

Software Engineering And Testing

Software Testing



Outline

- Coding Standards and Guidelines
- Code Review[Code Walkthrough, Code Inspection, Clean Room Testing]
- Software Documentation
- Verification, Verification of Requirements
- High-level and Low-Level Design
- How to Verify Code?
- Validation, Validation Activities: Unit Testing, Integration Testing, Function Testing, System Testing, Acceptance Testing,
- Overview of Regression Testing
- Introduction to Testing, Effective and Exhaustive Testing

Outline

- Software Testing Life Cycle (STLC)
- Definition and Goals of Testing, Black-box Testing [Equivalence Class Partitioning, Boundary Value Analysis]
- White-Box Testing [Basic Concepts, Statement Coverage, Branch Coverage, Multiple Condition Coverage, Path Coverage, McCabe's Cyclomatic Complexity Metric, Data Flow-based Testing]
- Debugging
- Introduction Program Analysis Tools - Static and Dynamic,
- Integration Testing
- Testing Object-Oriented Programs
- Smoke Testing

Outline

- Issues in OOT
- Issues in testing Inheritance
- Various OO Testing Techniques
- Test Organization, Structure of Testing Group, Test Planning, Detailed Test Design and Test Specifications.

We trust in GOD
Everything Else we TEST

Coding Standards and Guideline

- **A Coding Standards** : sets out standard ways of doing several things:
 - the way variables are named,
 - code is laid out,
 - maximum number of source lines allowed per function, etc.
- **Purpose of Having Coding Standards:**
 - A coding standard gives a uniform appearance to the codes written by different engineers.
 - It improves readability, and maintainability of the code and it reduces complexity also.
 - It helps in code reuse and helps to detect error easily.
 - It promotes sound programming practices and increases efficiency of the programmers.

Coding Standards and Guideline

- General coding standards refers to how the developer writes code, some essential standards regardless of the programming language are.

Coding Standards



Coding Standards and Guideline

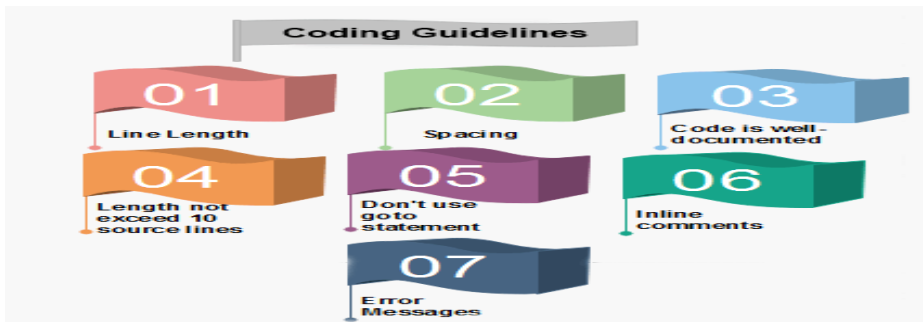
- **Indentation:** Proper and consistent indentation is essential in producing easy to read and maintainable programs.
Indentation should be used to:
 - Emphasize the body of a control structure such as a loop or a select statement.
 - Emphasize the body of a conditional statement
 - Emphasize a new scope block
- **Inline comments:** Inline comments analyze the functioning of the subroutine, or key aspects of the algorithm shall be frequently used.
- **Rules for limiting the use of global:** These rules file what types of data can be declared global and what cannot.

Coding Standards and Guideline

- **Structured Programming:** Structured (or Modular) Programming methods shall be used. "GOTO" statements shall not be used as they lead to "spaghetti" code, which is hard to read and maintain, except as outlined line in the FORTRAN Standards and Guidelines.
- **Naming conventions for global variables, local variables, and constant identifiers:** A possible naming convention can be that global variable names always begin with a capital letter, local variable names are made of small letters, and constant names are always capital letters.
- **Error return conventions and exception handling system:** Different functions in a program report the way error conditions are handled should be standard within an organization. For example, different tasks while encountering an error condition should either return a 0 or 1 consistently.

Coding Guideline

- Coding guidelines provide the programmer with a set of the best methods which can be used to make programs more comfortable to read and maintain.
- Most of the examples use the C language syntax, but the guidelines can be tested to all languages.
- The following are some representative coding guidelines recommended by many software development organizations.



Software Testing

Testing is the **process** of exercising a program with the specific **intent of finding errors** prior to delivery to the end user.

Don't view testing as a **"safety net"** that will catch all errors that occurred because of weak software engineering practice.



Objective of Software Testing



Code inspection and code walk throughs

- **After a module has been coded:**
 - code inspection and code walk through are carried out
 - ensures that coding standards are followed
 - helps detect as many errors as possible before testing.
- **Detect as many errors as possible during inspection and walkthrough:**
 - detected errors require less effort for correction
 - much higher effort needed if errors were to be detected during integration or system testing.
- The main objective of code walkthrough **is to discover the algorithmic and logical errors in the code.**
- **A few members of the development team select some test cases:** simulate execution of the code by hand using these test cases.

Code inspection and code walk throughs

- The principal aim of code inspection is to check for the presence of some common types of errors that usually creep into code due to programmer mistakes and oversights and to check whether coding standards have been adhered to.
- **Commonly Made Errors**
 - Use of uninitialized variables.
 - Non-terminating loops.
 - Array indices out of bounds.
 - Incompatible assignments.
 - Improper storage allocation and deallocation.
 - Actual and formal parameter mismatch in procedure calls.
 - Jumps into loops.

Code inspection and code walk throughs

- **Commonly Made Errors**
 - **Use of incorrect logical operators** or incorrect precedence among operators.
 - **Improper modification of loop variables.**
 - **Comparison of equality of floating point values, etc.**
 - **Also during code inspection, adherence to coding standards is checked.**

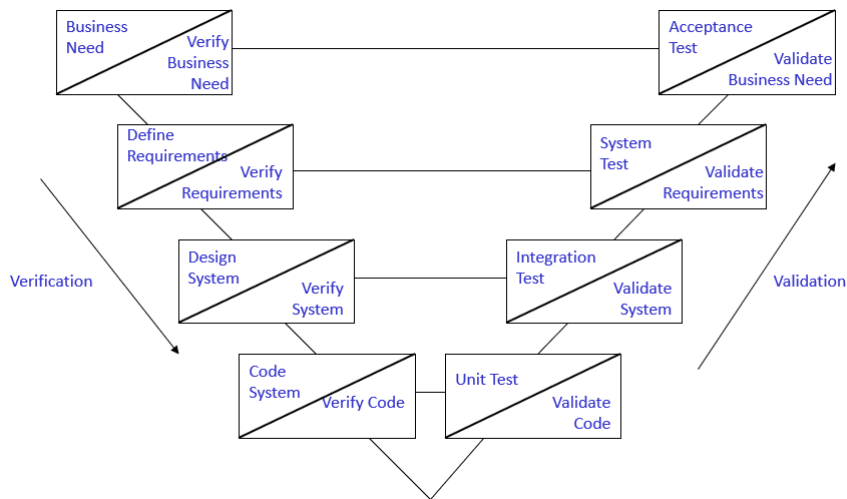
Cleanroom Technique

- **Commonly Made Errors**
- The cleanroom process was originally developed by **Harlan Mills** and several of his colleagues including **Alan Hevner at IBM**.
- The focus of the cleanroom process is on **defect prevention, rather than defect removal**.
- The name "cleanroom" was chosen to invoke the cleanrooms used in the electronics industry to prevent the introduction of defects during the fabrication of semiconductors.
- The computer code development philosophy relies on avoiding computer code defects by employing a rigorous examination method.
- The target of this computer code is that of the zero-defect computer code.

Software Documentation

- **When developing a software product we develop various kinds of documents :**
 - In addition to executable files and the source code:
 - **users' manual,**
 - **software requirements specification (SRS) document,**
 - **design document, test document,**
 - **installation manual, etc.**
- **All these documents are a vital part of good software development practice.**
- **Good documents enhance understandability and maintainability of a software product.**
- **Different types of software documents can be classified into:** internal documentation, external documentation (supporting documents).

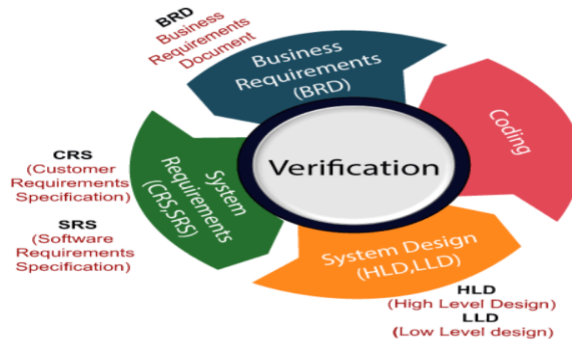
V'Life Cycle Model



17

Verification Activities

- All verification activities are performed in connection with the different phases of SDLC as follows:
 - Verification of Requirements and Objectives
 - Verification of High-Level Design
 - Verification of Low-Level Design
 - Verification of Coding (Unit Verification)



18

Verification Requirements

- All the requirements gathered from the user's viewpoint are verified.
- An acceptance criterion is prepared for that, which defines the goals and requirements of the proposed system and acceptance limits for each of the goals and requirements.
- Acceptance criterion is most important in case of real-time systems where performance is a critical issue in certain events.
- So it must be defined by the designers of the system and should not be overlooked, as they can create problems while testing the system.
- The tester works in parallel by performing the following two tasks:
 - The tester reviews the acceptance criteria in terms of its completeness, clarity, and test-ability so that necessary resources can be planned.
 - The tester prepares the Acceptance Test Plan which is referred at the time of Acceptance Testing.

How to Verify of High Level Design?

- Here there is highest possibility of finding bugs.
- So a formal specification of design is required known as SDD (S/W Design Document) according to the standard provided by IEEE.
- If bug is not detected in HLD, its fixing cost increases with every phase so verification of HLD is crucial.
- This design is divided in three parts:
 - Data Design
 - Architectural Design
 - Interface Design

How to Verify Low Level Design?

- A last preceding phase where internal details of each design entity are described:
 - Verify the SRS of each module.
 - Verify the SDD of each module.
 - In LLD, data structures, interfaces and algorithms are represented by design notations; so verify the consistency of every item with their design notations.
- Organizations can build a two-way traceability matrix between the SRS and design (both HLD and LLD) such that at the time of verification of the design each requirement in SRS is verified.

21

How to Verify Code?

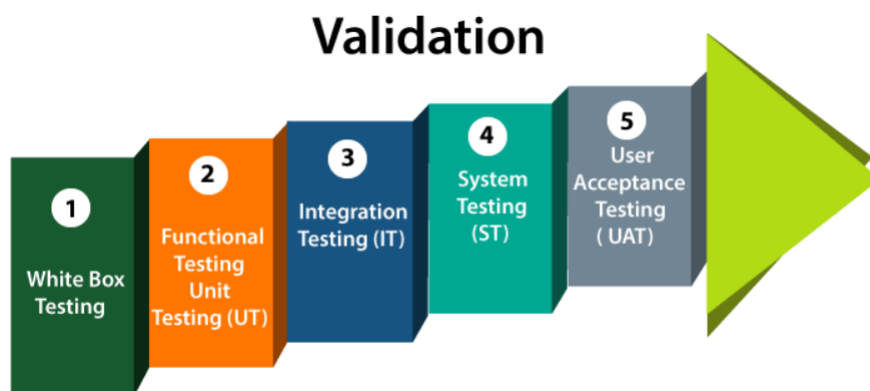
- Check that every design specification in HLD and LLD has been coded using traceability matrix.
- Examine the code against a language specification checklist.
- Verify every statement, control structure, loop, and logic.
- Misunderstood or incorrect Arithmetic precedence.
- Mixed mode operations.
- Incorrect initialization.
- Precision Inaccuracy.
- Incorrect symbolic representation of an expression.
- Different data types.
- Improper or nonexistent loop termination.
- Failure to exit.

22

How to Verify Code?

- Two types of techniques are used to verify the coding:
- **Static testing techniques** :No actual execution. Only static analysis of the code or type of conceptual analysis of the code
- **Dynamic testing techniques** : Complementary to static testing. It executes the code on some test data

23



Validation Activities

- **Unit Validation Testing:** the process of testing individual components of a system along with its all interfaces and functionalities.
 - A unit/module must be validated before integrating it with other modules.
 - First activity after coding phase.
- **Integration Testing:** process of combining and testing multiple components or modules together.
 - To uncover the bugs that are present when unit tested modules are integrated.
- **System Testing:** does not aim to test individual function but the system as a whole on various grounds where bugs may occur. E.g. if the system fails in some conditions, how does it recover?

25

Validation Activities

- **Acceptance Testing:** acceptance criteria is developed in the requirement phase and system can be tested against that criteria contracted with the customer when the system is ready.
- **Installation Testing:** Once testing team gives OK for producing the s/w, it is placed into an operational status where it is installed.
 - The installation testing does not test the system, but it tests the process of making the s/w system operational.
- **Regression Testing :** Is the process of testing the modified parts of the code and the parts that might get affected due to the modifications to ensure that no new errors have been introduced in the software after the modifications have been made. Regression means the return of something and in the software field, it refers to the return of a bug.

26

Examples of V & V

- A clickable button with name Submet.
- Verification would check the design doc and correcting the spelling mistake.
- Otherwise, the development team will create a button like



Example of Verification

27

Examples of V & V

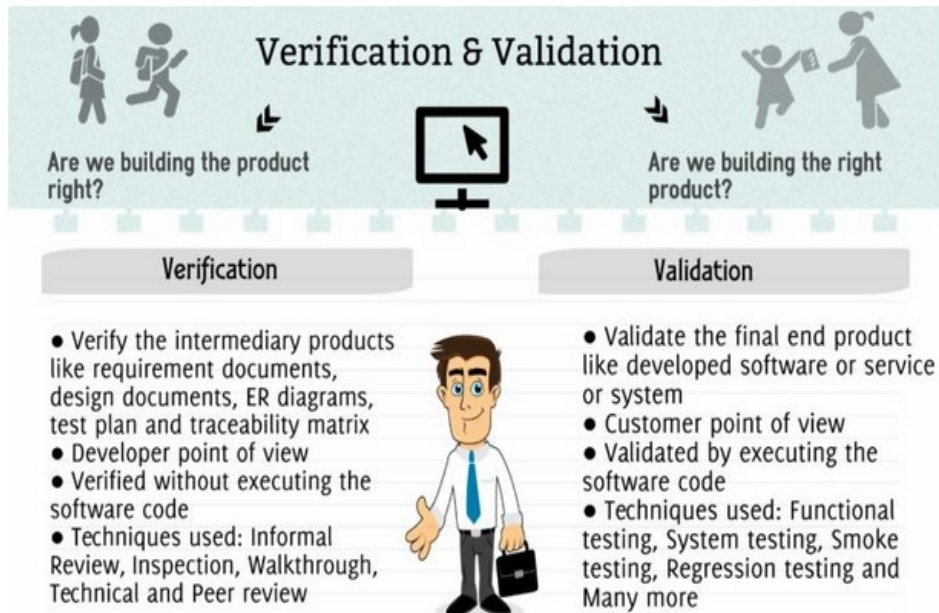
- So new specification is
- A clickable button with name Submit
- Once the code is ready, Validation is done. A Validation test found –



Example of Validation

- Owing to validation testing, the development team will make the submit button clickable.

28



Differences between verification and validation

	Verification	Validation
Definition	It is a process of checking if a product is developed as per the specifications.	It is a process of ensuring that the product meets the needs and expectations of stakeholders.
What it tests or checks for	It tests the requirements, architecture, design, and code of the software product.	It tests the usability, functionalities, and reliability of the end product.

Differences between verification and validation

	Verification	Validation
Coding requirement	It does not require executing the code.	It emphasizes executing the code to test the usability and functionality of the end product.
Activities include	A few activities involved in verification testing are requirements verification, design verification, and code verification.	The commonly-used validation activities in software testing are usability testing, performance testing, system testing, security testing, and functionality testing.

31

Differences between verification and validation

	Verification	Validation
Types of testing methods	A few verification methods are inspection, code review, desk-checking, and walkthroughs.	A few widely-used validation methods are black box testing, white box testing, integration testing, and acceptance testing.
Teams or persons involved	The quality assurance (QA) team would be engaged in the verification process.	The software testing team along with the QA team would be engaged in the validation process
Target of test	It targets internal aspects such as requirements, design, software architecture, database, and code.	It targets the end product that is ready to be deployed.

32

Effective Vs. Exhaustive Software Testing

- Exhaustive or complete software testing means that every statement in the program and every possible path combination with every possible combination of data must be executed.
- But soon, we will realize that exhaustive testing is out of scope. That is why the questions arise:
 - When are we done with testing? or
 - How do we know that we have tested enough?
- There may be many answers for these questions with respect to time, cost, customer, quality, etc.
- **Exhaustive or complete testing is not possible.**
- Therefore, we should concentrate on effective testing which emphasizes efficient techniques to test the software so that important features will be tested within the constrained resources.

33

Effective Vs. Exhaustive Software Testing

- Exhaustive Testing Examples
 - Suppose, a calculator accepts integers ranging from -1000000 to +1000000. The tester will try to input all the possible numbers in between the range. No matter how much time is consumed, the tester will test to its maximum ability to leave no scope. This will be exhaustive testing in this case.
 - Another example can be, a device can be accessed only through a code with combinations of numbers in pairs of 3. The maximum possible input combination in this case, will be 9 numbers from 0-9(no repetitions allowed) and 3 input permutations. So the number of test cases will look like- $9*9*9$ which is possible to execute. Here we have achieved what we call exhaustive testing.

34

Effective Vs. Exhaustive Software Testing

Exhaustive testing	Effective testing
Exhaustive testing aims at complete testing i.e no scope of further bug detection should be possible.	Effective testing is a smart way of prioritizing and covering all critical test cases that too with resource constraints.
It is impractical due to various reasons like time constraints, inadequate resources, achieving deadlines, etc.	It is a practical method as all deadlines can be met with almost no defects or issues in the final product.
It is not cost-effective.	It is a cost-effective method.
The whole process is quite complex and can be achieved for only smaller projects.	The process is not that complex nor time-consuming.

Seven Principles of Software Testing



Testing shows presence of Defects

Testing shows presence of defects, cannot prove absence of defects. Testing helps in finding undiscovered defects.



Exhaustive testing is Impossible

Impossible to test all possible input combinations of data and scenarios. Smarter ways of testing should be adopted



Early Testing

Start testing as soon as possible. Finding defects early on saves a lot of money rather than finding them later.



Defect Clustering

Equal distribution of bugs across the modules is not possible. Defect may be clustered in small piece of code/module.



Testing - Context Dependent

Testing is context dependent. Different websites are tested differently. Eg., Banking site tested differently than Shopping site.



Pesticide Paradox

Executing same test cases again and again will not help to identify more bugs. Review them regularly and modify if changes required



Absence-of-errors fallacy

Finding and fixing many bugs doesn't help. If it fails to meet user's requirements, it is not useful.

Software Testing Terminology and Methodology

- We tend to use the terms 'error', 'bug', 'defect', 'fault', 'failure', etc. interchangeably. But they don't mean the same.
- **Bugs** in general, are more important during software testing.
 - All the bugs do not have the same level of severity.
 - Some bugs are less important, as they are not going to affect the product badly.
 - On the other hand, bugs with high severity level will affect the product such that the product may not be released.
 - Therefore, bugs need to be classified according to their severity.
 - A bug has a complete cycle from its initiation to death.

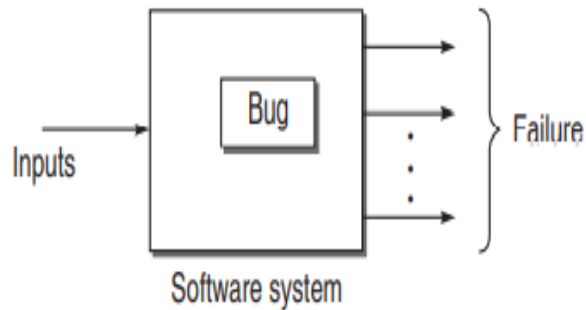
37

Software Testing Terminology and Methodology

- **Failure** : When the software is tested, failure is the first term being used.
 - It means the inability of a system or component to perform a required function according to its specification.
 - In other words, when results or behaviour of the system under test are different as compared to specified expectations, then failure exists.
- **Fault/Defect/Bug** : Fault is a condition that in actual causes a system to produce failure.
 - Fault is synonymous with the words defect or bug.
 - Therefore, fault is the reason embedded in any phase of SDLC and results in failures.
 - It can be said that failures are manifestation of bugs.

38

Software Testing Terminology and Methodology



39

Software Testing Terminology and Methodology

- **Fault/Defect/Bug :**
 - One failure may be due to one or more bugs and one bug may cause one or more failures.
 - Thus, when a bug is executed, then failures are generated.
 - But this is not always true.
 - Some bugs are hidden in the sense that these are not executed, as they do not get the required conditions in the system.
 - So, hidden bugs may not always produce failures.
 - They may execute only in certain rare conditions.

40

Why do bugs arise?



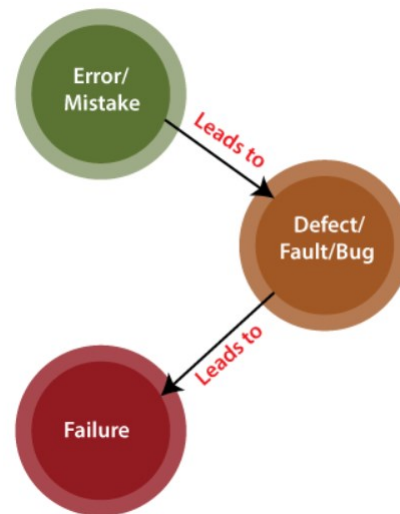
- Unclear Requirements
- Incorrect Requirements
- Complex design
- Poor skilled developer
- Complicated logic
- Insufficient time
- Less knowledge about product
- Poor skilled tester
- Changing requirements often
- Environment & System issues

BUG VERSUS DEFECT

BUG	DEFECT
A failure in a computer program that causes it to produce an incorrect or unexpected result or to behave in an unintended manner	A failure in a computer program that has a variation between the actual result and the expected result
A coding fault	A deviation from the original business requirement

Software Testing Terminology and Methodology

- **Error** : Whenever a development team member makes a mistake in any phase of SDLC, errors are produced.
 - It might be a typographical error, a misleading of a specification, a misunderstanding of what a subroutine does, and so on.
 - Error is a very general term used for human mistakes.
 - Thus, an error causes a bug and the bug in turn causes failures.



Software Testing Terminology and Methodology

Consider the following module in a software:

```

Module A()
{
    ... *

    while(a > n+1);
    {
        ...
        print("The value of x is", x);
    }

    ... *
}
  
```

Software Testing Terminology and Methodology

- Suppose the module shown above is expected to print the value of x which is critical for the use of software.
- But when this module will be executed, the value of x will not be printed. It is a failure of the program.
- When we try to look for the reason of the failure, we find that in Module A(), the while loop is not being executed.
- A condition is preventing the body of while loop to be executed.
- This is known as bug/defect/fault.
- On close observation, we find a semicolon being misplaced after the while loop which is not its correct syntax and it is not allowing the loop to execute.
- This mistake is known as an error.

45

Software Testing Terminology and Methodology

- **Test case :** Test case is a well-documented procedure designed to test the functionality of a feature in the system.
 - A test case has an identity and is associated with a program behaviour.
 - The primary purpose of designing a test case is to find errors in the system.
 - For designing the test case, it needs to provide a set of inputs and its corresponding expected outputs.

Test Case ID
Purpose
Preconditions
Inputs
Expected Outputs

46

Software Testing Terminology and Methodology

Test Case ID
Purpose
Preconditions
Inputs
Expected Outputs




- Test case ID : is the identification number given to each test case.
- Purpose : defines why the case is being designed.
- Preconditions : for running the inputs in a system can be defined, if required, in a test case.
- Inputs : should not be hypothetical.
 - Actual inputs must be provided, instead of general inputs.
 - For example, if two integer numbers have to be provided as input, then specifically mention them as 23 and 56.
- Expected outputs : are the outputs which should be produced when there is no failure.
 - Test cases are designed based on the chosen testing techniques. They provide inputs when the system is executed.
 - After execution, observed results are compared with expected outputs mentioned in the test case.

47

Bug Defect and Error



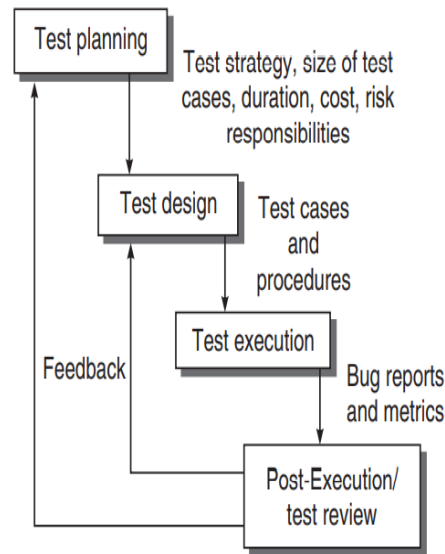
- **Difference in Bug, Defect and Error**
 - Bug is an error found BEFORE the application goes into production (By Tester).
 - Defect is an error found AFTER application goes into production. (By Customer)
 - Anything that is not confirming to expected behavior and it may be at any stage is Error.
 - An error can be a defect or Bug and deviated from Customer Requirement.

Error / Mistake	Defect / Bug/ Fault	Failure
Found by 	Found by 	Found by 
Developer	Tester	Customer

{Error, Defect & Failure}

Software Testing Life Cycle (STLC)

- The testing process divided into a well-defined sequence of steps is termed as software testing life cycle (STLC).
- The major contribution of STLC is to involve the testers at early stages of development.
- This has a significant benefit in the project schedule and cost.
- The STLC also helps the management in measuring specific milestones.



49

Software Testing Life Cycle (STLC)

- **Test Planning** : The goal of test planning is to take into account the **important issues of testing strategy**, viz. resources, schedules, responsibilities, risks, and priorities, as a roadmap.
- Broadly, following are the activities during test planning:
 - Defining the test strategy.
 - Estimate the number of test cases, their duration, and cost.
 - Plan the resources like the manpower to test, tools required, documents required.
 - Identifying areas of risks.
 - Defining the test completion criteria.
 - Identification of methodologies, techniques, and tools for various test cases.
 - Identifying reporting procedures, bug classification, databases for testing, bug severity levels, and project metrics

50

Software Testing Life Cycle (STLC)

- **Test Planning** : Based on the planning issues, analysis is done for various testing activities.
- The major output of test planning is the test plan document.
- Test plans are developed for each level of testing.
- After analyzing the issues, the following activities are performed:
 - Develop a test case format.
 - Develop test case plans according to every phase of SDLC.
 - Identify test cases to be automated (if applicable).
 - Prioritize the test cases according to their importance and criticality.
 - Define areas of stress and performance testing.
 - Plan the test cycles required for regression testing

51

Software Testing Life Cycle (STLC)

- **Test Design** : One of the major activities in testing is the design of test cases.
- However, this activity is not an intuitional process; rather it is a well-planned process.
- The test design is an important phase after test planning.
- It includes the following critical activities.
 - Determining the test objectives and their prioritization.
 - Preparing list of items to be tested
 - Mapping items to test cases
 - Selection of test case design techniques
 - Creating test cases and test data
 - Setting up the test environment and supporting tools



52

Software Testing Life Cycle (STLC)

- **Test Execution** : In this phase, all test cases are executed including verification and validation.
- Verification test cases are started at the end of each phase of SDLC.
- Validation test cases are started after the completion of a module.
- It is the decision of the test team to opt for automation or manual execution.

Test Execution Level	Person Responsible
Unit	Developer of the module
Integration	Testers and Developers
System	Testers, Developers, End-users
Acceptance	Testers, End-users

Software Testing Life Cycle (STLC)

- **Post-Execution/Test Review** : As we know, after successful test execution, bugs will be reported to the concerned developers.
- This phase is to analyse bug-related issues and get feedback so that maximum number of bugs can be removed.
- This is the primary goal of all test activities done earlier.
- As soon as the developer gets the bug report, he performs the following activities:
 - Understanding the bug
 - Reproducing the bug
 - Analysing the nature and cause of the bug
- After this, the results from manual and automated testing can be collected.
- The final bug report and associated metrics are reviewed and analysed for overall testing process.

Testing Tactics

- **Software testing techniques:** effective testing is a real challenge.
- Design effective test cases with most of the testing domains covered detecting the maximum number of bugs.
- Actual methods for designing test cases, software testing technique, implement the test cases on the software.
- These techniques are categorized into two parts.
- **Static testing:** it is a technique for assessing the structural characteristics of source code, design specifications or any notational representation that conforms to well defined syntactical rules.
- **Dynamic Testing:** All the methods that execute the code to test a software are known as dynamic testing techniques.
- The code is run on a number of inputs provided by the user and the corresponding results are checked. Further divided into sub two categories: Black box testing and White box testing. ⁵⁵

Testing Tactics

- **Black box testing:** It checks only the functionality of system.
 - This techniques received inputs given to system and the output is received after processing in the system.
 - It is also known as functional testing. It is used for system testing under validation.
- **White box testing:** Every design feature and its corresponding code is checked logically with every possible path execution.
 - So it takes care of structural paths instead of just outputs.
 - It is known as structural testing and also used for unit testing under verification.

