Software Testing

Unit – 1

Introduction

What is Software Testing?

- ✓Finding defects
- ✓Finding and reporting defects
- ✓Demonstrating correct functionality
- ✓Demonstrating incorrect functionality
- ✓Demonstrating robustness, reliability, security, maintainability, …
- ✓Measuring performance, reliability, …
- ✓Evaluating and measuring quality
- ✓Proving the software correct
- ✓Executing pre-defined test cases
- ✓Automatic error detection

Definition of Software Testing and its Role

Definition of Software Testing:

- Software Testing is a method to check whether the actual software product matches expected requirements and to ensure that software product is Defect free.
- It involves execution of software/system components using manual or automated tools to evaluate one or more properties of interest.
- The purpose of software testing is to identify errors, gaps or missing requirements in contrast to actual requirements.
- Testing is a the process of exercising a software component using a selected set of test cases, with the intent of revealing defects and evaluating quality.

Why Software Testing is Important?

- Software Testing is Important because if there are any bugs or errors in the software, it can be identified early and can be solved before delivery of the software product. Properly tested software product ensures reliability, security and high performance which further results in time saving, cost effectiveness and customer satisfaction.

Role of testing / Objectives of testing:

1. Finding defects which may get created by the programmer while developing the software.

2. Gaining confidence in and providing information about the level of quality.

3. To prevent defects.

4. To make sure that the end result meets the business and user requirements.

5. To ensure that it satisfies the BRS that is Business Requirement Specification and SRS that is System Requirement Specifications.

6. To gain the confidence of the customers by providing them a quality product

Terms: Failure, Error, Fault, Defect, Bug

FAILURE

A failure is the inability of a software system or component to perform its required functions within specified performance requirements. When a defect reaches the end customer it is called a Failure. During development, Failures are usually observed by testers.

ERROR

An error is a mistake, misconception, or misunderstanding on the part of a software developer. In the category of the developer, we include software engineers, programmers, analysts, and testers. For example, a developer may misunderstand a de-sign notation, or a programmer might type a variable name incorrectly – leads to an Error. It is the one that is generated because of the wrong login, loop or syntax. The error normally arises in software; it leads to a change in the functionality of the program.

FAULT

An incorrect step, process or data definition in a computer program that causes the program to perform in an unintended or unanticipated manner. A fault is introduced into the software as the result of an error. It is an anomaly in the software that may cause it to behave incorrectly, and not according to its specification. It is the result of the error.

Example: a fault is a crash and may be caused by dividing by 0, accessing bad memory, reusing a deleted pointer(address) etc.

DEFECT

It can be simply defined as a variance between expected and actual. The defect is an error found AFTER the application goes into production. It commonly refers to several troubles with the software products, with their external behavior or with its internal features. In other words, a Defect is a difference between expected and actual results in the context of testing. It is the deviation of the customer requirement.

BUG

A bug is the result of a coding error. An Error found in the development environment before the product is shipped to the customer. A programming error that causes a program to work poorly, produce incorrect results or crash. An error in software or hardware that causes a program to malfunction. A bug is the terminology of Tester.

Nature of errors OR Categories of Software Errors:

- User interface errors such as output errors or incorrect user messages
- Function errors
- Hardware defects
- Incorrect program version
- Requirements errors
- Design errors
- Documentation errors
- Architecture errors
- Module interface errors
- Performance errors
- Logic errors such as calculation errors

Why do defects occur in software?

Software is written by human beings

- Who know something, but not everything
- Who have skills, but aren’t perfect
- Who don’t usually use rigorous methods
- Who do make mistakes (errors)

Under increasing pressure to deliver to strict deadlines

- No time to check, assumptions may be wrong
- Systems may be incomplete

Software is complex, abstract and invisible

- Hard to understand
- Hard to see if it is complete or working correctly
- No one person can fully understand large systems
- Numerous external interfaces and dependencies

Sources of defects

Education

- Developers does not understand well enough what he or she is doing
- Lack of proper education leads to errors in specification, design, coding, and testing

Communication

- Developers do not know enough
- Information does not reach all stakeholders
- Information is lost

Oversight

- Omitting to do necessary things

Transcription

- Developer knows what to do but simply makes a mistake

Process

- Process is not applicable for the actual situation
- Process places restrictions that cause errors

Goals of Testing

The main goal of software testing is to find bugs as early as possible and fix bugs and make sure that the software is bug-free. The goals of software testing may be classified into three major categories as follows:

1. Immediate Goals
2. Long-term Goals
3. Post-Implementation Goals

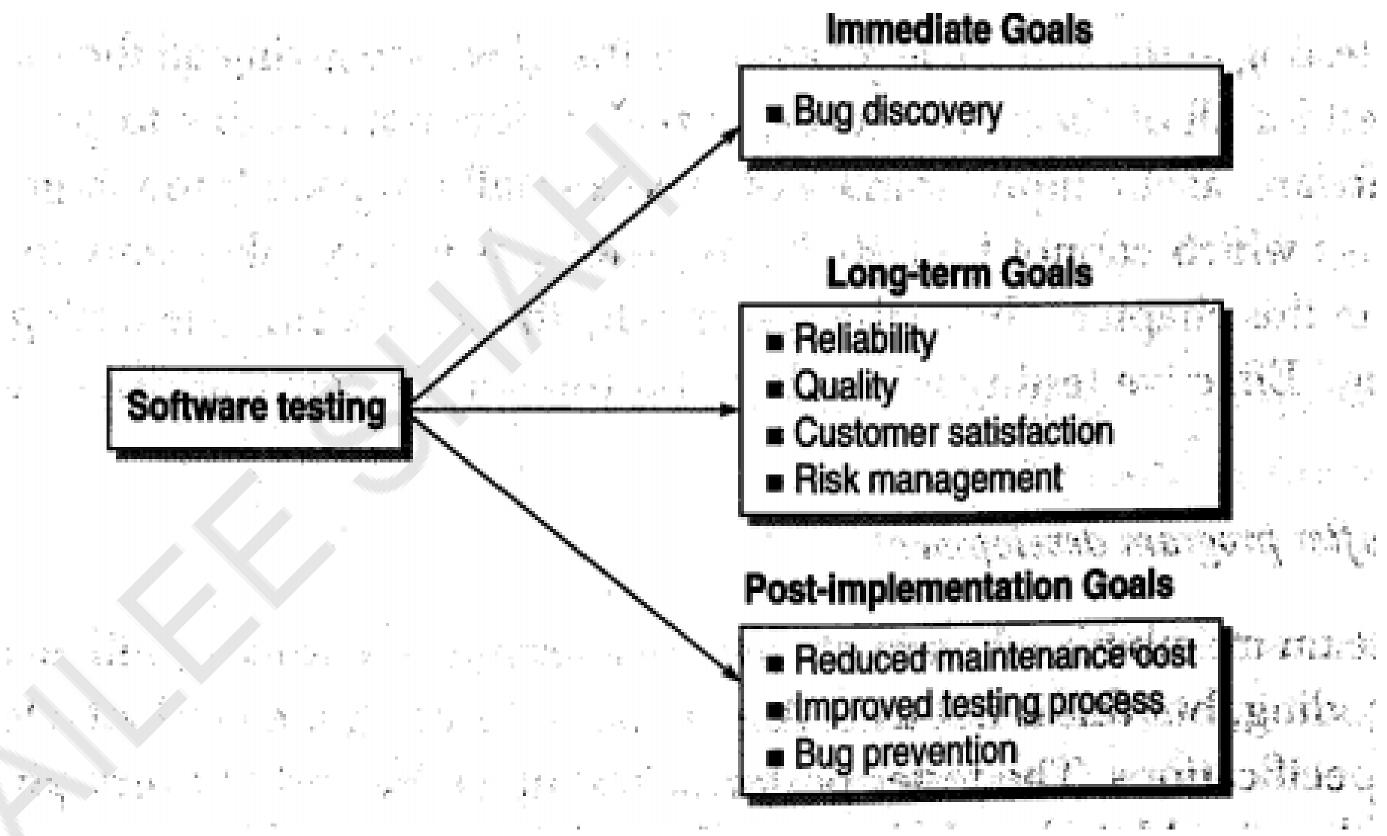


Figure 1: Software Testing Goals

1. Immediate Goals: These objectives are the direct outcomes of testing. These objectives may be set at any time during the SDLC process. Some of these are covered in detail below:

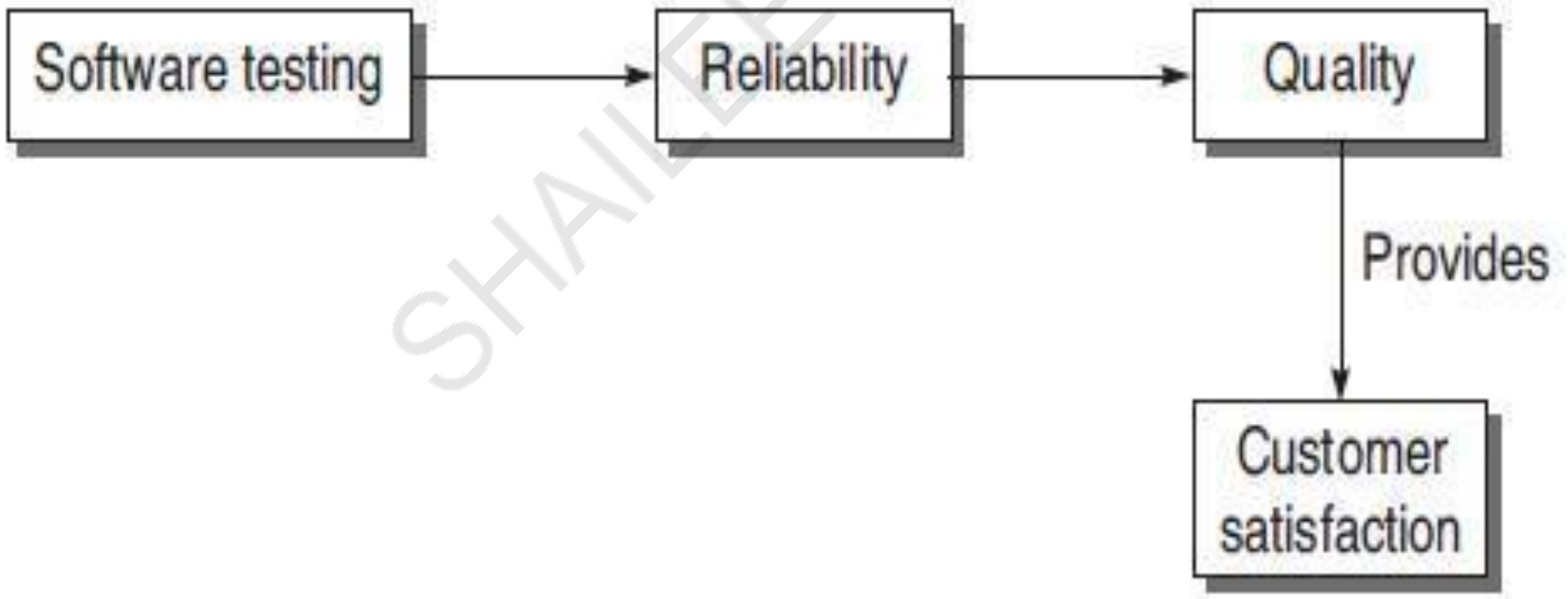
Bug Discovery: This is the immediate goal of software testing to find errors at any stage of software development. The number of bugs is discovered in the early stage of testing. The primary purpose of software testing is to detect flaws at any step of the development process. The higher the number of issues detected at an early stage, the higher the software testing success rate.

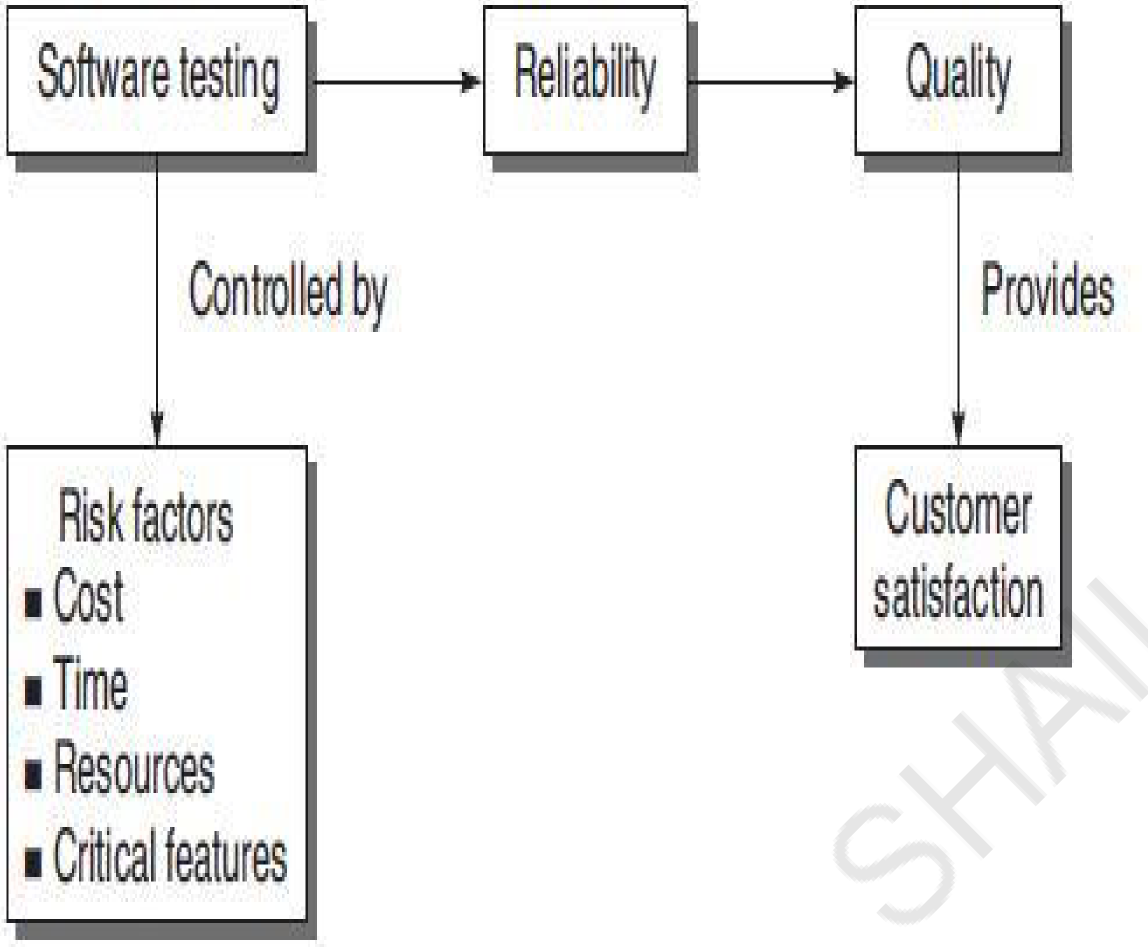
Bug Prevention: This is the immediate action of bug discovery, that occurs as a result of bug discovery. Everyone in the software development team learns how to code from the behavior and analysis of issues detected, ensuring that bugs are not duplicated in subsequent phases or future projects.

2. Long-Term Goals: These objectives have an impact on product quality in the long run after one cycle of the SDLC is completed. Some of these are covered in detail below:

Quality: This goal enhances the quality of the software product. Because software is also a product, the user’s priority is its quality. Superior quality is ensured by thorough testing. Correctness, integrity, efficiency, and reliability are all aspects that influence quality. To attain quality, you must achieve all of the above-mentioned quality characteristics.

Customer Satisfaction: This goal verifies the customer’s satisfaction with a developed software product. The primary purpose of software testing, from the user’s standpoint, is customer satisfaction. Testing should be extensive and thorough if we want the client and customer to be happy with the software product.





- **Reliability:** It is a matter of confidence that the software will not fail. In short, reliability means gaining the confidence of the customers by providing them with a quality product.
- **Risk Management:** Risk is the possibility of a negative or undesirable outcome or event.
- We need to handle risk because if it happens, then it may cause very negative impact. For example, loosing customers, dissatisfied client and other stakeholders of project.

3. **Post Implemented Goals:** After the product is released, these objectives become critical. Some of these are covered in detail below:

Reduce Maintenance Cost: Post-released errors are costlier to fix and difficult to identify. Because effective software does not wear out, the maintenance cost of any software product is not the same as the physical cost. The failure of a software product due to faults is the only expense of maintenance. Because they are difficult to discover, post-release mistakes always cost more to rectify. As a result, if testing is done thoroughly and effectively, the risk of failure is lowered, and maintenance costs are reduced as a result.

Improved Software Testing Process: These goals improve the testing process for future use or software projects. These goals are known as post-implementation goals. A project's testing procedure may not be completely successful, and there may be room for improvement. As a result, the bug history and post-implementation results can be evaluated to identify stumbling blocks in the current testing process that can be avoided in future projects.

Principles of Testing

- Testing shows the presence of defects
- Exhaustive testing is not possible
- Early testing
- Defect clustering
- Pesticide paradox
- Testing is context-dependent
- Absence of errors fallacy

Testing shows the presence of defects: The goal of software testing is to make the software fail. Software testing reduces the presence of defects. Software testing talks about the presence of defects and doesn't talk about the absence of defects. Software testing can ensure that defects are present but it cannot prove that software is defect-free. Even multiple testing can never ensure that software is 100% bug-free. Testing can reduce the number of defects but not remove all defects.

Exhaustive testing is not possible: It is the process of testing the functionality of the software in all possible inputs (valid or invalid) and pre-conditions is known as exhaustive testing. Exhaustive testing is impossible means the software can never test at every test case. It can test only some test cases and assume that the software is correct and it will produce the correct output in every test case. If the software will test every test case then it will take more cost, effort, etc., which is impractical.

Early Testing: To find the defect in the software, early test activity shall be started. The defect detected in the early phases of SDLC will be very less expensive. For better performance of software, software testing will start at the initial phase i.e. testing will perform at the requirement analysis phase.

Defect clustering: In a project, a small number of modules can contain most of the defects. Pareto Principle to software testing state that 80% of software defect comes from 20% of modules.

Pesticide paradox: Repeating the same test cases, again and again, will not find new bugs. So it is necessary to review the test cases and add or update test cases to find new bugs.

Testing is context-dependent: The testing approach depends on the context of the software developed. Different types of software need to perform different types of testing. For example, The testing of the e-commerce site is different from the testing of the Android application.

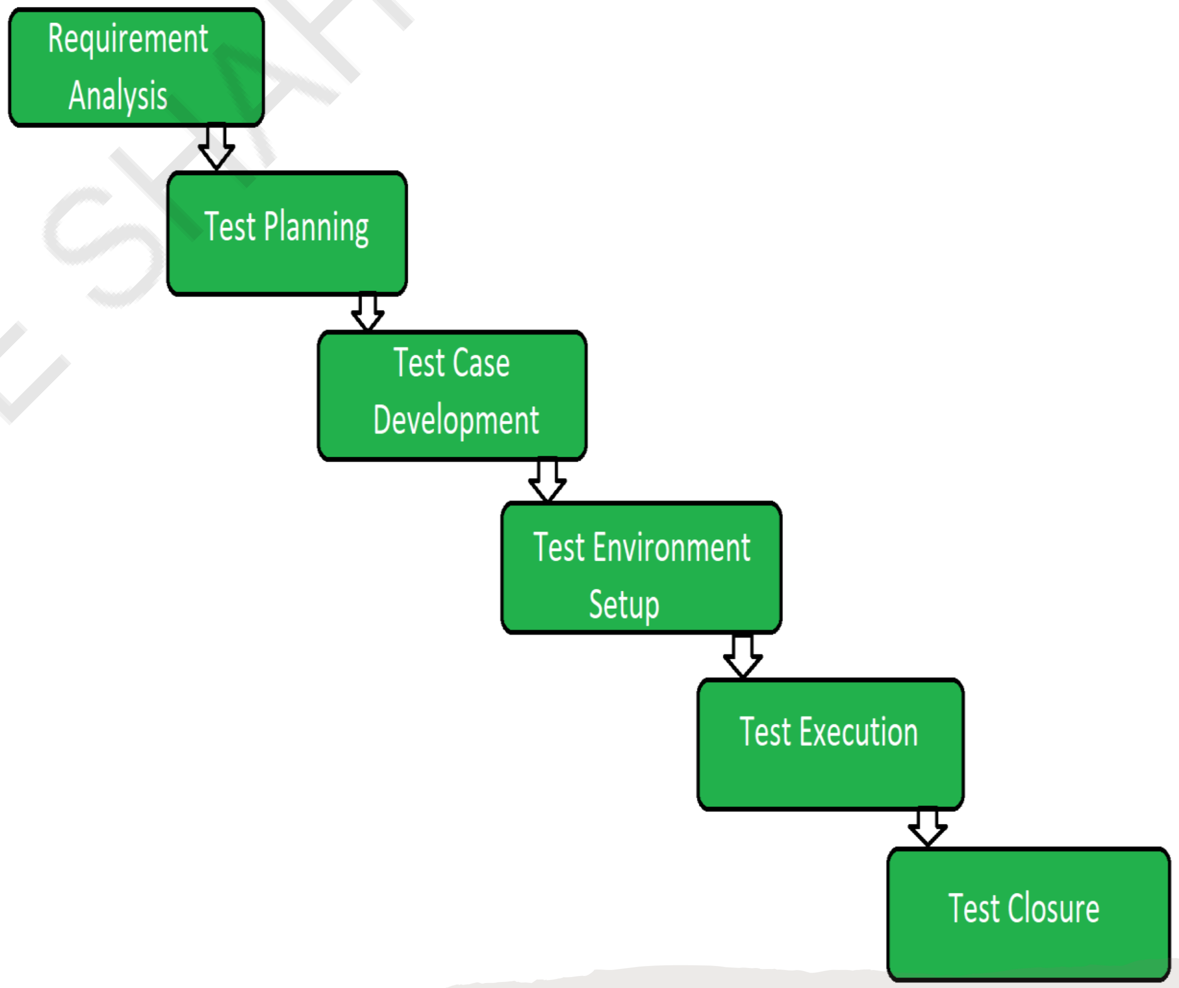
Absence of errors fallacy: If a built software is 99% bug-free but it does not follow the user requirement then it is unusable. It is not only necessary that software is 99% bug-free but it is also mandatory to fulfill all the customer requirements.

Software Testing Life Cycle

The procedure of software testing is also known as STLC (Software Testing Life Cycle) which includes phases of the testing process. The testing process is executed in a well-planned and systematic manner.

Software testing life cycle contains the following steps:

1. Requirement Analysis
2. Test Planning
3. Test Case Development
4. Test Environment Setup
5. Text Execution
6. Test Closure



Requirement Analysis:

Requirement Analysis is the first step of Software Testing Life Cycle (STLC). In this phase quality assurance team understands the requirements like what is to be tested. If anything is missing or not understandable then quality assurance team meets with the stakeholders to better understand the detail knowledge of requirement. It is done with the help of SRS documentation(software requirement specification).

Test Planning:

Test Planning is most efficient phase of software testing life cycle where all testing plans are defined. In this phase manager of the testing team calculates estimated effort and cost for the testing work. This phase gets started once the requirement gathering phase is completed.

Test Case Development:

The test case development phase gets started once the test planning phase is completed. In this phase testing team note down the detailed test cases. Testing team also prepare the required test data for the testing. When the test cases are prepared then they are reviewed by quality assurance team.

Test Environment Setup:

Test environment setup is the vital part of the STLC. Basically test environment decides the conditions on which software is tested. This is independent activity and can be started along with test case development. In this process the testing team is not involved. either the developer or the customer creates the testing environment.

Test Execution:

After the test case development and test environment setup test execution phase gets started. In this phase testing team start executing test cases based on prepared test cases in the earlier step.

Test Closure:

This is the last stage of STLC in which the process of testing is analyzed.

Characteristics of STLC:

STLC is a fundamental part of Software Development Life Cycle (SDLC)but STLC consists of only the testing phases.

STLC starts as soon as requirements are defined or software requirement document is shared by stakeholders.

STLC yields a step-by-step process to ensure quality software.

While the software or the product is developing in the early stage of STLC, the tester can analyse and define the scope of Testing, entry and exit criteria and also the Test Cases. It helps in reducing the Test Cycle time and provide better quality.

When development phase is over, the Testers are ready with Test Cases and start with execution. This helps in finding bugs in the initial phase.

Verification and Validation: V-testing Life cycle

Verification

Verification testing includes different activities such as business requirements, system requirements, design review, and code walkthrough while developing a product.

It is also known as static testing, where we are ensuring that "we are developing the right product or not". And it also checks that the developed application fulfilling all the requirements given by the client.

This is done by **Developer.**

Methods of Verification : Static Testing

A walkthrough of verification of a mobile application

There are three phases in the verification testing of a mobile application development:

- Requirements Verification
- Design Verification
- Code Verification

- **Requirements verification** is the process of verifying and confirming that the requirements are complete, clear, and correct. Before the mobile application goes for design, the testing team verifies business requirements or customer requirements for their correctness and completeness.
- **Design verification** is a process of checking if the design of the software meets the design specifications by providing evidence. Here, the testing team checks if layouts, prototypes, navigational charts, architectural designs, and database logical models of the mobile application meet the functional and non-functional requirements specifications.
- **Code verification** is a process of checking the code for its completeness, correctness, and consistency. Here, the testing team checks if construction artifacts such as source code, user interfaces, and database physical model of the mobile application meet the design specification.

Validation

- Validation testing is testing where tester performed functional and non-functional testing. Here functional testing includes Unit Testing (UT), Integration Testing (IT) and System Testing (ST), and non-functional testing includes User acceptance testing (UAT).
- Validation testing is also known as dynamic testing, where we are ensuring that "we have developed the product right." And it also checks that the software meets the business needs of the client.
- Methods of Verification : Dynamic Testing
- This is done by **Tester**.

A walkthrough of validation of a mobile application

- Validation emphasizes checking the functionality, usability, and performance of the mobile application.

- **Functionality testing** checks if the mobile application is working as expected. For instance, while testing the functionality of a ticket-booking application, the testing team tries to validate it through:
- Installing, running, and updating the application from distribution channels like Google Play and the App Store
- Booking tickets in the real-time environment (fields testing)
- Interruptions testing
- **Usability testing** checks if the application offers a convenient browsing experience. User interface and navigations are validated based on various criteria which include satisfaction, efficiency, and effectiveness.
- **Performance testing** enables testers to validate the application by checking its reaction and speed under the specific workload. Software testing teams often use techniques such as load testing, stress testing, and volume testing to validate the performance of the mobile application.
- **Note: Verification and Validation process are done under the V model of the software development life cycle.**

Verification	Validation
We check whether we are developing the right product or not.	We check whether the developed product is right.
Verification is also known as static testing.	Validation is also known as dynamic testing.
Verification includes different methods like Inspections, Reviews, and Walkthroughs.	Validation includes testing like functional testing, system testing, integration, and User acceptance testing.
It is a process of checking the work-products (not the final product) of a development cycle to decide whether the product meets the specified requirements.	It is a process of checking the software during or at the end of the development cycle to decide whether the software follow the specified business requirements.
Quality assurance comes under verification testing.	Quality control comes under validation testing.
The execution of code does not happen in the verification testing.	In validation testing, the execution of code happens.
In verification testing, we can find the bugs early in the development phase of the product.	In the validation testing, we can find those bugs, which are not caught in the verification process.
Verification testing is executed by the Quality assurance team to make sure that the product is developed according to customers' requirements.	Validation testing is executed by the testing team to test the application.
Verification is done before the validation testing.	After verification testing, validation testing takes place.
In this type of testing, we can verify that the inputs follow the outputs or not.	In this type of testing, we can validate that the user accepts the product or not.

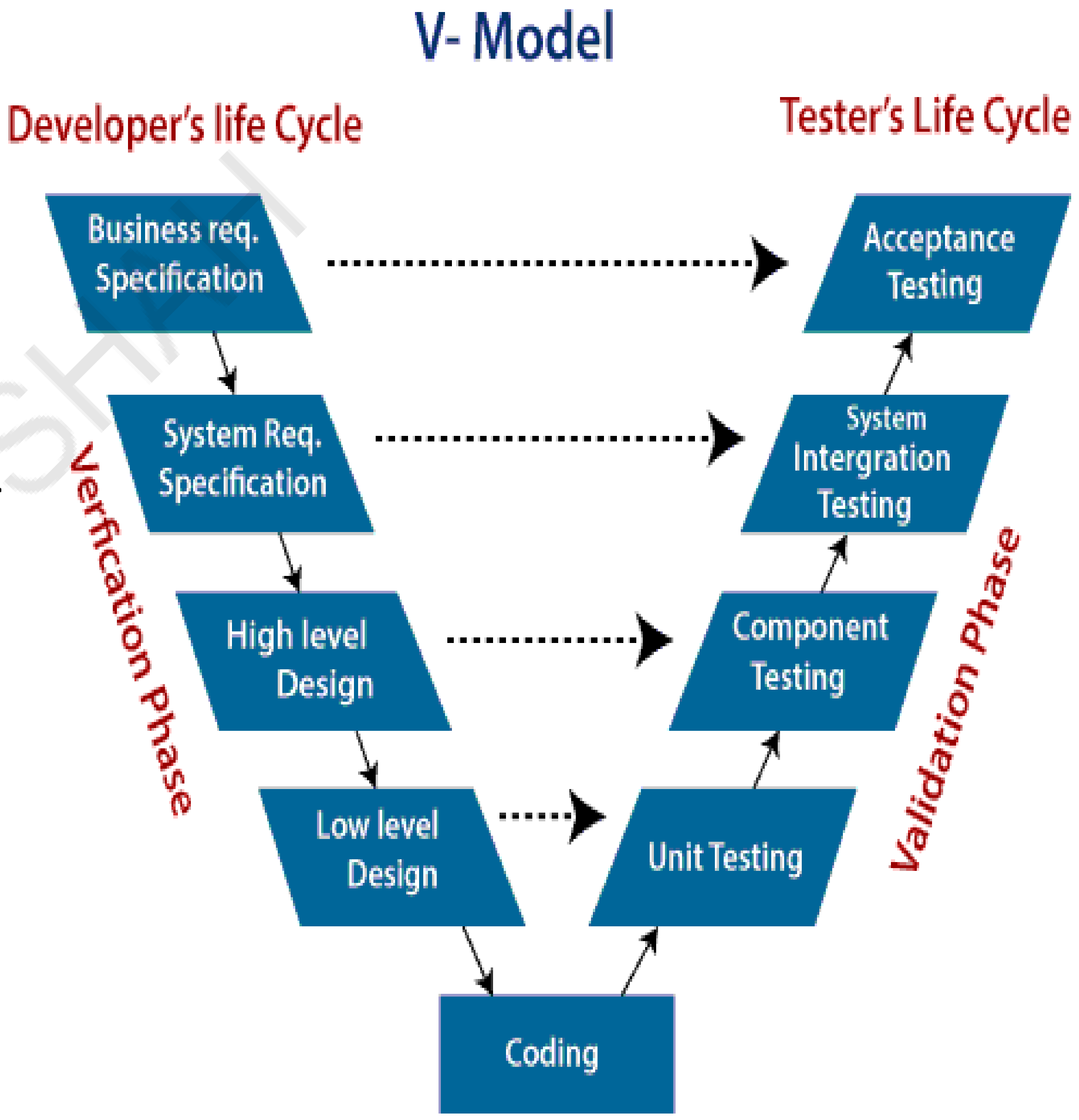
V-testing Life cycle

The V-model is an SDLC model where execution of processes happens in a sequential manner in a 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 the 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. This is a highly-disciplined model and the next phase starts only after completion of the previous phase.

V-Model - Design

Under the V-Model, the corresponding testing phase of the development phase is planned in parallel. So, there are Verification phases on one side of the 'V' and Validation phases on the other side. The Coding Phase joins the two sides of the V-Model.



There are the various phases of Verification Phase of V-model:

Business requirement analysis: This is the first step where product requirements understood from the customer's side. This phase contains detailed communication to understand customer's expectations and exact requirements.

System Design: In this stage system engineers analyze and interpret the business of the proposed system by studying the user requirements document.

Architecture Design: The baseline in selecting the architecture is that it should understand all which typically consists of the list of modules, brief functionality of each module, their interface relationships, dependencies, database tables, architecture diagrams, technology detail, etc. The integration testing model is carried out in a particular phase.

Module Design: In the module design phase, the system breaks down into small modules. The detailed design of the modules is specified, which is known as Low-Level Design

Coding Phase: After designing, the coding phase is started. Based on the requirements, a suitable programming language is decided. There are some guidelines and standards for coding. Before checking in the repository, the final build is optimized for better performance, and the code goes through many code reviews to check the performance.

Unit Testing: In the V-Model, Unit Test Plans (UTPs) are developed during the module design phase. These UTPs are executed to eliminate errors at code level or unit level. A unit is the smallest entity which can independently exist, e.g., a program module. Unit testing verifies that the smallest entity can function correctly when isolated from the rest of the codes/ units.

Integration Testing: Integration Test Plans are developed during the Architectural Design Phase. These tests verify that groups created and tested independently can coexist and communicate among themselves.

System Testing: System Tests Plans are developed during System Design Phase. Unlike Unit and Integration Test Plans, System Tests Plans are composed by the client's business team. System Test ensures that expectations from an application developer are met.

Acceptance Testing: Acceptance testing is related to the business requirement analysis part. It includes testing the software product in user atmosphere. Acceptance tests reveal the compatibility problems with the different systems, which is available within the user atmosphere. It conjointly discovers the non-functional problems like load and performance defects within the real user atmosphere.

Static and Dynamic Testing

Static Testing

Static testing is testing, which checks the application without executing the code. It is a verification process. Some of the essential activities are done under static testing such as business requirement review, design review, code walkthroughs, and the test documentation review.

Static testing is performed in the white box testing phase, where the programmer checks every line of the code before handling over to the Test Engineer.

Static testing can be done manually or with the help of tools to improve the quality of the application by finding the error at the early stage of development; that why it is also called the verification process.

The documents review, high and low-level design review, code walkthrough take place in the verification process.

Dynamic Testing

Dynamic testing is testing, which is done when the code is executed at the run time environment. It is a validation process where functional testing [unit, integration, and system testing] and non-functional testing [user acceptance testing] are performed.

We will perform the dynamic testing to check whether the application or software is working fine during and after the installation of the application without any error.

Static testing	Dynamic testing
In static testing, we will check the code or the application without executing the code.	In dynamic testing, we will check the code/application by executing the code.
Static testing includes activities like code Review, Walkthrough, etc.	Dynamic testing includes activities like functional and non-functional testing such as UT (usability testing), IT (integration testing), ST (System testing) & UAT (user acceptance testing).
Static testing is a Verification Process.	Dynamic testing is a Validation Process.
Static testing is used to prevent defects.	Dynamic testing is used to find and fix the defects.
Static testing is a more cost-effective process.	Dynamic testing is a less cost-effective process.
This type of testing can be performed before the compilation of code.	Dynamic testing can be done only after the executables are prepared.
Under static testing, we can perform the statement coverage testing and structural testing.	Equivalence Partitioning and Boundary Value Analysis technique are performed under dynamic testing.
It involves the checklist and process which has been followed by the test engineer.	This type of testing required the test case for the execution of the code.