

Module 1

Testing Methodology

Evolution of Software Testing

- In the early days of software development, Software Testing was considered only as a debugging process for removing the errors after the development of software.
- By 1970, software engineering term was in common use. But software testing was just a beginning at that time.
- In 1978, G.J. Myers realized the need to discuss the techniques of software testing in a separate subject. He wrote the book “The Art of Software Testing” which is a classic work on software testing.
- Myers discussed the psychology of testing and emphasized that **testing should be done with the mind-set of finding the errors not to demonstrate that errors are not present.**
- By 1980, software professionals and organizations started talking about the quality in software. Organizations started Quality assurance teams for the project, which take care of all the testing activities for the project right from the beginning.

Evolution of Software Testing

- In the 1990s testing tools finally came into their own. There was flood of various tools, which are absolutely vital to adequate testing of the software systems. However, they do not solve all the problems and cannot replace a testing process.
- Gelperin and Hetzel [79] have characterized the growth of software testing with time. Based on this, we can divide the evolution of software testing in following phases:

Evolution of Software Testing

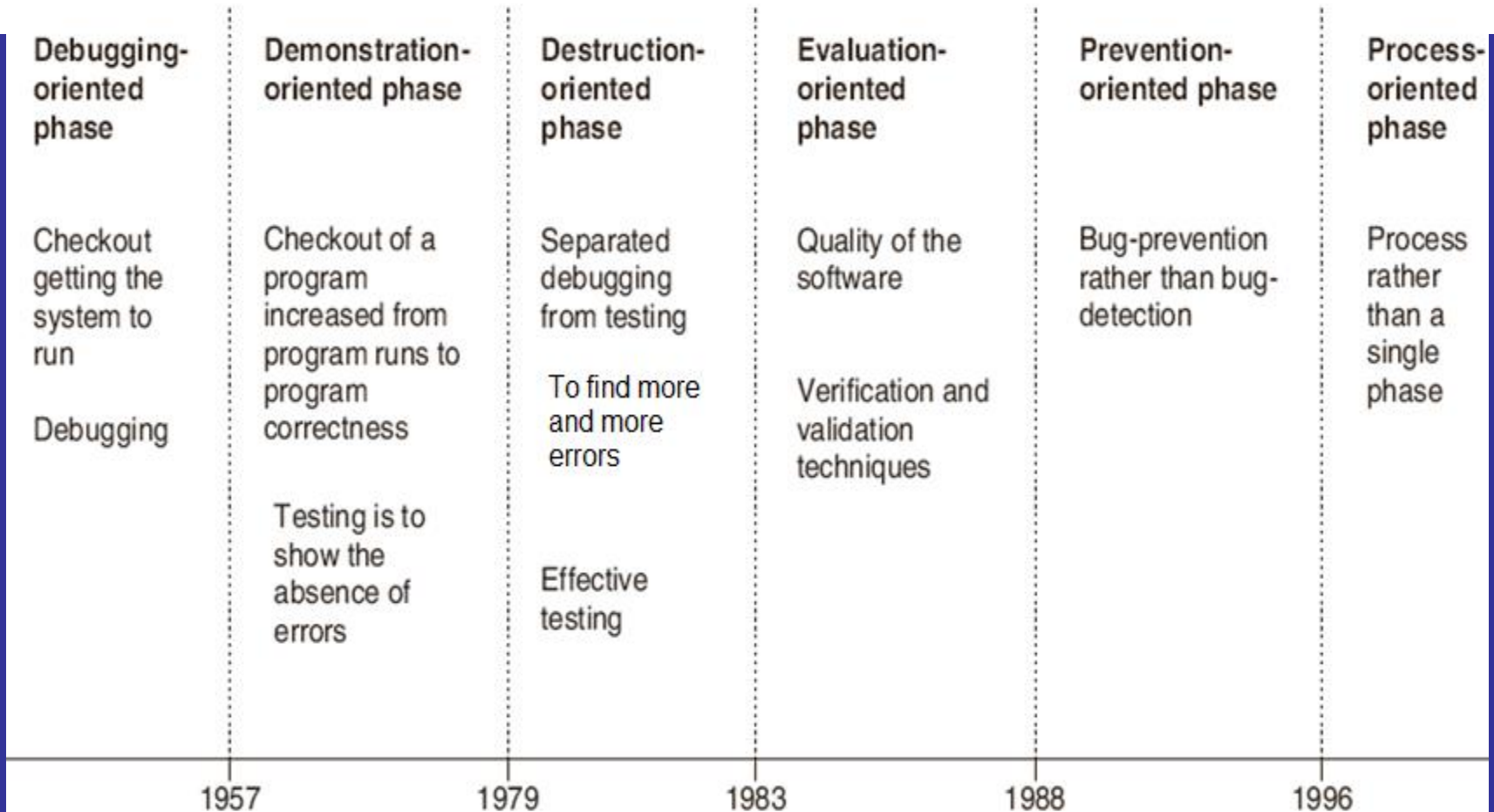
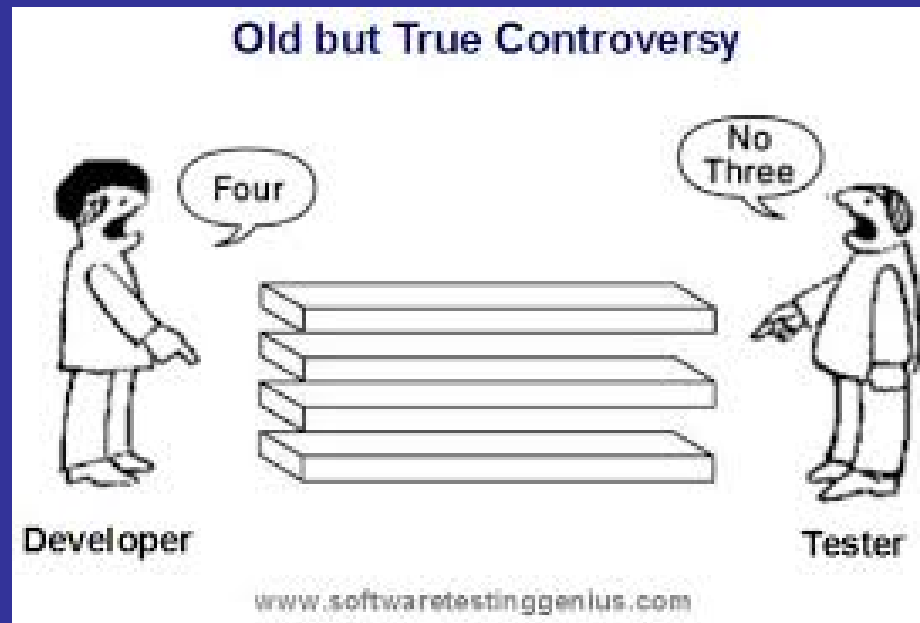


Figure 1.1 Evolution phases of software testing

Psychology for Software Testing:

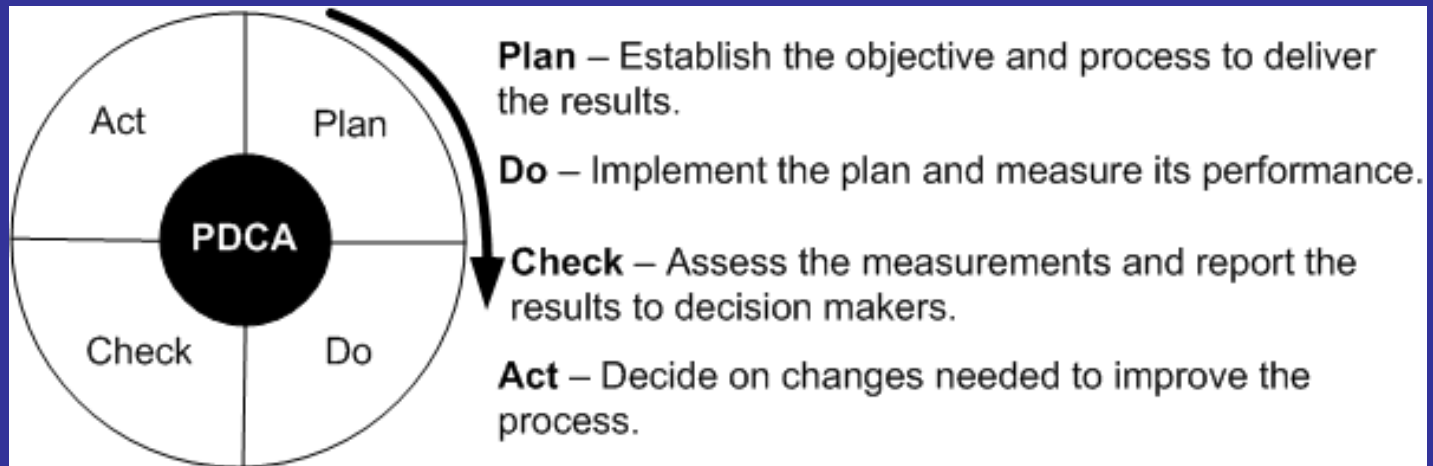
Testing is the process of demonstrating that there are no errors.

Testing is the process of executing a program with the intent of finding errors.



The Quality Revolution

The Shewhart cycle



- Deming introduced Shewhart's PDCA cycle to Japanese researchers
- It illustrates the activity sequence:
 - Setting goals
 - Assigning them to measurable milestones
 - Assessing the progress against the milestones
 - Take action to improve the process in the next cycle

Software Testing Goals

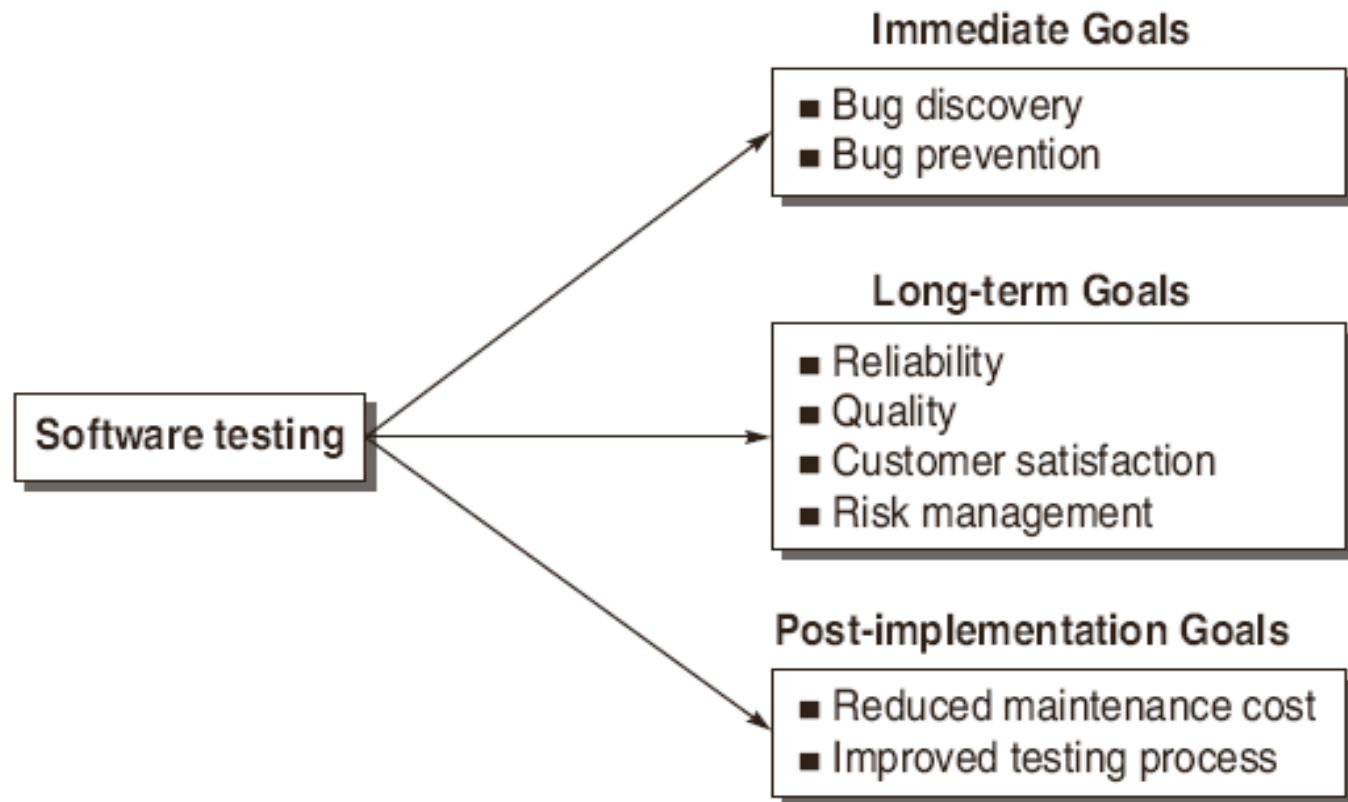
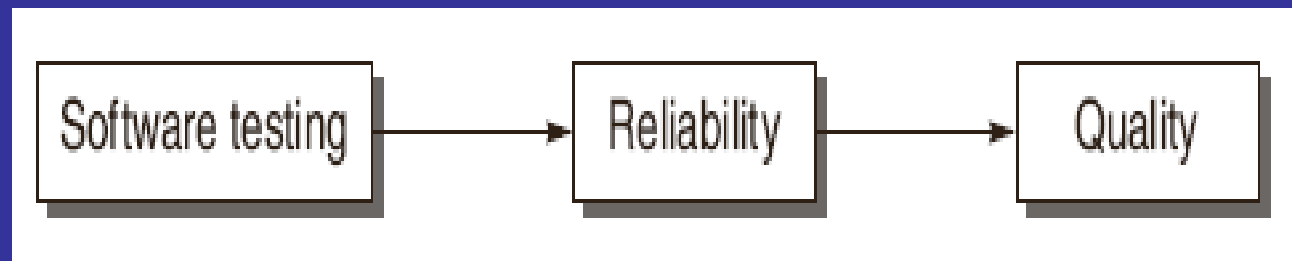
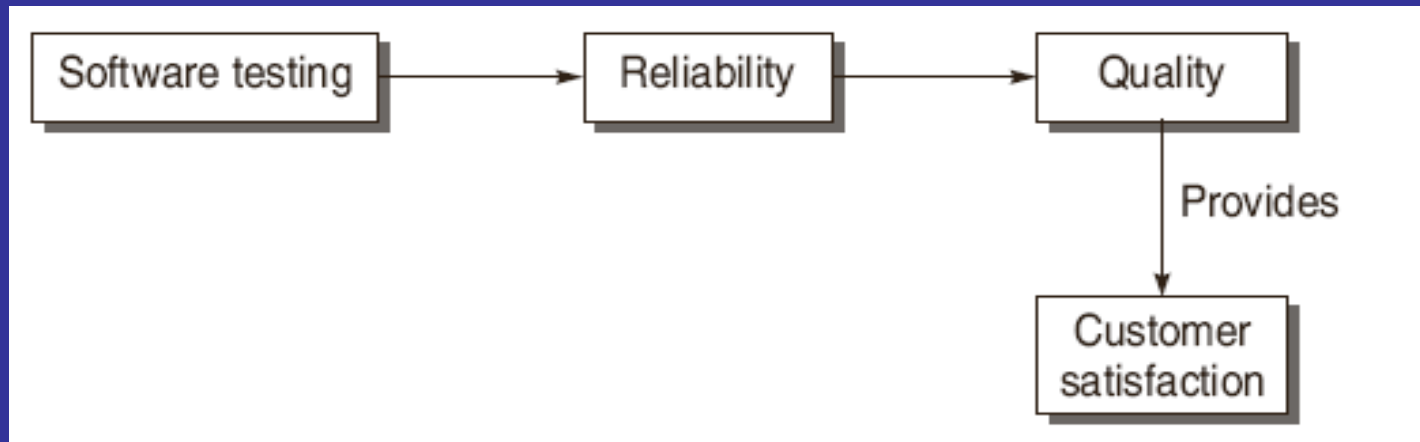


Figure 1.2 Software testing goals

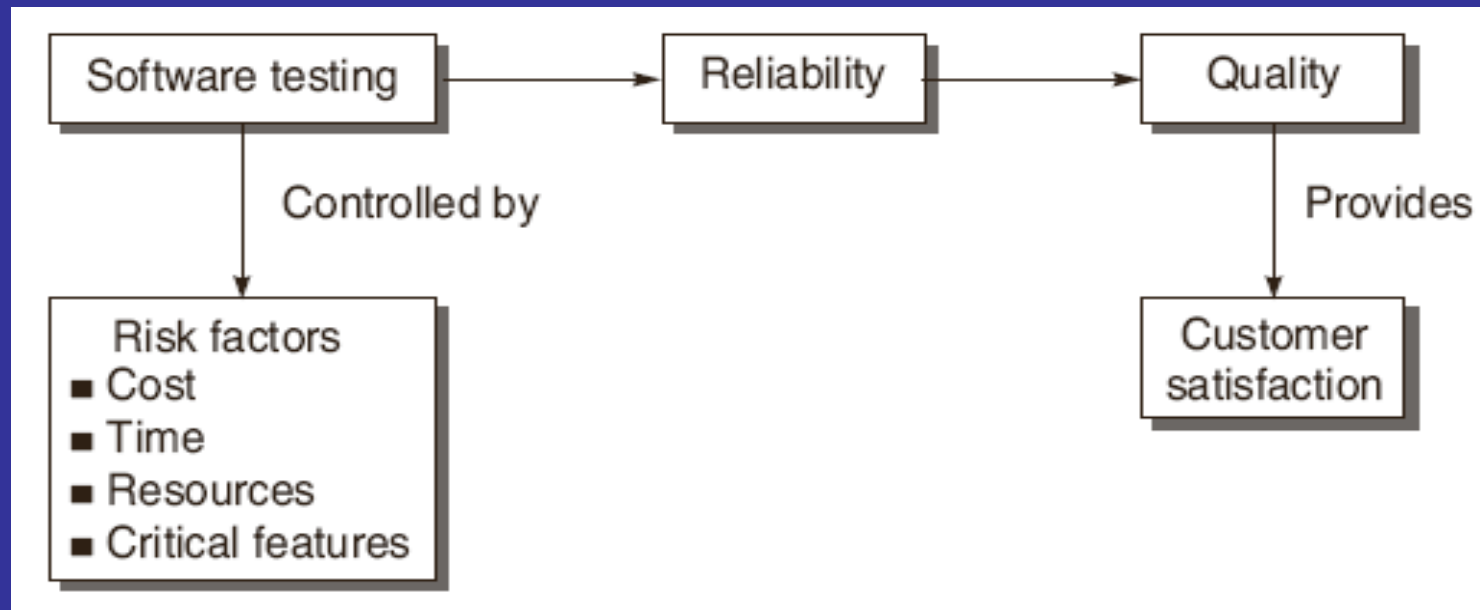
Testing produces Reliability and Quality



Quality leads to customer satisfaction



Testing control Risk factors



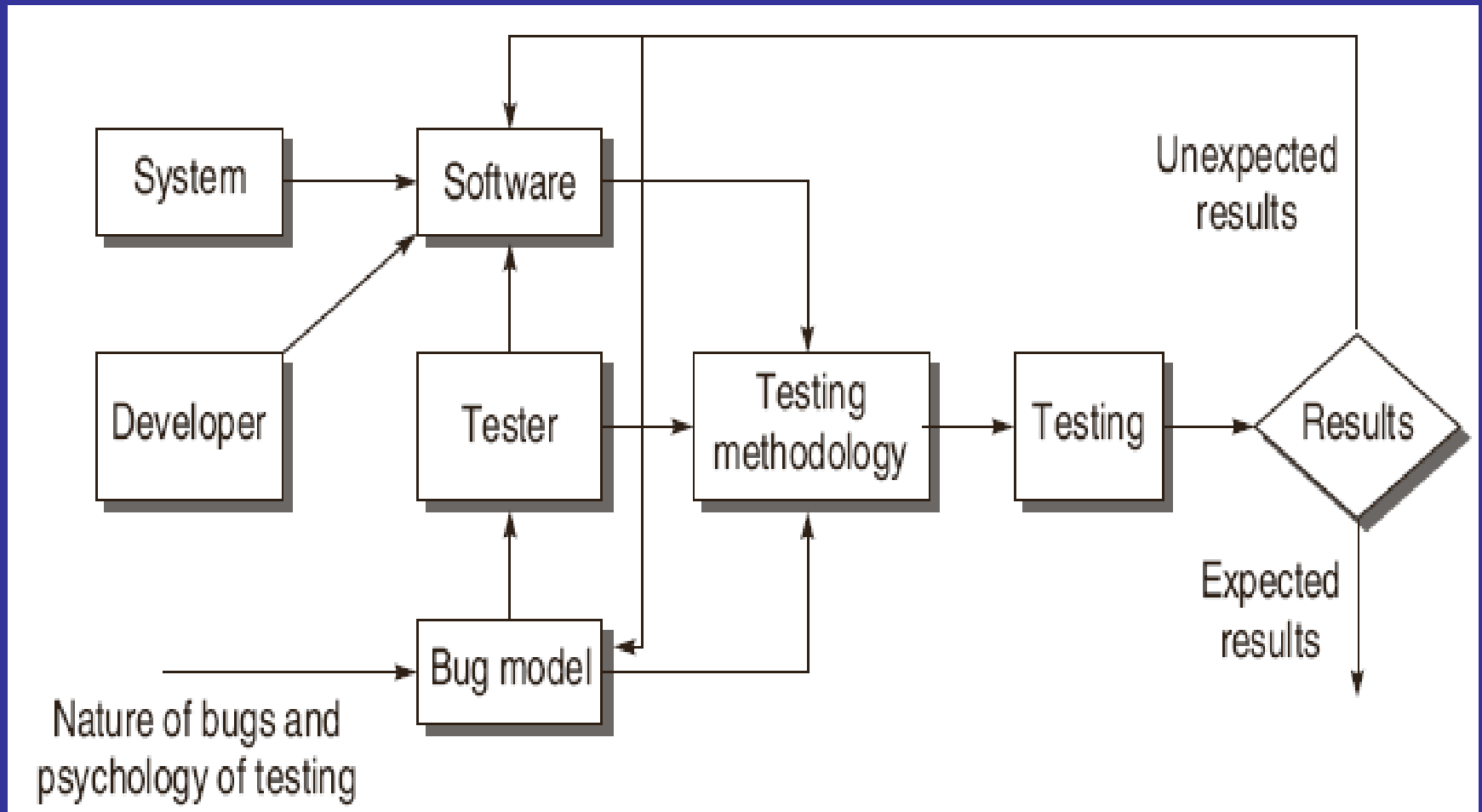
Software Testing Definitions

- *“Testing is the process of executing a program with the intent of finding errors.”*
- Myers [2]
- *“A successful test is one that uncovers an as-yet-undiscovered error.”*
- Myers [2]
- *“Testing can show the presence of bugs but never their absence.”*
- W. Dijkstra [125].

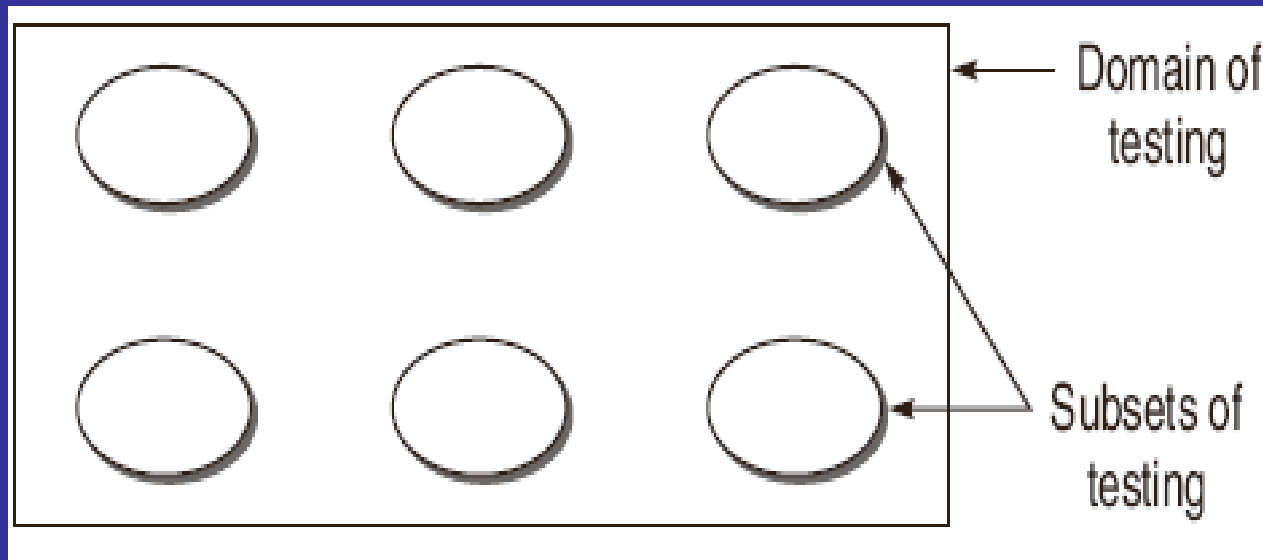
Software Testing Definitions

- *“Testing is a concurrent lifecycle process of engineering, using and maintaining testware (i.e. testing artifacts) in order to measure and improve the quality of the software being Tested.”*
- Craig [117]
- *“Software testing is a process that detects important bugs with the objective of having better quality software.”*

Model for Software Testing



Effective Software Testing vs Exhaustive Software Testing



Effective Software Testing vs Exhaustive Software Testing

The domain of possible inputs to the software is too large to test.

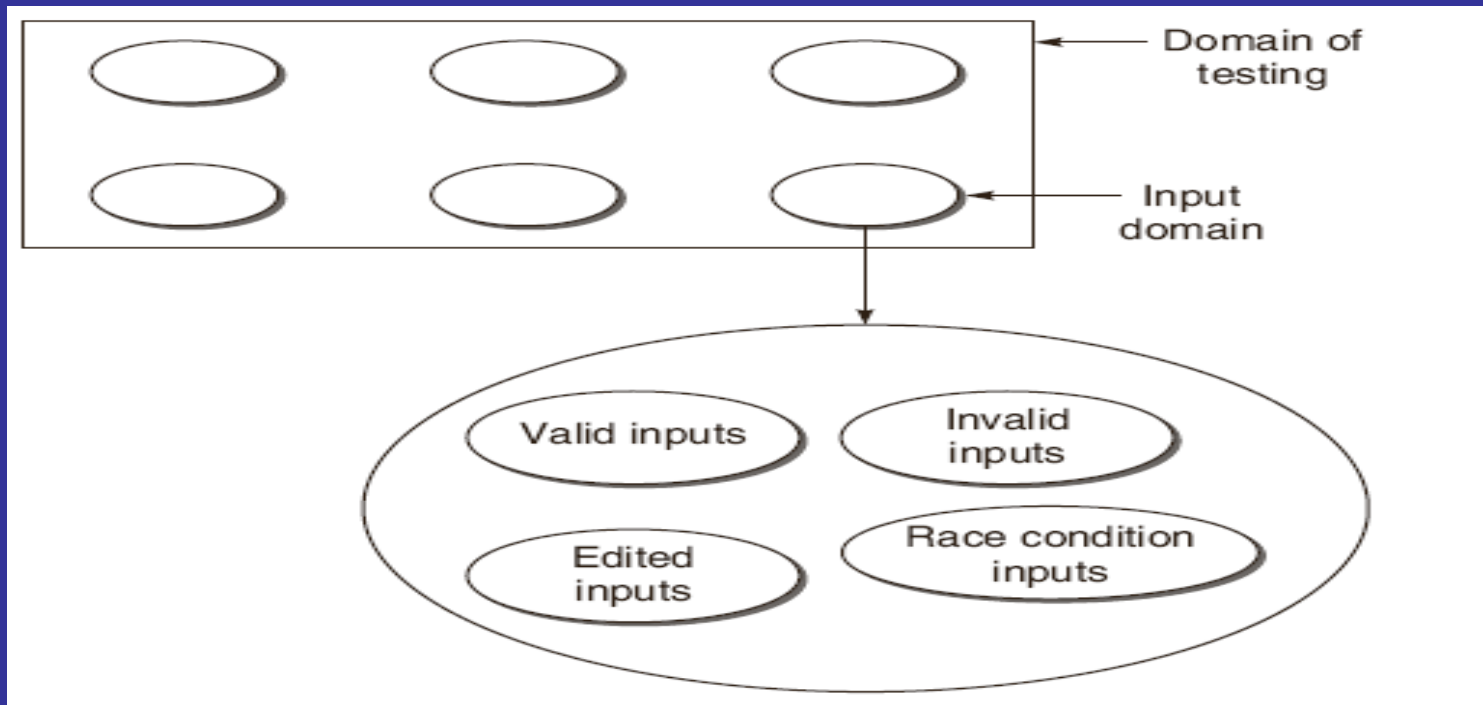
- **Valid Inputs**
- **Invalid Inputs**
- **Edited Inputs**
- **Race Conditions**

Effective Software Testing vs Exhaustive Software Testing

P2 - Exhaustive testing is impossible

- Testing everything is not feasible
- Instead of exhaustive testing, risk analysis and priorities should be used to focus testing efforts
- For example: In an application in one screen if there are 15 input fields, each having 5 possible values, then to test all the valid combinations you would need 30517578125 (5^{15}) tests. This is very unlikely that the project timescales would allow for this number of tests.

Effective Software Testing vs Exhaustive Software Testing

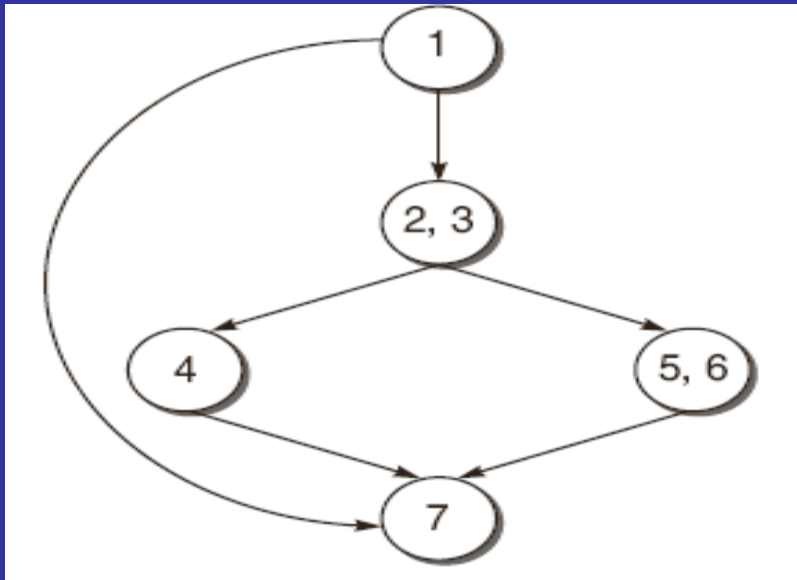


Effective Software Testing vs Exhaustive Software Testing

There are too many possible paths through the program to test.

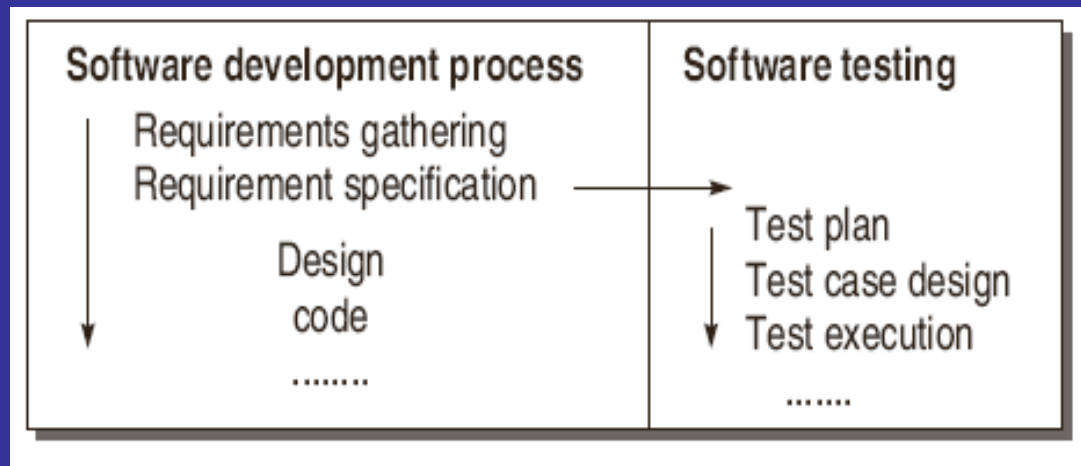
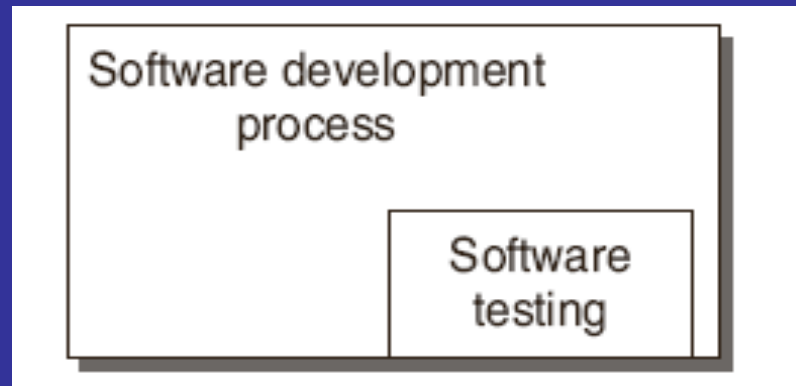
```
1for (int i = 0; i < n; ++i)
2{
3    if (m >= 0)
4        x[i] = x[i] + 10;
5    else
6        x[i] = x[i] - 2;
7}...
```

Effective Software Testing vs Exhaustive Software Testing



- Total number of paths will be $2^n + 1$, where n is the number of times the loop will be carried out.
- if n is 20, then the number of paths will be $2^{20} + 1$, i.e. 1048577.
- **Every Design error cannot be found.**

Software Testing as a Process



Software Testing as a Process

An organization for the better quality software must adopt a testing process and consider the following points:

- Testing process should be organized such that there is enough time for important and critical features of the software.
- Testing techniques should be adopted such that these techniques detect maximum bugs.
- Quality factors should be quantified so that there is clear understanding in running the testing process. In other words, process should be driven by the quantified quality goals. In this way, process can be monitored and measured.
- Testing procedures and steps must be defined and documented.
- There must be scope for continuous process improvement.

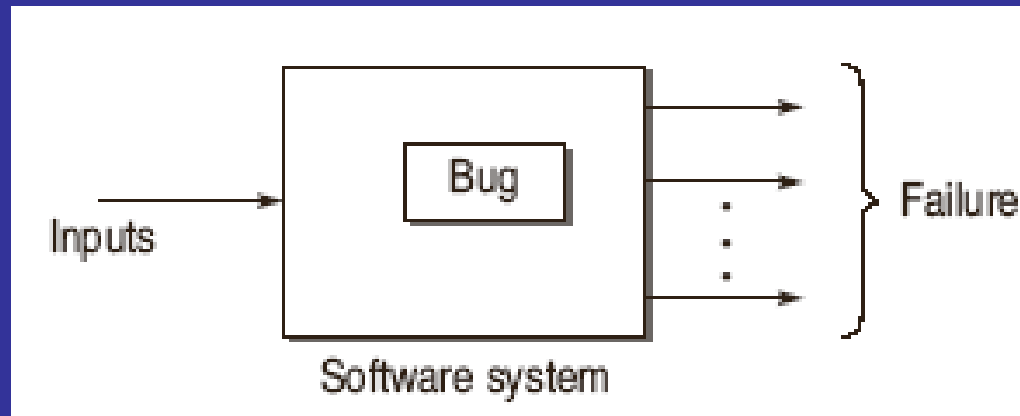
Software Testing Terminology

- **Failure**

The inability of a system or component to perform a required function according to its specification.

- **Fault / Defect / Bug**

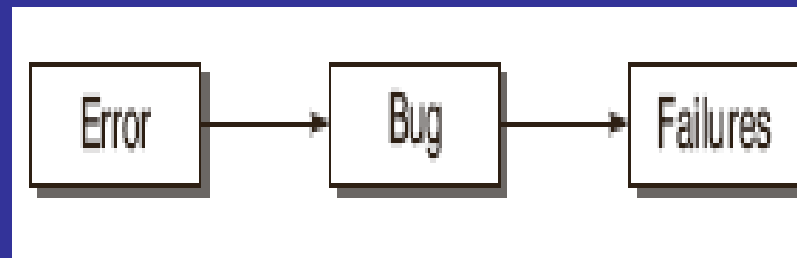
Fault is a condition that in actual causes a system to produce failure. It can be said that failures are manifestation of bugs.



Software Testing Terminology

Error

Whenever a member of development team makes any 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. Thus, error is a very general term used for human mistakes.

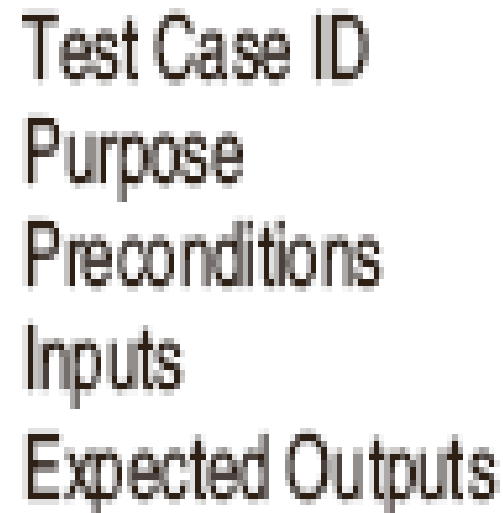


Software Testing Terminology

```
Module A()  
{  
  ---  
  While(a > n+1);  
  {  
    ---  
    print("The value of x is",x);  
  }  
  ---  
}
```


Software Testing Terminology

Test Case is a well – documented procedure designed to test the functionality of a feature in the system.



Test Case ID
Purpose
Preconditions
Inputs
Expected Outputs

Software Testing Terminology

- **Testware**

The documents created during the testing activities are known as Testware. (Test Plan, test specifications, test case design , test reports etc.)

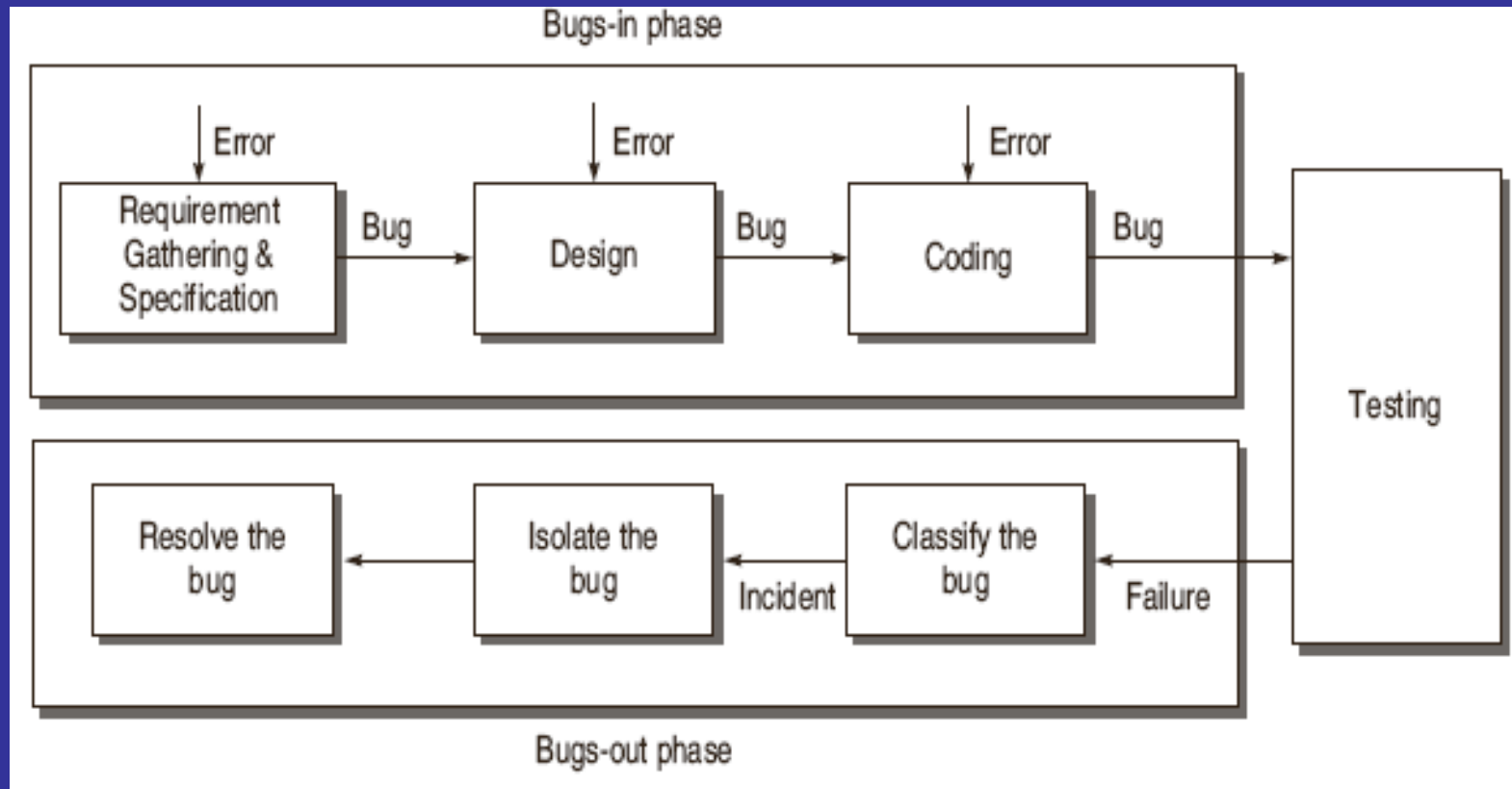
- **Incident**

The symptom(s) associated with a failure that alerts the user to the occurrence of a failure.

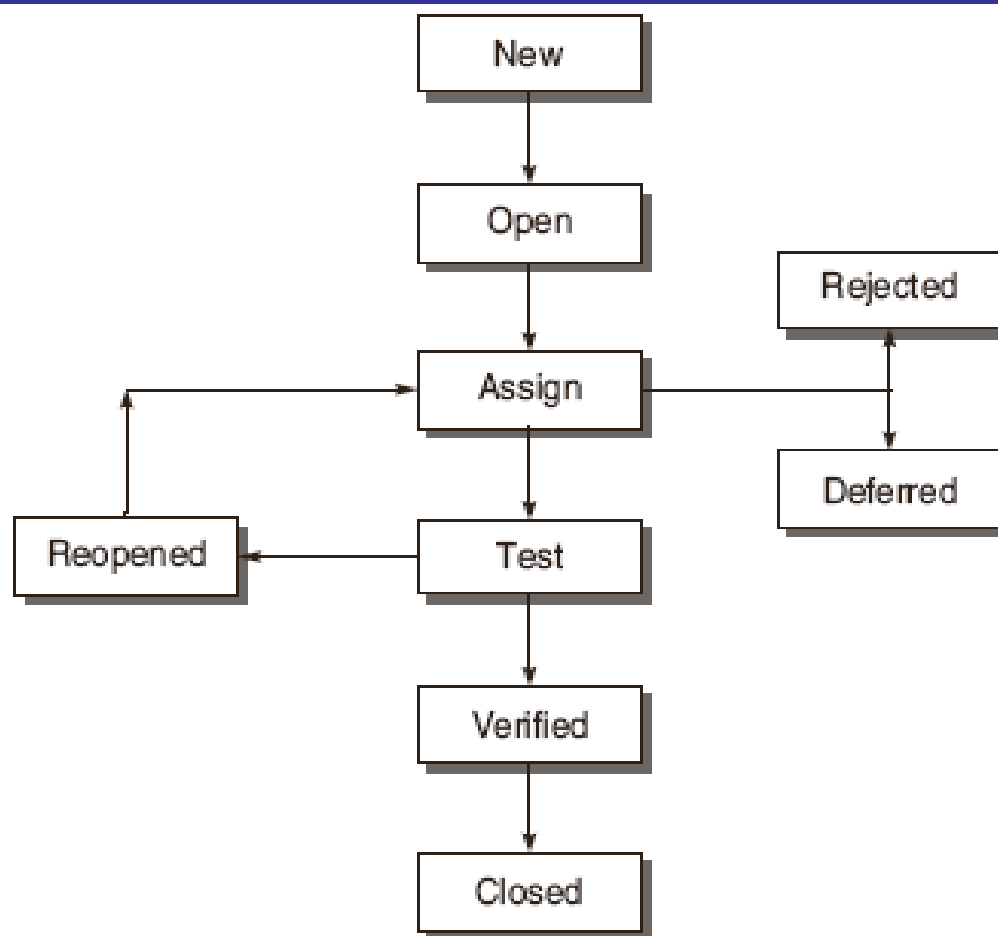
- **Test Oracle**

To judge the success or failure of a test(correctness of the system for some test) *Comparing actual results with expected results by hand.*

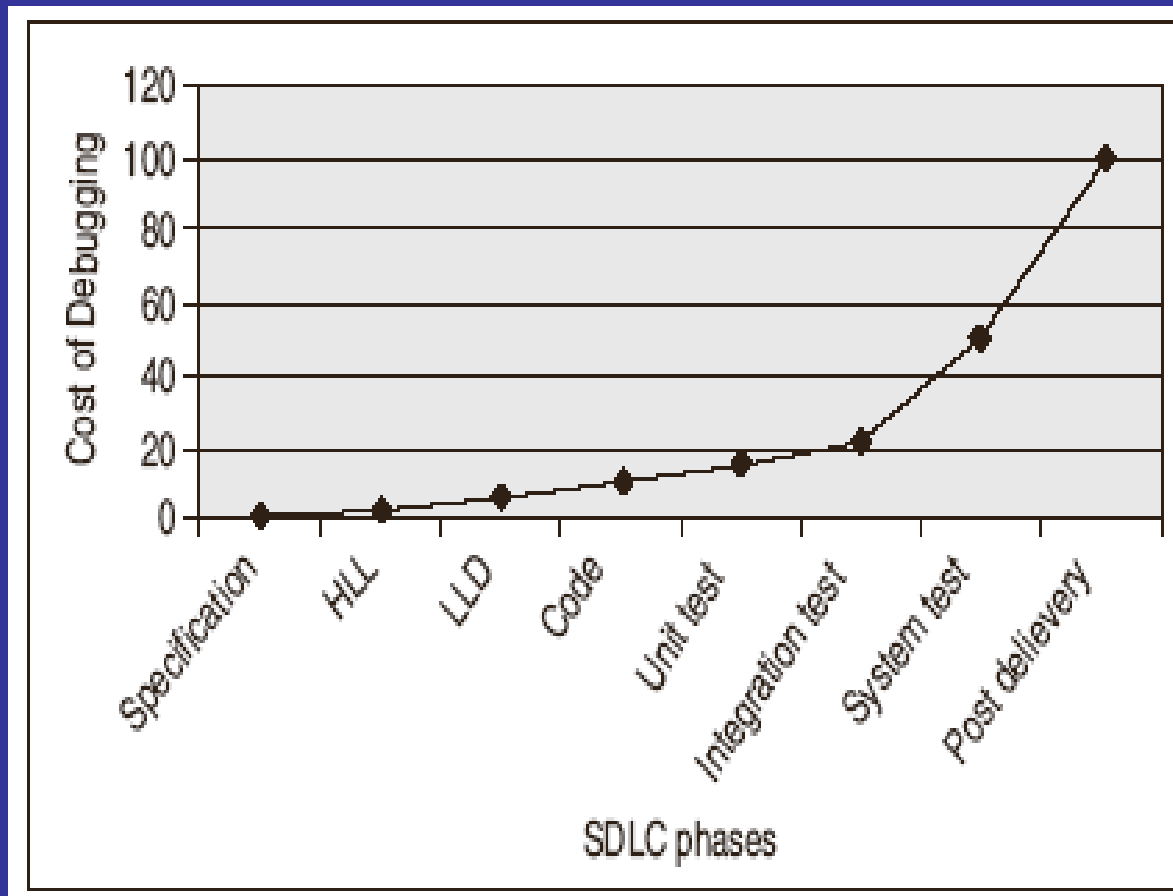
Life Cycle of a Bug



States of a Bug



Bug affects Economics of Software Testing



Bug Classification based on Criticality

- **Critical Bugs**
the worst effect on the functioning of software such that it stops or hangs the normal functioning of the software.
- **Major Bug**
This type of bug does not stop the functioning of the software but it causes a functionality to fail to meet its requirements as expected.
- **Medium Bugs**
Medium bugs are less critical in nature as compared to critical and major bugs.(not according to standards- Redundant /Truncated output)
- **Minor Bugs**
This type of bug does not affect the functioning of the software.(Typographical error or misaligned printout)

Bug Classification based on SDLC

Requirements and Specifications Bugs

Design Bugs

Control Flow Bugs

Logic Bugs: Improper layout of cases, missing cases

Processing Bugs: Arithmetic errors, Incorrect data conversion

Data Flow Bugs

Error Handling Bugs

Race Condition Bugs

Boundary Related Bugs

User Interface Bugs

Coding Bugs

Interface and Integration Bugs

System Bugs

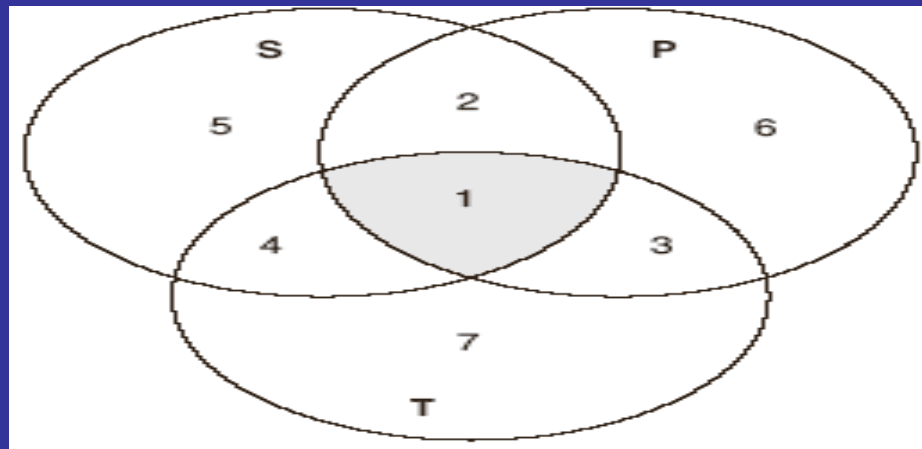
Testing Bugs

Testing Principles

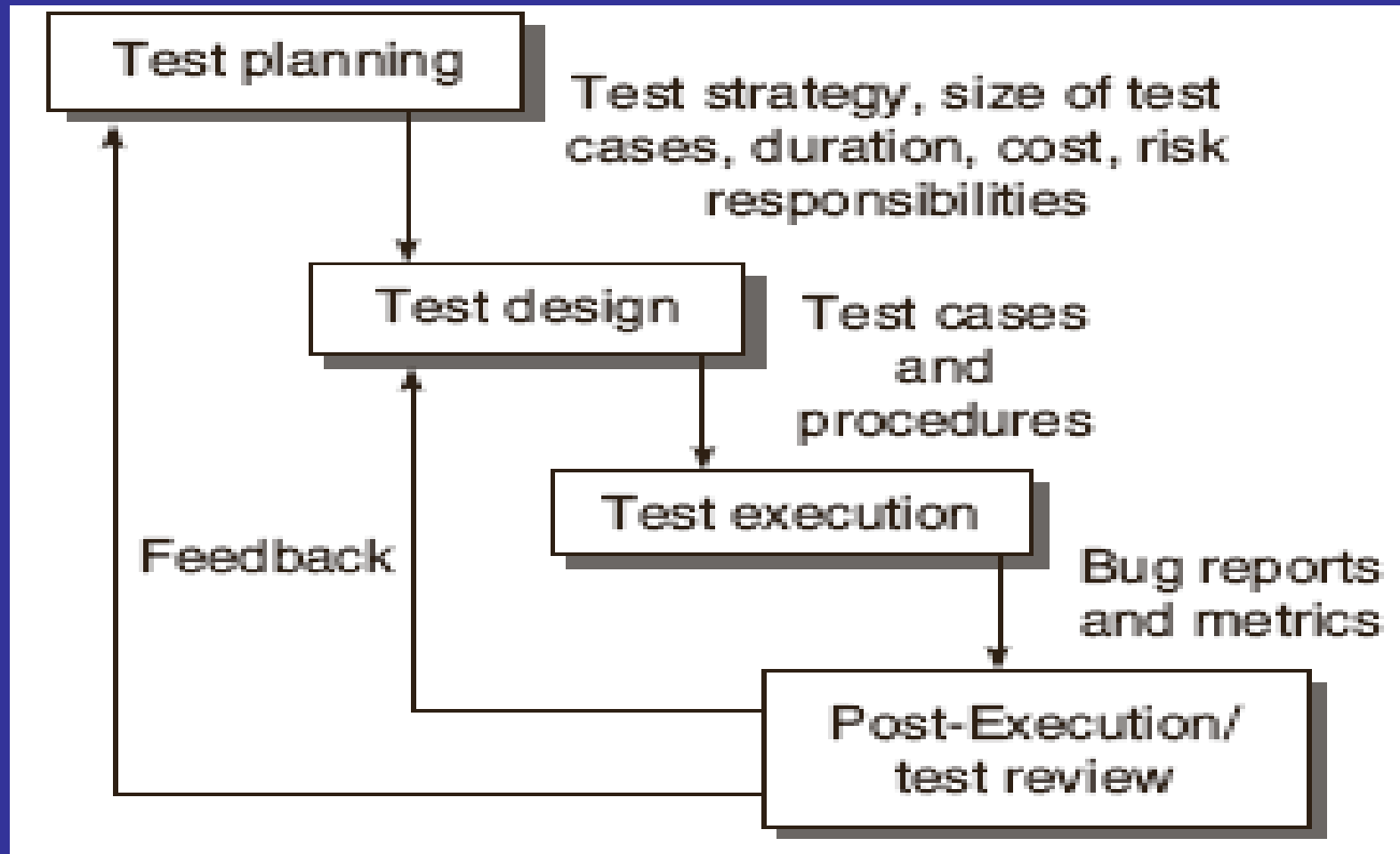
- **Effective Testing not Exhaustive Testing**
- **Testing is not a single phase performed in SDLC**
- **Destructive approach for constructive testing**
- **Early Testing is the best policy.**
- **The probability of the existence of an error in a section of a program is proportional to the number of errors already found in that section.**
- **Testing strategy should start at the smallest module level and expand toward the whole program.**

Testing Principles

- Testing should also be performed by an independent team.
- Everything must be recorded in software testing.
- Invalid inputs and unexpected behavior have a high probability of finding an error.
- Testers must participate in specification and design reviews.



Software Testing Life Cycle (STLC): Well defined series of steps to ensure successful and effective testing.



Software Testing Life Cycle (STLC): Well defined series of steps to ensure successful and effective testing.

- 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.

Test Planning

- Defining the Test Strategy
- Estimate of 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, project metrics

Test Planning

The major output of test planning is the test plan document. Test plans are developed for each level of testing. After analysing the issues, the following activities are performed:

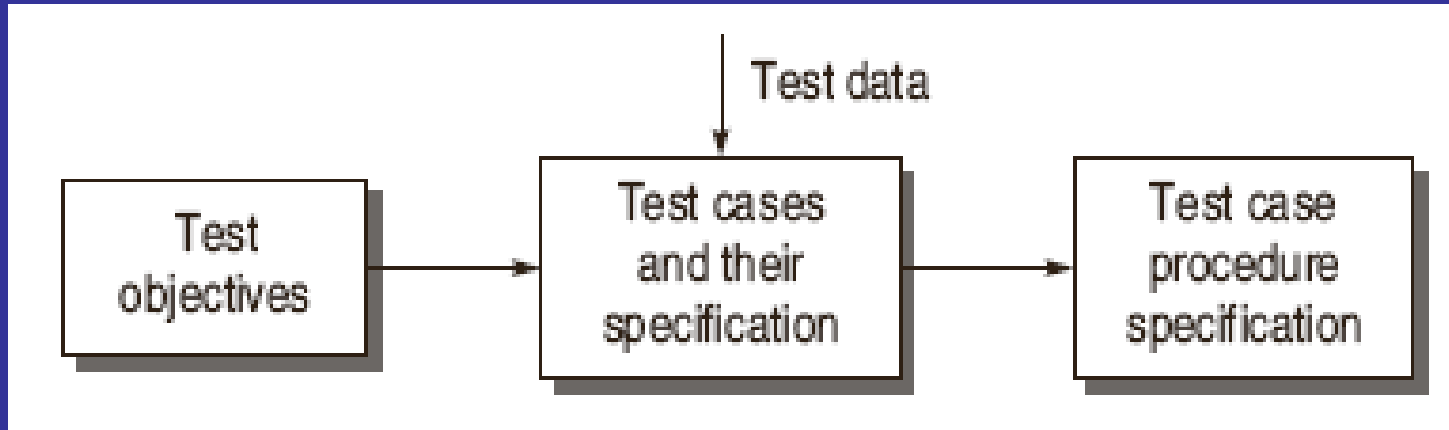
- Develop a test case format.
- Develop test case plans according to every phase of SDLC.
- Identify test cases to be automated.
- Prioritize the test cases according to their importance and criticality.
- Define areas of stress and performance testing.
- Plan the test cycles required for regression testing.

Test Plan: A document describing the scope, approach, resources and schedule of intended test activities as a roadmap.

Test Design

- Determining the test objectives and their Prioritization(broad categories of things to test)
- Preparing List of Items to be Tested under each objective
- Mapping items to test cases
- Selection of Test case design techniques(black box and white box)
- Creating Test Cases and Test Data
- Setting up the test environment and supporting tools
- Creating Test Procedure Specification(description of how the test case will be run). It is in the form of sequenced steps. This procedure is actually used by the tester at the time of execution of test cases.

Test Design

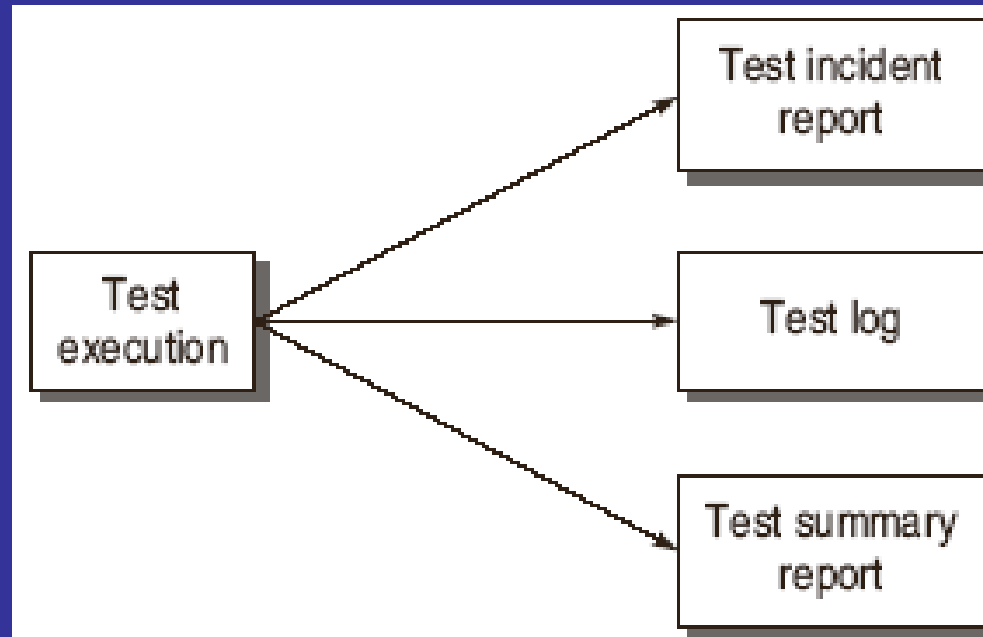


All the details specified in the test design phase are documented in the test design specification. This document provides the details of the input specifications, output specifications, environmental needs, and other procedural requirements for the test case.

Test Execution: Verification and Validation

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 results are documented in the test incident reports, test logs, testing status, and test summary reports etc.



Post-Execution / Test Review

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.

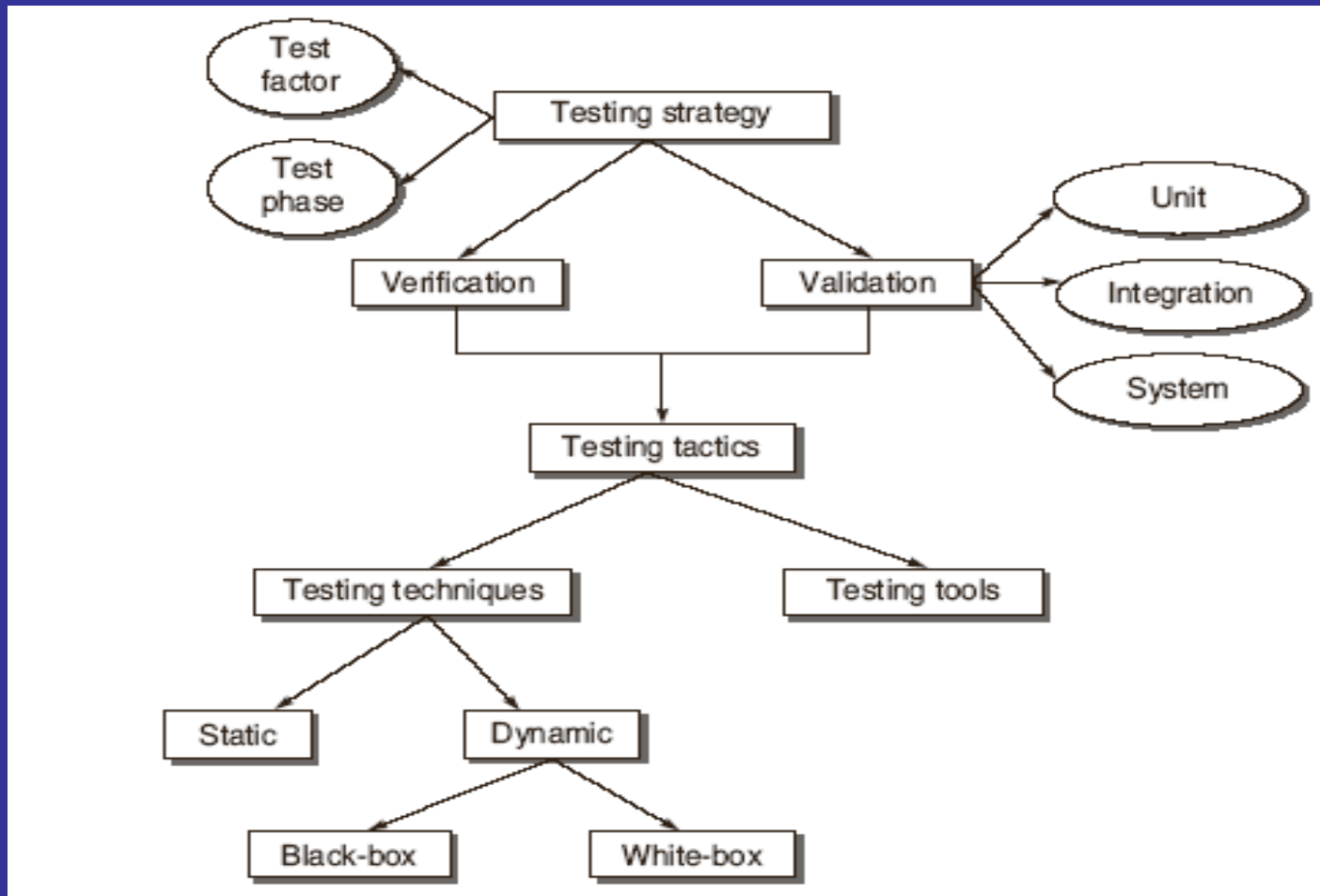
As soon as developer gets the bug report, he perform the following activities:

- Understanding the Bug
- Reproducing the bug
- Analyzing the nature and cause of the bug

Review Process:

- *Reliability analysis*
- *Coverage analysis*
- *Overall defect analysis* (Quality Improvement)

Software Testing Methodology is the organization of software testing by means of which the test strategy and test tactics are achieved.



Test Strategy

Planning of the whole testing process into a well-planned series of steps. Test strategy provides a roadmap that includes very specific activities that must be performed by the test team in order to achieve a specific goal.

- Test Factors Test factors are risk factors or issues related to the system under development. Risk factors need to be selected and ranked according to a specific system under development.
- Test Phase This is another component on which the testing strategy is based. It refers to the phases of SDLC where testing will be performed. Testing strategy may be different for different models of SDLC, e.g. strategies will be different for waterfall and spiral models.

Test Strategy Matrix: A test strategy matrix identifies the concerns that will become the focus of test planning and execution.

In this way, this matrix becomes an input to develop the testing strategy.

- **Select and Rank Test Factors**
- **Identify the System Development Phases**
- **Identify the Risks(concerns) associated with System under Development**
- **Plan the test strategy for every risk identified.**

Let's take a project as an example.
Suppose a new operating system has to be designed, which needs a test strategy.

Test Factors	Test Phase					
	Requirements	Design	Code	Unit test	Integration test	System test
Portability	Is portability feature mentioned in specifications according to different hardware?					Is system testing performed on MIPS and INTEL platforms?
Service Level	Is time frame for booting mentioned?	Is time frame incorporated in design of the module?				

Reference: Software Testing Principles and Practices, Nareish Chaudhary, Oxford University

Risks and test objectives - examples

Risk	Test Objective
The web site fails to function correctly on the user's client operating system and browser configuration.	To demonstrate that the application functions correctly on selected combinations of operating systems and browser version combinations.
Bank statement details presented in the client browser do not match records in the back-end legacy banking systems.	To demonstrate that statement details presented in the client browser reconcile with back-end legacy systems.
Vulnerabilities that hackers could exploit exist in the web site networking infrastructure.	To demonstrate through audit, scanning and ethical hacking that there are no security vulnerabilities in the web site networking infrastructure.

Load Testing Objectives

- "To have a response time under XX seconds"
- "Error rate is under XX%"
- "The infrastructure can handle XXX users"

Development of Test Strategy

The testing strategy should start at the component level and finish at the integration of the entire system. Thus, a test strategy includes testing the components being built for the system, and slowly shifts towards testing the whole system. This gives rise to two basic terms—Verification and Validation—the basis for any type of testing. It can also be said that the testing process is a combination of verification and validation.

Software Verification : The process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.

Software Validation : The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements . (in conformance with customer expectation)

Verification: “Are we building the product right?”

Validation: “Are we building the right product?”

Verification would check the design doc and correcting the spelling mistake.
consider the following specification

A clickable button with name Submet

- Otherwise, the development team will create a button like



Submet

- So new specification is

A clickable button with name Submit

- Once the code is ready, Validation is done. A Validation test found –

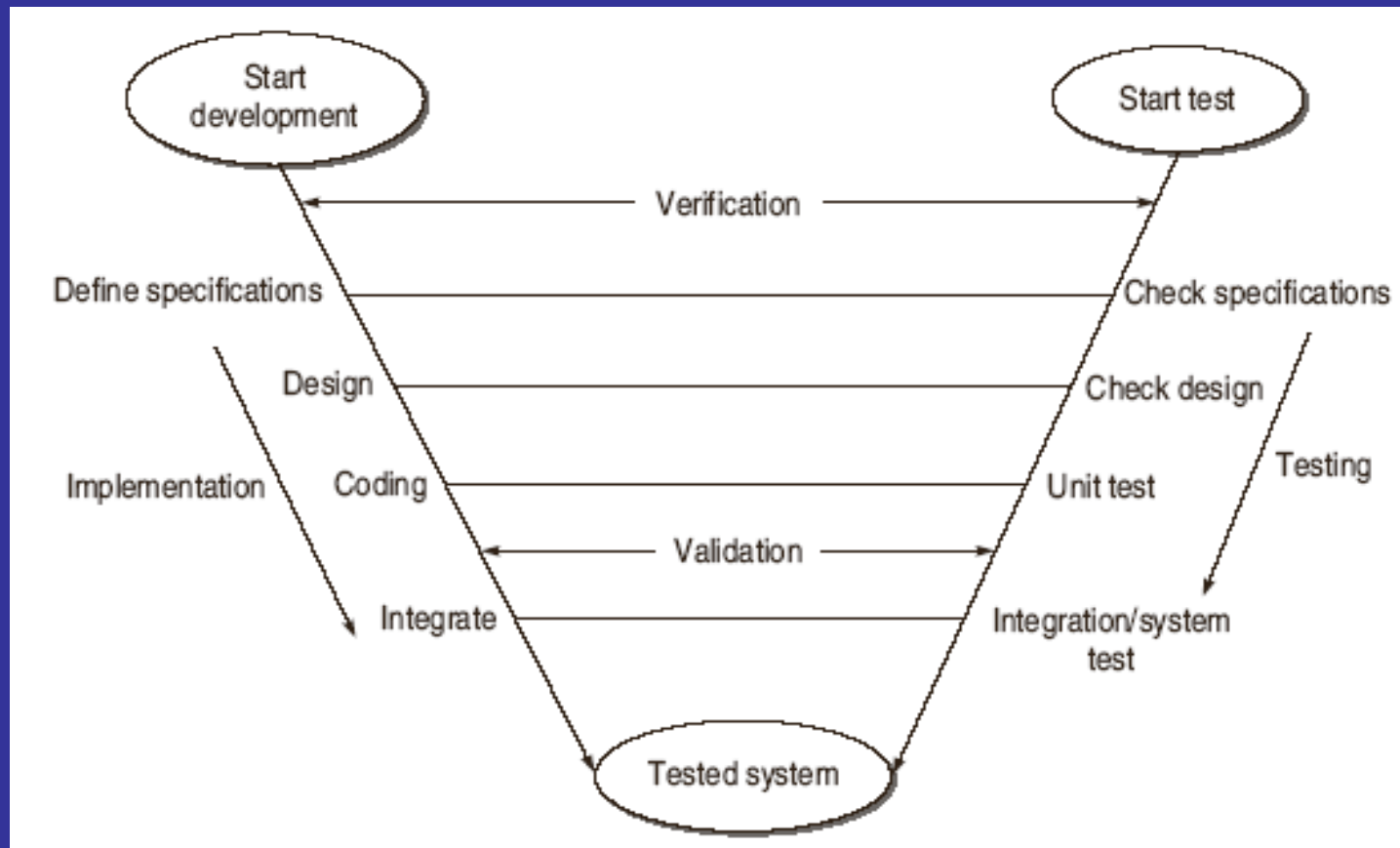
Button **NOT** Clickable



Submit

Owing to Validation testing, the development team will
make the submit button clickable

V Testing Life Cycle Model



Validation Activities

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing

Testing Tactics

The ways to perform various types of testing under a specific test strategy.

#Manual Testing

#Automated Testing

Software Testing techniques: Methods for design test cases.

#Static Testing

#Dynamic Testing

- * White-Box Testing

- * Black-Box Testing

Testing Tools :

Resource for performing a test process

Considerations in Developing Testing Methodologies

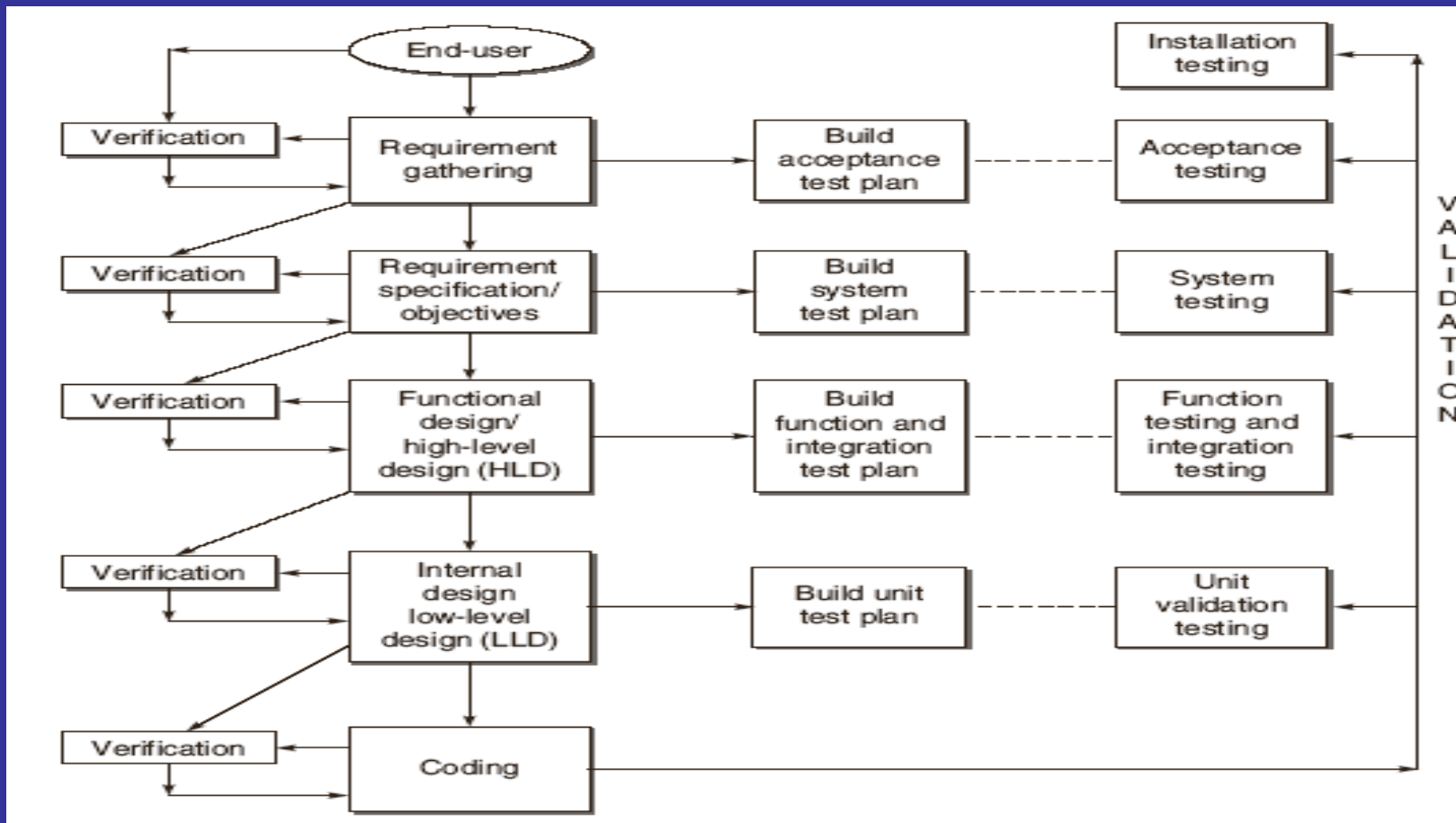
Determine project risks

Determine the type of development project

Identify test activities according to SDLC phase

Build test plan

Verification and Validation (V & V) Activities



VERIFICATION

Verification is a set of activities that ensures correct implementation of specific functions in a software.

Verification is to check whether the software conforms to specifications.

- If verification is not performed at early stages, there are always a chance of mismatch between the required product and the delivered product.
- Verification exposes more errors.
- Early verification decreases the cost of fixing bugs.
- Early verification enhances the quality of software.

VERIFICATION ACTIVITIES :All the verification activities are performed in connection with the different phases of SDLC. The following verification activities have been identified:

- Verification of Requirements and Objectives
- Verification of High-Level Design
- Verification of Low-Level Design
- Verification of Coding (Unit Verification)

Verification of Requirements

1. Verification of acceptance criteria

(An acceptance criterion defines the goals and requirements of the proposed system and acceptance limits for each of the goals and requirements.)

2. Acceptance Test plan

1. Verification of Objectives/Specifications(SRS): The purpose of this verification is to ensure that the user's needs are properly understood before proceeding with the project.

2. System Test plan

In verifying the requirements and objectives, the tester must consider both functional and non-functional requirements.

- Correctness
- Unambiguous(Every requirement has only one interpretation.)
- Consistent(No specification should contradict or conflict with another.)
- Completeness
- Updation
- Traceability
 - Backward Traceability
 - Forward Traceability.

Verification of High Level Design

1. The tester verifies the high-level design.
2. The tester also prepares a Function Test Plan which is based on the SRS. This plan will be referenced at the time of Function Testing .
3. The tester also prepares an Integration Test Plan which will be referred at the time of integration testing.
4. The tester verifies that all the components and their interfaces are in tune with requirements of the user. Every requirement in SRS should map the design.

Verification of High Level Design

Data Design: It creates a model of data and/or information that is represented at a high level of abstraction (the customer/user's view of data).

Verification of Data Design

- Check whether sizes of data structure have been estimated appropriately.
- Check the provisions of overflow in a data structure.
- Check the consistency of data formats with the requirements.
- Check whether data usage is consistent with its declaration.
- Check the relationships among data objects in data dictionary.
- Check the consistency of databases and data warehouses with requirements in SRS.

Verification of High Level Design

Architectural Design: It focuses on the representation of the structure of software components, their properties, and interactions.

Verification of Architectural Design

- Check that every functional requirement in SRS has been taken care in this design.
- Check whether all exceptions handling conditions have been taken care.
- Verify the process of transform mapping and transaction mapping used for transition from the requirement model to architectural design.
- check the functionality of each module according to the requirements specified.
- Check the inter-dependence and interface between the modules.
 - Coupling and Module Cohesion.

Verification of High Level Design

Interface Design: It creates an effective communication medium between the interfaces of different software modules, interfaces between the software system and any other external entity, and interfaces between a user and the software system.

Verification of Interface Design

- Check all the interfaces between modules according to architecture design.
- Check all the interfaces between software and other non-human producer and consumer of information.
- Check all the interfaces between human and computer.
- Check all the above interfaces for their consistency.
- Check that the response time for all the interfaces are within required ranges.
- Help Facility
- Error messages and warnings
- Check mapping between every menu option and their corresponding commands.

Verify Low Level Design

1. The tester verifies the LLD. The details and logic of each module is verified such that the high-level and low-level abstractions are consistent.

2. The tester also prepares the Unit Test Plan which will be referred at the time of Unit Testing

- 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.
- Traceability matrix between SRS and SDD.

How to Verify Code

Coding is the process of converting LLD specifications into a specific language. This is the last phase when we get the operational software with the source 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

How to Verify Code

Two kinds of techniques are used to verify the coding:
(a) static testing, and (b) dynamic testing.

Static testing techniques :

This technique does not involve actual execution. It considers only static analysis of the code or some form of conceptual execution of the code.

Dynamic testing techniques:

It is complementary to the static testing technique. It executes the code on some test data. The developer is the key person in this process who can verify the code of his module by using the dynamic testing technique.

How to Verify Code

UNIT VERIFICATION :

Verification of coding cannot be done for the whole system. Moreover, the system is divided into modules. Therefore, verification of coding means the verification of code of modules by their developers. This is also known as unit verification testing.

Listed below are the points to be considered while performing unit verification :

- Interfaces are verified to ensure that information properly flows in and out of the program unit under test.
- The local data structure is verified to maintain data integrity.
- Boundary conditions are checked to verify that the module is working fine on boundaries also.
- All independent paths through the control structure are exercised to ensure that all statements in a module have been executed at least once.
- All error handling paths are tested.

Unit verification is largely white-box oriented

Validation

- Validation is the next step after verification.
- Validation is performed largely by black box testing techniques.
- Developing tests that will determine whether the product satisfies the users' requirements, as stated in the requirement specification.
- Developing tests that will determine whether the product's actual behavior matches the desired behavior, as described in the functional design specification.
- The bugs, which are still existing in the software after coding need to be uncovered.
- last chance to discover the bugs otherwise these bugs will move to the final product released to the customer.
- Validation enhances the quality of software.

Validation Activities

Validation Test Plan

- **Acceptance Test Plan**
- **System Test Plan**
- **Function Test Plan**
- **Integration Test Plan**
- **Unit Test Plan**
-

Validation Test Execution

- **Unit Validation Testing**
- **Integration Testing**
- **Function Testing**
- **System Testing**
- **Acceptance Testing**
- **Installation Testing**

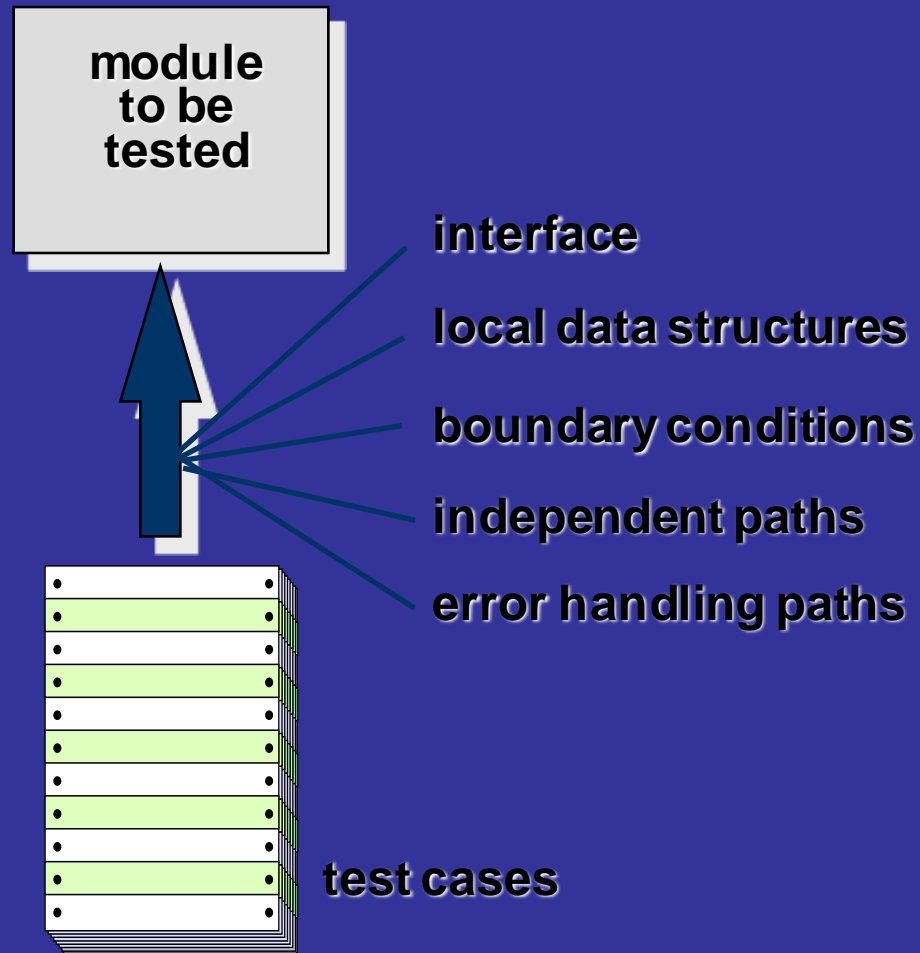
Concept of Unit Testing

- Unit is?
 - Function
 - Procedure
 - Method
 - Module
 - Component

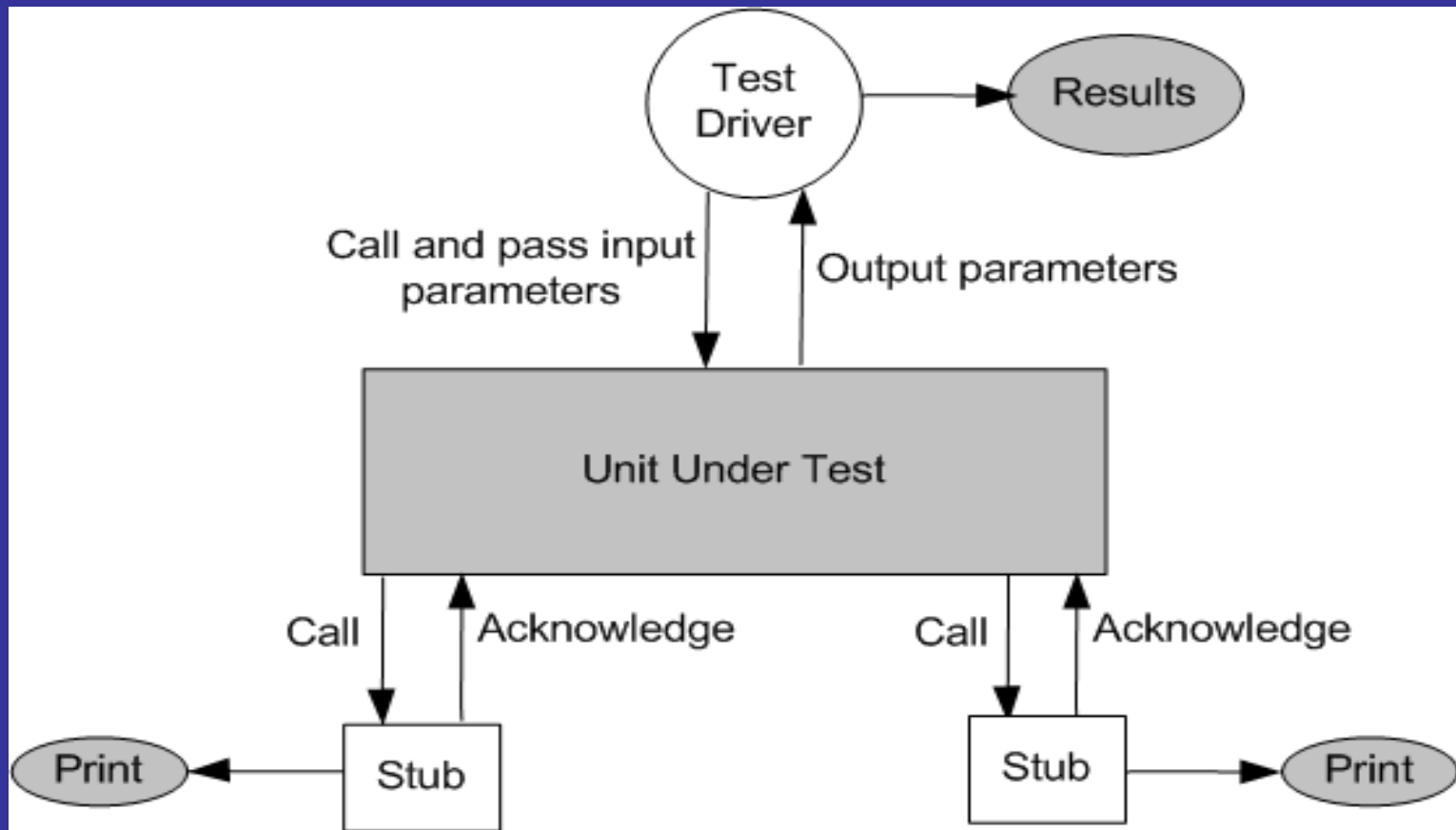
Unit Testing

- Testing program unit in isolation i.e. in a stand alone manner.
- Objective: Unit works as expected.

Unit Testing



Unit Testing



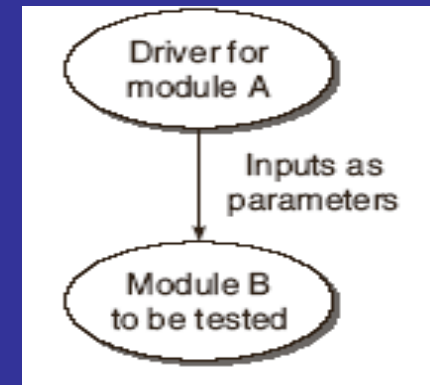
Dynamic unit test environment

Unit Testing

- The environment of a unit is emulated and tested in isolation
- The caller unit is known as *test driver*
 - A *test driver* is a program that invokes the unit under test (UUT)
 - It provides input data to unit under test and report the test result
- The emulation of the units called by the UUT are called *stubs*
 - It is a dummy program
- The *test driver* and the *stubs* are together called *scaffolding*
- The low-level design document provides guidance for selection of input test data

Unit Validation Testing

- **Drivers**

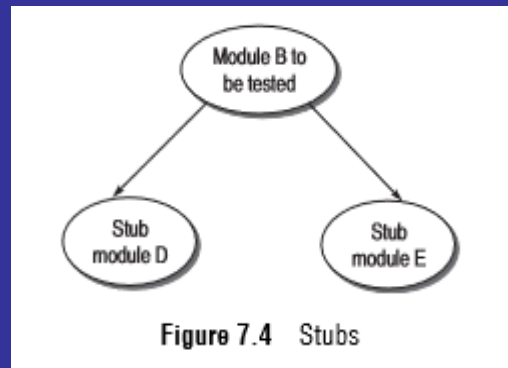


- **A test driver is supporting code and data used to provide an environment for testing part of a system in isolation.**
- **A test driver may take inputs in the following form and call the unit to be tested:**
 - It may hardcode the inputs as parameters of the calling unit.
 - It may take the inputs from the user.
 - It may read the inputs from a file.

Unit Validation Testing

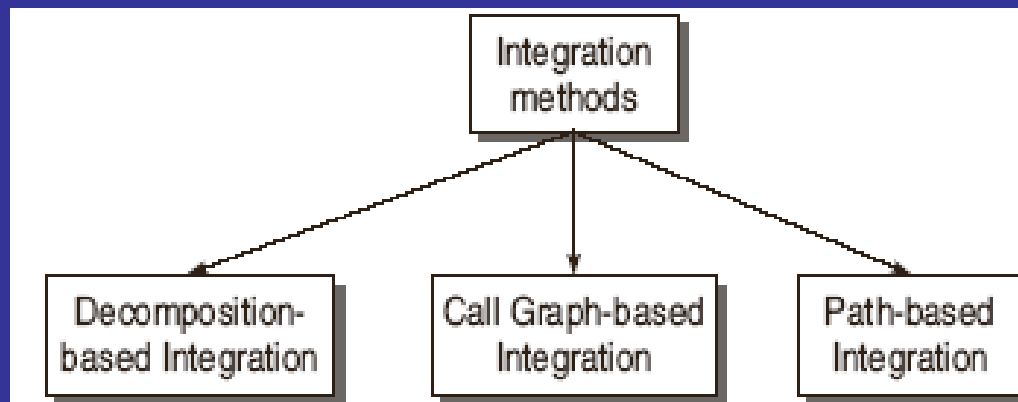
Stubs

Stub can be defined as a piece of software that works similar to a unit which is referenced by the Unit being tested, but it is much simpler than the actual unit.

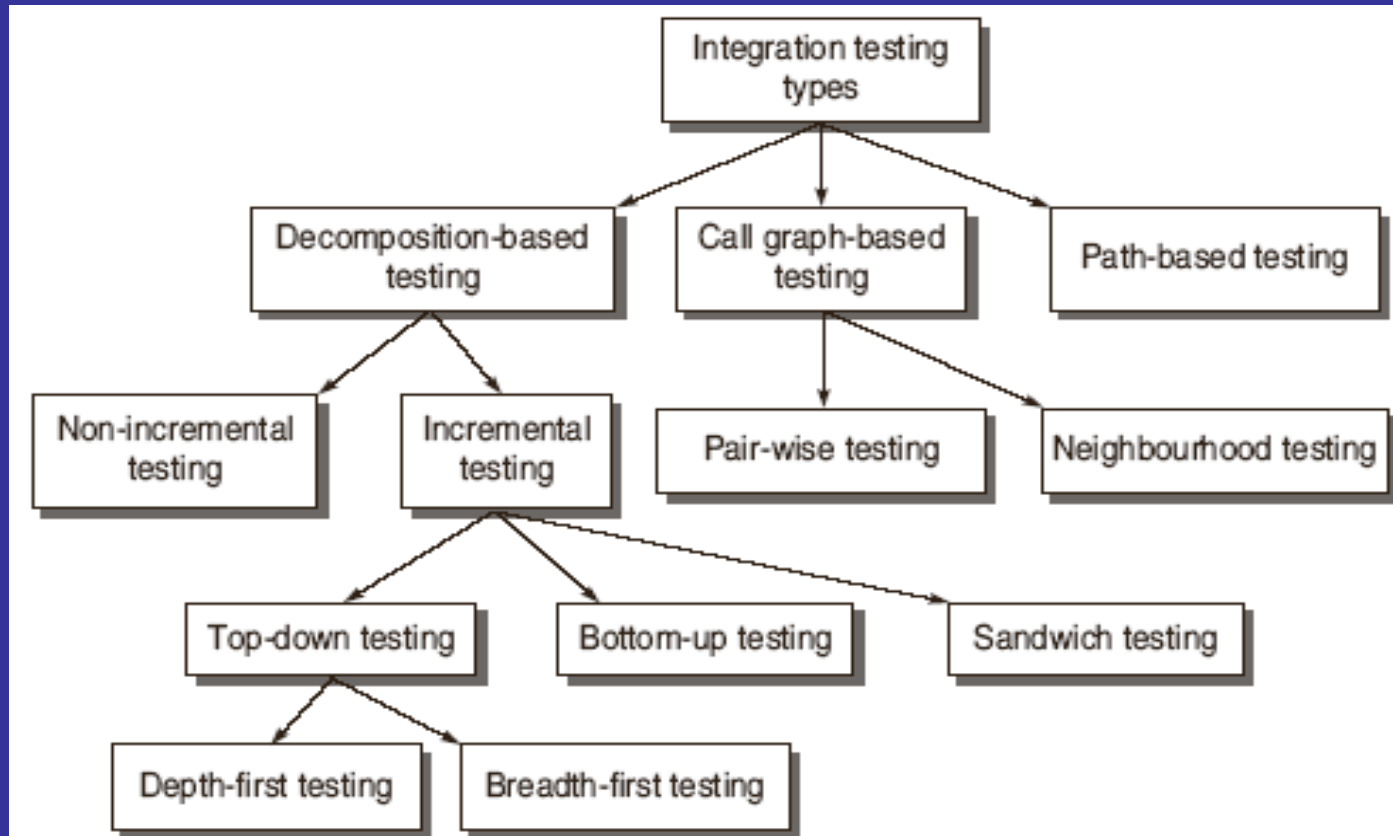


Integration Testing

- Integration testing exposes inconsistency between the modules such as improper call or return sequences.
- Data can be lost across an interface.
- One module when combined with another module may not give the desired result.
- Data types and their valid ranges may mismatch between the modules.



Integration Testing



Decomposition based integration testing

- Based on decomposition of design into functional components or modules.
- The integration testing effort is computed as number of test sessions. A test session is one set of test cases for a specific configuration.

The total test sessions in decomposition based integration is computed as:

Number of test sessions = nodes – leaves + edges

Decomposition based integration testing

1.Non-Incremental Integration Testing

- Big Bang Method

2.Incremental Integration Testing

- Top-down Integration Testing
- Bottom – up Integration Testing
- Practical approach for Integration Testing
 - Sandwich Integration Testing

Incremental Integration Testing

Issue	Top-Down Testing	Bottom-Up Testing
Architectural Design	It discovers errors in high-level design, thus detects errors at an early stage.	High-level design is validated at a later stage.
System Demonstration	Since we integrate the modules from top to bottom, the high-level design slowly expands as a working system. Therefore, feasibility of the system can be demonstrated to the top management.	It may not be possible to show the feasibility of the design. However, if some modules are already built as reusable components, then it may be possible to produce some kind of demonstration.
Test Implementation	$(nodes - 1)$ stubs are required for the subordinate modules.	$(nodes - leaves)$ test drivers are required for super-ordinate modules to test the lower-level modules.

Practical Approach for Integration Testing

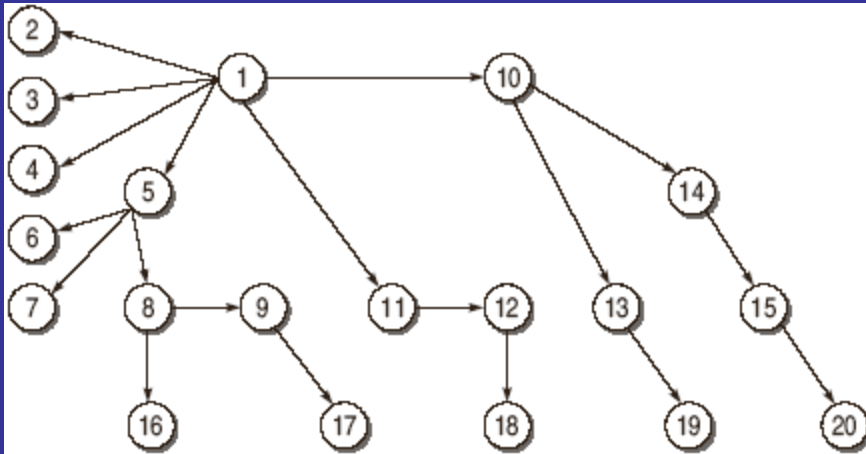
- ❖ There is no single strategy adopted for industry practice.
- ❖ For integrating the modules, one cannot rely on a single strategy. There are situations depending upon the project in hand which will force to integrate the modules by combining top-down and bottom-up techniques.
- ❖ This combined approach is sometimes known as Sandwich Integration testing.
- ❖ The practical approach for adopting sandwich testing is driven by the following factors:
 - Priority
 - Availability

Call Graph Based Integration

Refine the functional decomposition tree into a form of module calling graph, then we are moving towards behavioural testing at the integration level.

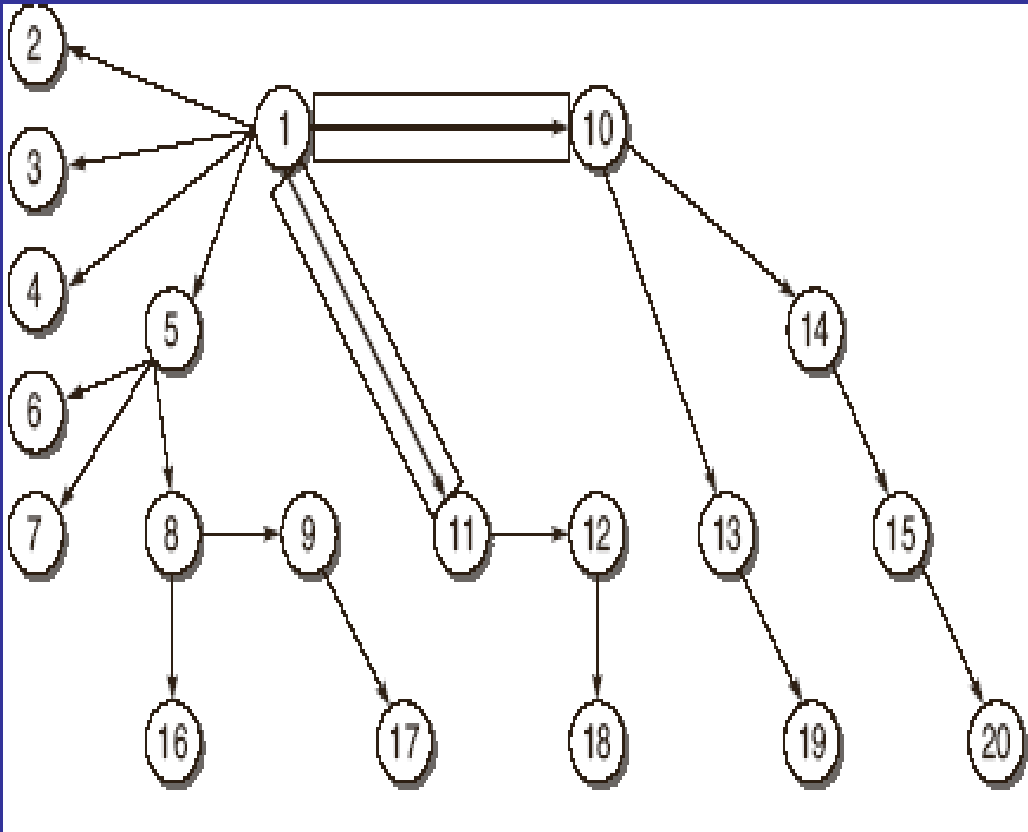
This can be done with the help of a call graph

A call graph is a directed graph wherein nodes are modules or units and a directed edge from one node to another node means one module has called another module. The call graph can be captured in a matrix form which is known as adjacency matrix.



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	x	x	x	x					x	x									
2																			
3																			
4																			
5					x	x	x												
6																			
7																			
8								x							x				
9																x			
10												x	x						
11											x								
12																	x		
13																		x	
14														x					
15																			x
16																			
17																			
18																			
19																			
20																			

Pair-wise Integration



Consider only one pair of calling and called modules and then we can make a set of pairs for all such modules.

Number of test sessions=no. of edges in call graph
= 19

Neighborhood Integration

Node	Neighbourhoods	
	Predecessors	Successors
1	-----	2,3,4,5,10,11
5	1	6,7,8
8	5	9,16
9	8	17
10	1	13,14
11	1	12
12	11	18
13	10	19
14	10	15
15	14	20

The neighborhood for a node is the immediate predecessor as well as the immediate successor of the node.

The total test sessions in neighborhood integration can be calculated as:

$$\begin{aligned}\text{Neighborhoods} &= \text{nodes} - \text{sink nodes} \\ &= 20 - 10 \\ &= 10\end{aligned}$$

where Sink Node is an instruction in a module at which execution terminates.

Path Based Integration

This passing of control from one unit to another unit is necessary for integration testing. Also, there should be information within the module regarding instructions that call the module or return to the module. This must be tested at the time of integration. It can be done with the help of path-based integration defined by Paul C.

Source Node : It is an instruction in the module at which the execution starts or resumes. The nodes where the control is being transferred after calling the module are also source nodes.

Sink Node: It is an instruction in a module at which the execution terminates. The nodes from which the control is transferred are also sink nodes.

Path Based Integration

Module Execution Path (MEP) : It is a path consisting of a set of executable statements within a module like in a flow graph.

Message:When the control from one unit is transferred to another unit, then the programming language mechanism used to do this is known as a message.

MM-Path (Path consisting of MEPs and messages):

MM-path is a set of MEPs and transfer of control among different units in the form of messages.

MM-Path Graph:

It can be defined as an extended flow graph where nodes are MEPs and edges are messages. It returns from the last called unit to the first unit where the call was made.

In this graph, messages are highlighted with thick lines.

Path Based Integration

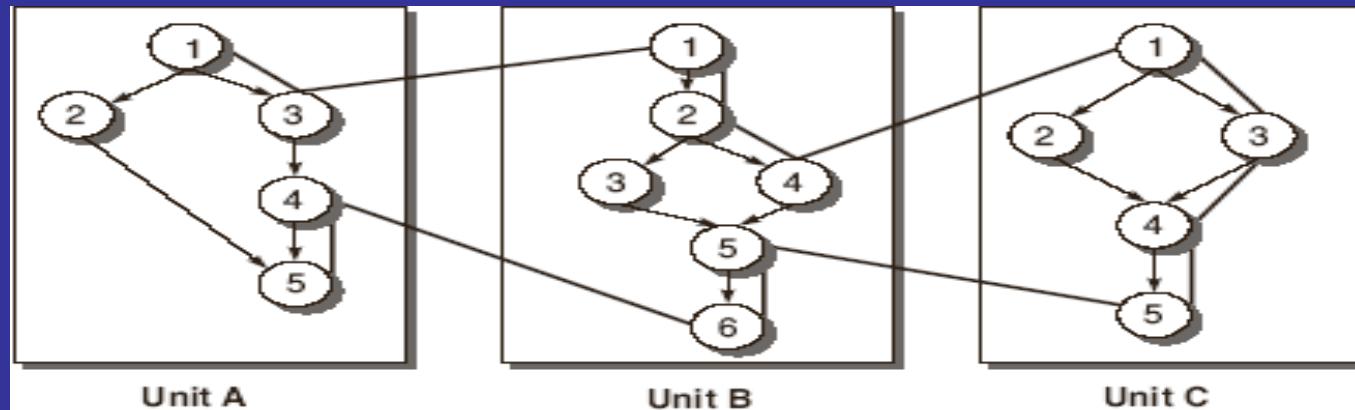


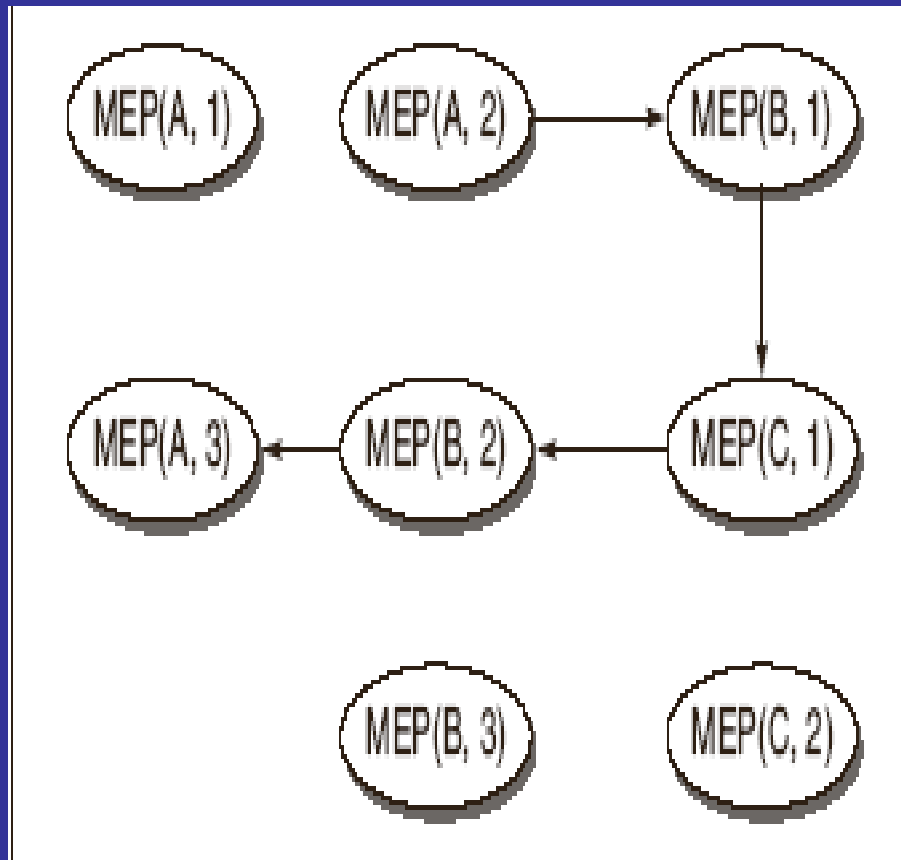
Figure 7.12 MM-path

Table 7.3 MM-path details

	Source Nodes	Sink Nodes	MEPs
Unit A	1,4	3,5	MEP(A,1) = <1,2,5> MEP(A,2) = <1,3> MEP(A,3) = <4,5>
Unit B	1,5	4,6	MEP(B,1) = <1,2,4> MEP(B,2) = <5,6> MEP(B,3) = <1,2,3,4,5,6>
Unit C	1	5	MEP(C,1) = <1,3,4,5> MEP(C,2) = <1,2,4,5>

Path Based Integration

MEP Graph



Function Testing

Functional Testing is a testing technique that is used to test the features/functionality of the system or Software.

Every functionality of the system specified in the functions is tested according to its external specifications. An external specification is a precise description of the software behaviour from the viewpoint of the outside world

- The process of attempting to detect discrepancies between the functional specifications of a software and its actual behavior.
- The function test must determine if each component or business event:
 - performs in accordance to the specifications,
 - responds correctly to all conditions that may be presented by incoming events / data,
 - moves data correctly from one business event to the next (including data stores), and
 - business events are initiated in the order required to meet the business objectives of the system.

Function Testing

The primary processes/deliverables for requirements based function test are discussed below:

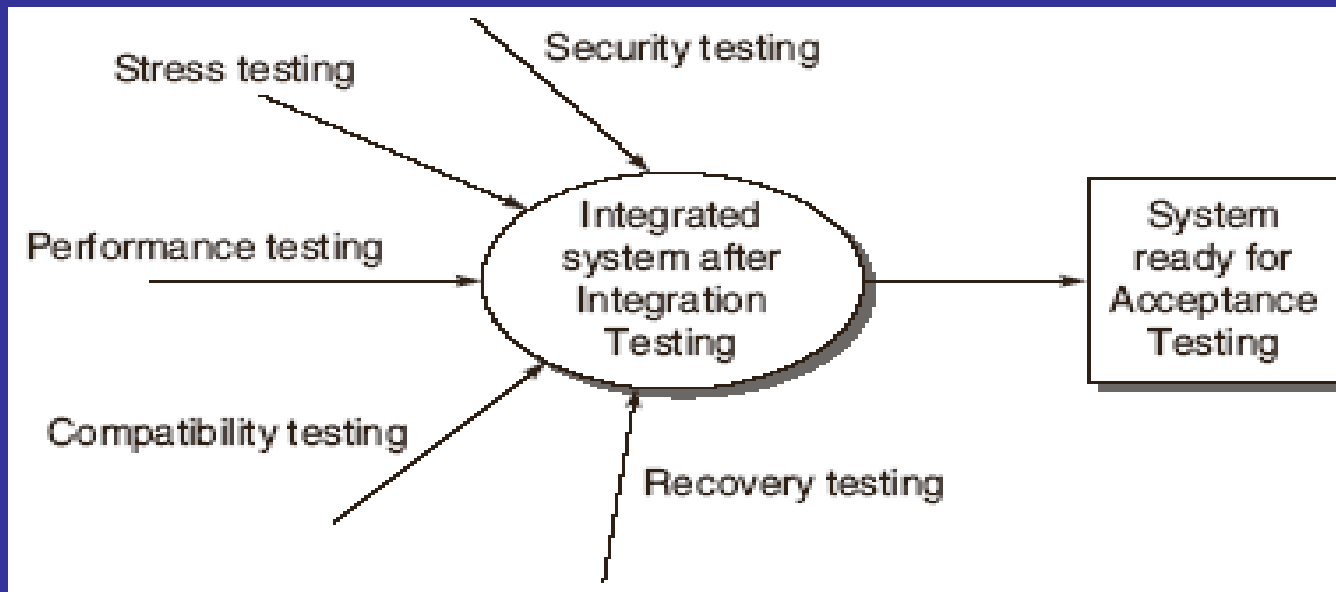
- **Test Planning:** During planning, the test leader with assistance from the test team defines the scope, schedule, and deliverables for the function test cycle.
- **Functional Decomposition**
- **Requirement Definition:** The testing organization needs specified requirements in the form of proper documents to proceed with the function test.
- **Test Case Design:** A tester designs and implements a test case to validate that the product performs in accordance with the requirements.
- **Function Coverage Matrix**

Functions	Priority	Test Cases
F1	3	T2,T4
F2	1	T1,T3

- **Test Case Execution**

System Testing

SYSTEM TESTING is a level of software testing where a complete and integrated software is tested. The purpose of this test is to evaluate the system's compliance with the specified requirements.



Categories of System Tests

Recovery Testing

- Recovery is just like the exception handling feature in a programming language.
- Recovery is the ability of a system to restart operations after the integrity of the application has been lost.
- It reverts to a point where the system was functioning correctly and then reprocesses the transactions up until the point of failure .
- Recovery Testing is the activity of testing how well the software is able to recover from crashes , hardware failures and other similar problems.
- It is the forced failure of the software in various ways to verify that the recovery is properly performed.
 - Checkpoints
 - Swichover

Security Testing

Security tests are designed to verify that the system meets the security requirements. Security may include controlling access to data, encrypting data in communication, ensuring secrecy of stored data, auditing security events, etc

- **Confidentiality**-It is the requirement that data and the processes be protected from unauthorized disclosure
- **Integrity**-It is the requirement that data and process be protected from unauthorized modification
- **Availability**-It is the requirement that data and processes be protected from the denial of service to authorized users
- **Authentication**- A measure designed to establish the validity of a transmission, message, or originator. It allows the receiver to have confidence that the information it receives originates from a specific known source.
- **Authorization**- It is the process of determining that a requester is allowed to receive a service or perform an operation. Access control is an example of authorization.
- **Non-repudiation**- A measure intended to prevent the later denial that an action happened, or a communication took place, etc.

Security Testing

Security Testing is the process of attempting to devise test cases to evaluate the adequacy of protective procedures and countermeasures.

- Security test scenarios should include negative scenarios such as misuse and abuse of the software system.
- Security requirements should be associated with each functional requirement. For example, the log-on requirement in a client-server system must specify the number of retries allowed, the action to be taken if the log-on fails, and so on.
- A software project has security issues that are global in nature, and are therefore, related to the application's architecture and overall implementation. For example, a Web application may have a global requirement that all private customer data of any kind is stored in encrypted form in the database

Security Testing

- Useful types of security tests includes the following:
 - Verify that only authorized accesses to the system are permitted
 - Verify the correctness of both encryption and decryption algorithms for systems where data/messages are encoded.
 - Verify that illegal reading of files, to which the perpetrator is not authorized, is not allowed
 - Ensure that virus checkers prevent or curtail entry of viruses into the system
 - Try to identify any “backdoors” in the system usually left open by the software developers

Performance Testing

Performance testing is to test the run time performance of the system on the basis of various parameters.

- The performance testing requires that performance requirements must be clearly mentioned in SRS and system test plans. The main thing is that these requirements must be quantified.
- For example, a requirement that the system return a response to a query in a reasonable amount is not an acceptable requirement; the time must be specified in a quantitative way.

Performance Testing

- Tests are designed to determine the performance of the actual system compared to the expected one
- Tests are designed to verify response time, execution time, throughput, resource utilization and traffic rate
- One needs to be clear about the specific data to be captured in order to evaluate performance metrics.
- For example, if the objective is to evaluate the response time, then one needs to capture
 - End-to-end response time (as seen by external user)
 - CPU time
 - Network connection time
 - Database access time
 - Network connection time
 - Waiting time

Stress Tests

- The goal of stress testing is to evaluate and determine the behavior of a software component while the offered load is in excess of its designed capacity
- The system is deliberately stressed by pushing it to and beyond its specified limits
- It ensures that the system can perform acceptably under worst-case conditions, under an expected peak load. If the limit is exceeded and the system does fail, then the recovery mechanism should be invoked
- Stress tests are targeted to bring out the problems associated with one or more of the following:
 - Memory leak: A failure in a program to release discarded memory
 - Buffer allocation: To control the allocation and freeing of buffers
 - Memory carving: A useful tool for analyzing physical and virtual memory dumps when the memory structures are unknown or have been overwritten.

Load and Stability Tests

- Tests are designed to ensure that the system remains stable for a long period of time under full load
- When a large number of users are introduced and applications that run for months without restarting, a number of problems are likely to occur:
 - the system slows down
 - the system encounters functionality problems
 - the system crashes altogether
- Load and stability testing typically involves exercising the system with virtual users and measuring the performance to verify whether the system can support the anticipated load
- This kind of testing help one to understand the ways the system will fare in real-life situations

JMeter
Performance Test

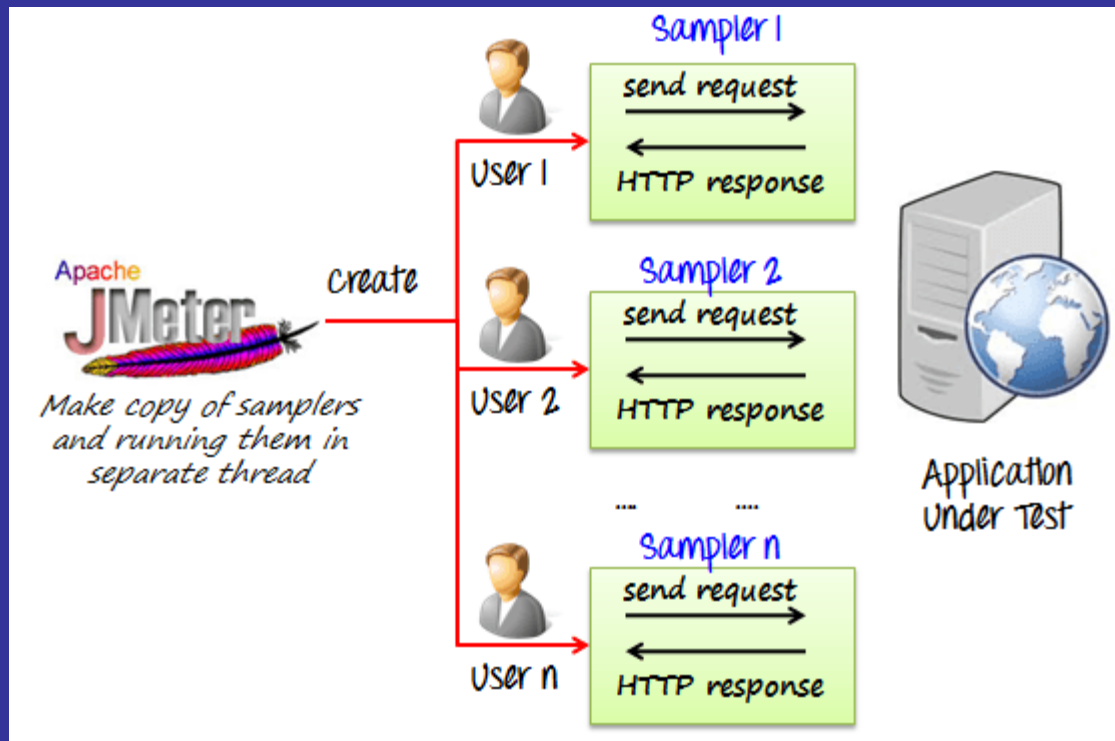
Includes



Load Test



Stress Test



Usability Testing

Testing to determine the extent to which the software product is understood, easy to learn, easy to operate and attractive to the users under specified conditions.

- **Ease of Use**
- **Interface steps**
- **Response Time**
- **Help System**
- **Error Messages**

Usability Testing

Graphical User Interface Tests

- Tests are designed to look-and-feel the interface to the users of an application system
- Tests are designed to verify different components such as icons, menu bars, dialog boxes, scroll bars, list boxes, and radio buttons
- The GUI can be utilized to test the functionality behind the interface, such as accurate response to database queries
- Tests the usefulness of the on-line help, error messages, tutorials, and user manuals
- The usability characteristics of the GUI is tested, which includes the following
 - ***Accessibility***: Can users enter, navigate, and exit with relative ease?
 - ***Responsiveness***: Can users do what they want and when they want in a way that is clear?
 - ***Efficiency***: Can users do what they want to with minimum number of steps and time?
 - ***Comprehensibility***: Do users understand the product structure with a minimum amount of effort?

Compatibility/Conversion/Configuration Testing

Compatibility Testing is a type of Software testing to check whether your software is capable of running on different hardware, operating systems, applications, network environments or Mobile devices.

- Operating systems: The specifications must state all the targeted end-user operating systems on which the system being developed will be run.
- Software/ Hardware: The product may need to operate with certain versions of web browsers, with hardware devices such as printers, or with other software, such as virus scanners or word processors.
- Conversion Testing: Compatibility may also extend to upgrades from previous versions of the software. Therefore, in this case, the system must be upgraded properly and all the data and information from the previous version should also be considered.
- Ranking of possible configurations(most to the least common, for the target system)
- Testers must identify appropriate test cases and data for compatibility testing.

Acceptance Testing

- Acceptance Testing is the formal testing conducted to determine whether a software system satisfies its acceptance criteria and to enable buyer to determine whether to accept the system or not.
 - Determine whether the software is fit for the user to use.
 - Making users confident about product
 - Determine whether a software system satisfies its acceptance criteria.
 - Enable the buyer to determine whether to accept the system.
-
- Alpha Testing Beta Testing

Acceptance Testing

Alpha Testing :

Alpha testing is a type of acceptance testing; performed to identify all possible issues/bugs before releasing the product to everyday users or public. The focus of this testing is to simulate real users by using blackbox and whitebox techniques. The aim is to carry out the tasks that a typical user might perform. Alpha testing is carried out in a lab environment and usually the testers are internal employees of the organization. To put it as simple as possible, this kind of testing is called alpha only because it is done early on, near the end of the development of the software, and before beta testing.

Acceptance Testing

Beta Testing:

Beta Testing of a product is performed by "real users" of the software application in a "real environment" and can be considered as a form of external user acceptance testing. Beta version of the software is released to a limited number of end-users of the product to obtain feedback on the product quality. Beta testing reduces product failure risks and provides increased quality of the product through customer validation. It is the final test before shipping a product to the customers. Direct feedback from customers is a major advantage of Beta Testing. This testing helps to test the product in real time environment.

Acceptance Testing

Entry Criteria for Alpha testing:

- Software requirements document or Business requirements specification
- Test Cases for all the requirements
- Testing Team with good knowledge about the software application
- Test Lab environment setup
- QA Build ready for execution
- Test Management tool for uploading test cases and logging defects
- Traceability Matrix to ensure that each design requirement has atleast one test case that verifies it

Exit Criteria for Alpha testing

- All the test cases have been executed and passed.
- All severity issues need to be fixed and closed
- Delivery of Test summary report
- Make sure that no more additional features can be included
- Sign off on Alpha testing

Acceptance Testing

Entrance criteria for Beta Testing:

- Positive responses from alpha sites
- Sign off document on Alpha testing
- Beta version of the software should be ready
- Environment ready to release the software application to the public
- Beta sites are ready for installation

Exit Criteria for Beta Testing:

- Feedback report should be prepared from public(good feedback)
- Prepare Beta test summary report
- Notify bug fixing issues to developers

References:

1. Software Testing Principles and Practices, Naresh Chauhan, Second edition, Oxford Higher Education
2. <https://www.guru99.com/software-testing.html>