artifacts and at the end of the life cycle in the desired product/project.
- Fig. 1.12 shows Waterfall testing cycle. The waterfall model is a linear and sequential model defined for software engineering life cycle.

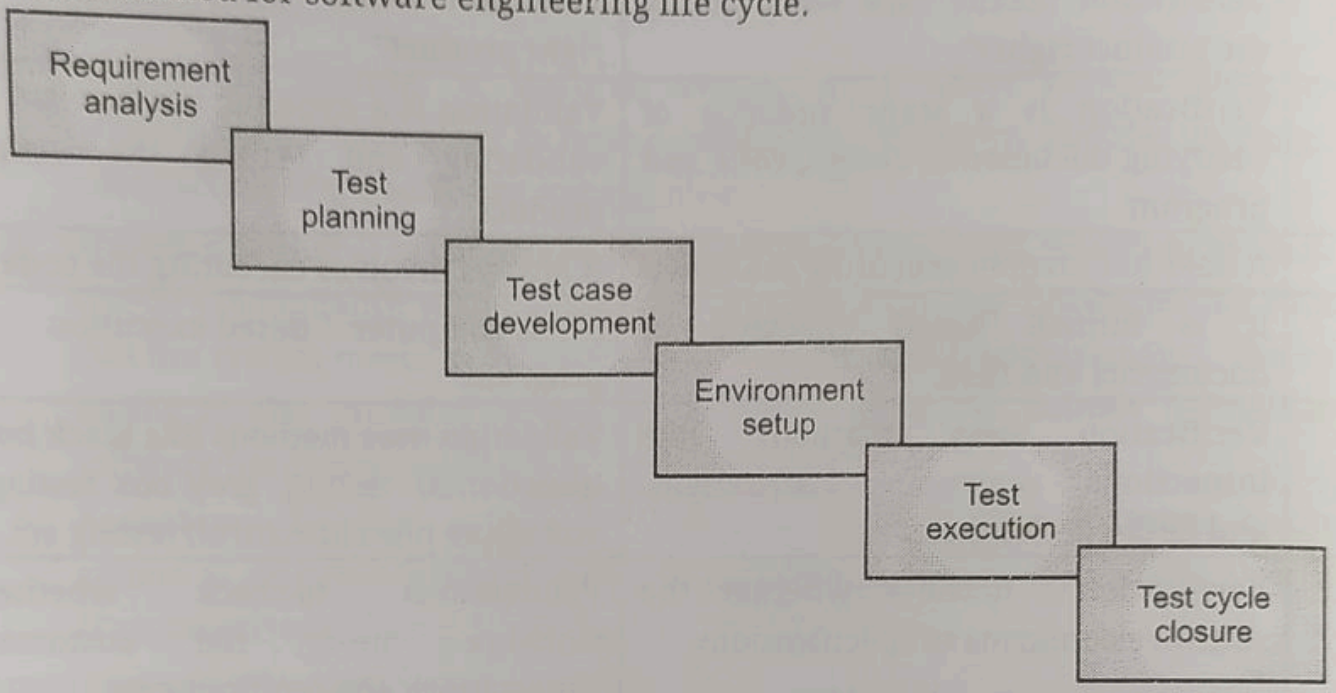

Fig. 1.12: Waterfall Model for Software Testing Life Cycle (STLC)

- Fig. 1.12 shows the phases involved in the STLC. Each of these phases is linked to a specific entry and exit criterion as well as to activities and services.
- The entry criteria contain the required elements that must be executed before starting the test process. The exit criteria define the items that must be completed before the test can be completed.
- Various phases in the Fig. 1.12 are as explained below:

1. **Requirements Analysis:** Testing should begin in the requirements phase to understand requirements in detail. The requirements can be both functional (what the software program should do) and non-functional (which defines the overall performance, security, availability, etc. of the system). At this stage, the feasibility study is used. During the design phase, testers work with developers in determining what aspects of a design are testable and with what parameters those tests work.

   Activities in Requirement Phase of STLC:

   - Identify test environment details where testing is supposed to be carried out.
   - Identify types of tests to be performed.
   - Gather details about testing priorities.
   - Prepare Requirement Traceability Matrix (RTM). The RTM is a document that maps and traces user requirement with test cases. It captures all requirements proposed by the client and requirement traceability in a single document,

delivered at the conclusion of the software development life cycle. The main purpose of RTM is to validate that all requirements are checked via test cases such that no functionality is unchecked during software testing.

2. **Test Planning:** In this phase the test strategy or the test plan is defined.

   Activities in Test Planning Phase of STLC:

   - Preparation of test plan/strategy document for various types of testing.
   - Test tool selection.
   - Test effort estimation.
   - Resource planning and determining roles and responsibilities.
   - Training requirement.

3. **Test Case Development:** Test procedures, test scenarios, test cases, test datasets, test scripts to use in testing software. The test case development phase involves the creation, verification and rework of test cases and test scripts after the test plan is ready. Initially, the test data is identified then created and reviewed and then reworked based on the preconditions.

   Activities in Test Case Development Phase of STLC:

   - Create test cases, automation scripts (if applicable).
   - Review and baseline test cases and scripts.
   - Create test data (if Test Environment is available).

4. **Test Environment Setup:** This STLC phase decides the software and hardware conditions under which a work product is tested.

   Activities in Test Environment Setup Phase of STLC:

   - Understand the required architecture, environment set-up and prepare hardware and software requirement list for the test environment.
   - Setup test environment and test data.
   - Perform smoke test on the build.

5. **Test Execution:** Test execution phase is carried out by the testers in which testing of the software build is done based on test plans and test cases prepared. The process consists of test script execution, test script maintenance and bug reporting. If bugs are reported then it is reverted back to development team for correction and retesting will be performed. Testers execute the software based on the plans and test documents then report any errors found to the development team. The test cases are run according to the given steps and expected results and every test case is marked as pass or fail and test results are created.

Activities in Test Execution Phase in STLC:

- Execute tests as per plan.
- Document test results, and log defects for failed cases.
- Map defects to test cases in RTM.
- Retest the Defect fixes.
- Track the defects to closure.

6. **Test Closure Activity:** Once, the test meets the exit criteria, the activities such as capturing the key outputs, lessons learned, results, logs, documents related to the project are archived and used as a reference for future projects. Test cycle closure phase is completion of test execution which involves several activities like test completion reporting, collection of test completion matrices and test results. Testing team members meet, discuss and analyze testing artifacts to identify strategies that have to be implemented in future, taking lessons from current test cycle. The idea is to remove process bottlenecks for future test cycles.

Activities in Test Cycle Closure Phase in STLC:

- Evaluate cycle completion criteria based on time, test coverage, cost, software, critical business objectives, and quality.
- Prepare test metrics based on the above parameters.
- Document the learning out of the project.
- Prepare Test closure report.
- Qualitative and quantitative reporting of quality of the work product to the customer.
- Test result analysis to find out the defect distribution by type and severity.

Concept of V Model:

- The first variant of the waterfall model is the 'V model' of development (so called because it is in the shape of the alphabet V). The V-model is also known as Verification and Validation (V&V) model.
- The V model (though essentially a waterfall with sequential phases) is different from it, in that testing forms an important part of the model to arrive at a better quality product/project.
- The V-model is SDLC model where execution of processes happens in a sequential manner in V shape. It is also known as Verification and Validation model.
- The V model is an extension of the waterfall model and is based on association of a testing phase for each corresponding development stage. This means that for every single phase in the development cycle there is a directly associated testing phase.
- The V model is a highly disciplined model and next phase starts only after completion of the previous phase.

- The V model consists of a number of phases. The Verification Phases are on the left hand side of the V, the Coding Phase is at the bottom of the V and the Validation Phases are on the right hand side of the V, (See Fig. 1.13).
- Fig. 1.13 shows the different phases in V model of SDLC.

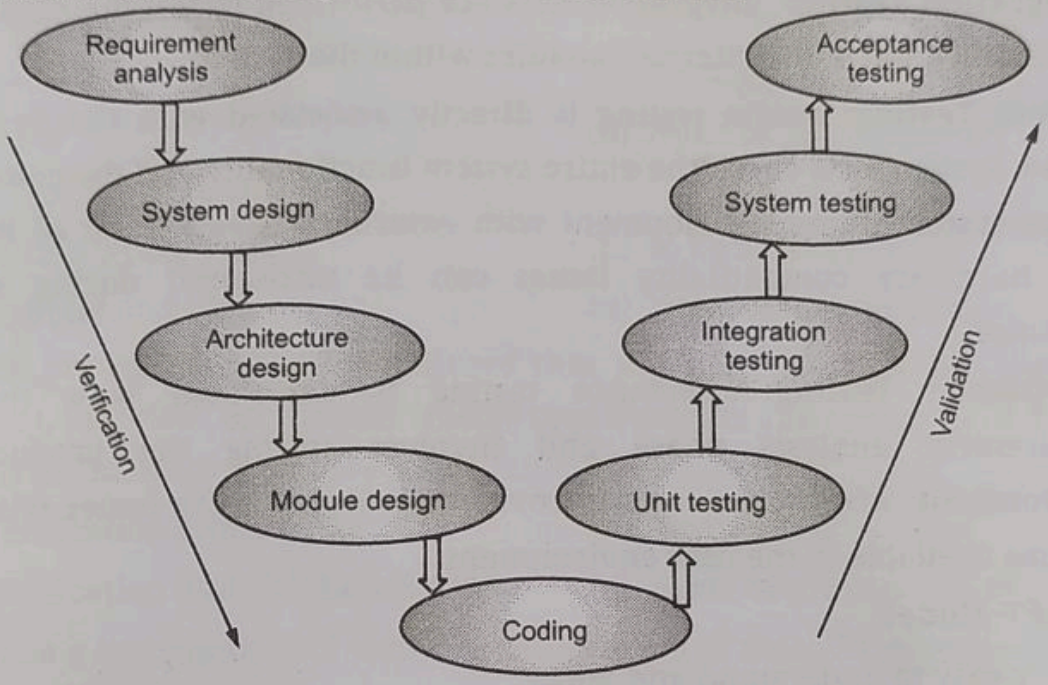

Fig. 1.13: V Model

- Fig. 1.13 shows following phases of V model.

1. **Verification Phases:** Following are the Verification phases in V-Model:
   - (i) **Requirement Analysis:** This is the first phase in the development cycle where the product requirements are understood from the customer perspective. This phase involves detailed communication with the customer to understand his expectations and exact requirement.
   - (ii) **System Design:** Once, you have the clear and detailed product require-ments, it is time to design the complete system. System design would comprise of understanding and detailing the complete hardware and communication setup for the product under development.
   - (iii) **Architectural Design:** Architectural specifications are understood and designed in this phase. System design is broken down further into modules taking up different functionality. This is also referred to as High Level Design (HLD).
   - (iv) **Module Design:** In this phase the detailed internal design for all the system modules is specified, referred to as Low Level Design
   - (v) **Coding Phase:** The actual coding of the system modules designed in the design phase is taken up in the Coding phase.

2. **Validation Phases:** Following are the Validation phases in V Model:
   (i) **Unit Testing:** Unit tests designed in the module design phase are executed on the code during this validation phase.
   (ii) **Integration Testing:** Integration tests are performed to test the coexistence and communication of the internal modules within the system.
   (iii) **System Testing:** System testing is directly associated with the System design phase. System tests check the entire system functionality and the communication of the system under development with external systems. Most of the software and hardware compatibility issues can be uncovered during system test execution.
   (iv) **Acceptance Testing:** Acceptance testing is associated with the business requirement analysis phase and involves testing the product in user environment. Acceptance tests uncover the compatibility issues with the other systems available in the user environment.

Advantages of V Model:

1. It is very easy to understand and apply.
2. It works well for smaller projects where requirements are very well understood.
3. The simplicity of this model also makes it easier to manage.
4. This is a highly disciplined model and Phases are completed one at a time.
5. This model is time saving and quick.
6. This model avoids the downward flow of the defects.
7. In this model testing activities like planning, test designing happens well before coding.

Disadvantages of V Model:

1. The model is not flexible to changes.
2. This model has high risk and uncertainty.
3. This model is poor for long and ongoing projects.
4. In this model once, an application is in the testing stage, it is difficult to go back and change functionality.
5. V-model is very rigid.
6. This model is not a good model for complex and object-oriented projects.
7. Software is developed during the implementation phase, so no early prototypes of the software are produced.