UNIT-III – LEVELS OF TESTING

The need for Levers of Testing – Unit Test – Unit Test Planning – Designing the Unit Tests – The Test Harness – Running the Unit tests and Recording results – Integration tests – Designing Integration Tests – Integration Test Planning – Scenario testing – Defect bash elimination System Testing – Acceptance testing – Performance testing – Regression Testing – Internationalization testing – Ad- hoc testing – Alpha, Beta Tests – Testing OO systems – Usability and Accessibility testing – Configuration testing – Compatibility testing – Testing the documentation – Website testing.

NEED FOR LEVELS OF TESTING

- Execution based software testing, especially large systems, is usually carried out at different levels mostly 3-4 levels.
- Major Phases of testing:
- **Unit Testing**
- **Integration Testing**
- **System Testing**

✓ A Principal goal is to detect functional and structural defects in the unit.

✓ At the integration level several components are tested as group, and tester investigates component interaction.

✓ At the system level the system as a whole is tested and a principal goal is to evaluate attribute such as ability, reliability and performance **Level of Testing and Software Development Paradigms**

- The approach used to design and develop a software system has an impact on how testers plan and design suitable tests.
- The TWO major approaches to system development are **Bottom up and Top down approaches.**

  These approaches are supported by two major types of programming languages- **procedure Oriented and Object Oriented.**

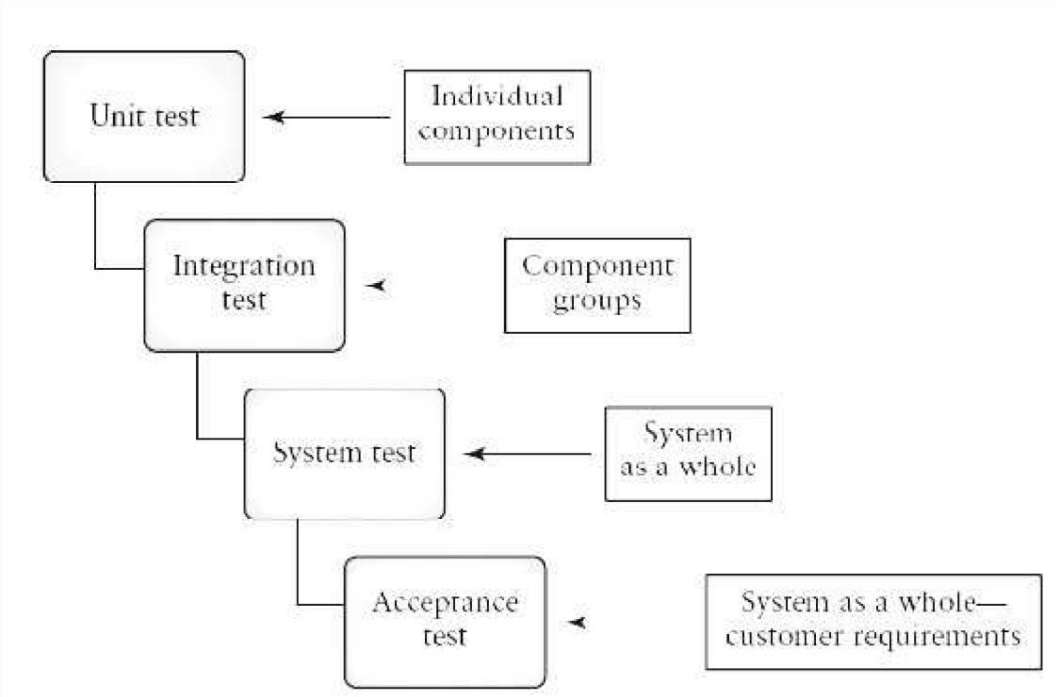


FIG 3.1 LEVELS OF TESTING

- The different nature of the code produced requires testers to use different strategies to identify and test components and component groups.
- Systems developed with procedural languages are generally viewed as being composed of passive data and active procedures
- When test cases are developed the focus is on generating input data to pass to the procedures (or functions) in order to reveal defects.

- Object oriented systems are viewed as being composed of active data along with allowed operations on that data, all encapsulated within a unit similar to abstract data type.

UNIT TESTING

Unit test: Functions, Procedures, Classes, and Method as Unit

- **A workable definition for a software unit is as follows**

- A Unit is the smallest possible testable software component
- It can be characterized in several ways. For example a unit in a typical procedure oriented software system:
- perform a single cohesive function
- can be compiled separately
- is a task in a work breakdown structure (from the manager's point of view)
- contains code that can fit on a single page or screen. ● A unit is traditionally viewed as a function or procedure implemented in a procedural (imperative) programming language.
- A unit may also be a small sized COTS component purchased from an outside vendor that is undergoing evaluation by the purchaser, or simple module retrieved from an in-house reuse library

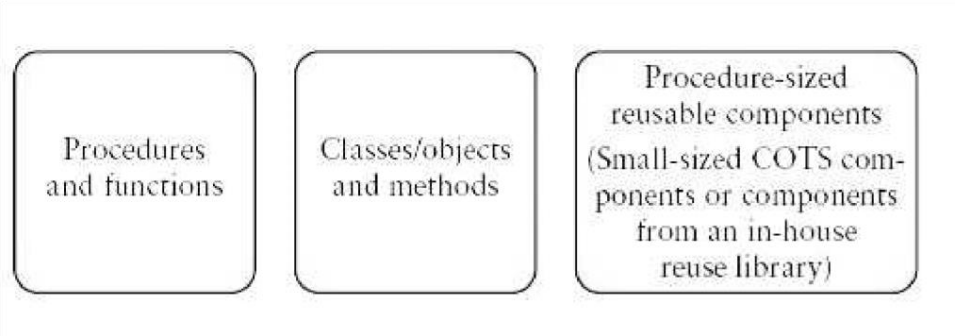


FIG. 3.1 SOME COMPONENTS SUITABLE FOR UNIT TEST

Unit Test- The Need for Preparation

- The principal goal for unit testing is insure that each individual software is functioning according to its specification
- Good testing practice calls for unit tests that are planned and public. Planning includes designing test to reveal defects such as functional description defects, algorithmic defects, data defects, and control logic and sequence defects.
- Resources should be allocated and test cases should be developed, using both white and black box test design strategies.
- The unit should be tested by an independent tester (other than testers) and the test results and defects found should be recorded as apart of the unit history (made public).
- Each unit should also be reviewed by a team of reviewers, preferably before the unit test.
- Unit test in many cases is performed informally by the unit developer soon after the module is completed, and it compiles cleanly.
- Some developers also perform an informal review of the unit.
- To prepare for unit test the developers/ testers must perform several tasks. These are:-
    1. **Plan the general approach to unit testing**
    2. **Design the test cases, and test procedures**
    3. **Define relationships between the test**
    4. **Prepare the auxiliary code necessary for unit test.**

Unit test Planning

- A general unit test plan should be prepared.

- It may be prepared as a component of the master test plan or a stand alone plan.
- It should be developed in conjunction with the master plan and the project plan for each project
- **Phase 1 : <u>Describe Unit test Approach and Risks</u>**

In this phase of unit test planning the general approach to unit test planning is outlined: The test planner:

- identifies test risks
- describes techniques to be used for designing the test cases for units
- describes techniques to be used for data validation and recording of test results
- describes the requirement for test harness and other software that interfaces with unit to be tested eg:- any special software needed for testing object oriented units

  - During this phase the planner also identifies completeness requirements ie what will be covered by the unit test and to what degree (state, functionality, control, and data flow patterns)
  - Planner also identifies termination condition for unit test. ○ They include coverage requirement and special cases ○ Special cases may result in abnormal termination of unit test ○ Planner estimate the resources needed for unit test such as hardware, software and staff and develop tentative schedule under constraints identified at that time

<u>Phase 2:- Identify Unit Features to be Tested</u>

- This phase requires information from the unit specification and detailed design description

- The planner determines which features of each unit will be tested, for example functions, performance requirement, state and state transition, control structures, messages and data flow patterns
- Some features will be covered by the tests, they should be mentioned and risks of not testing them be assessed.
- Input and output of each test unit should be identified. **Phase 3: Add levels of Detail to the Plan**
- In this phase the planner refines the plan as produced in the previous two phases.

  The planner adds new details to the approach, resource and scheduling portions of the unit test plan.
- **Eg:-** Existing test cases that can be reused for this project can be identified in the phase.
- Unit availability and integration scheduling information should be included in the revised version of the test plan.
- Planner must be sure to include a description of how test results will be recorded.
- Test related documents that will be required for this task eg. test logs, test incidents report should be described.

Designing the Unit test

Part of the preparation work for unit test involves unit test design. It is important to specify

1. The test cases (including I/O and expected output for each test cases) and
2. The test procedures (steps required run the test)

- As a part of the unit test design process, developers / tester should also describe the relationship between the tests. Test suites can be defined that binds related tests together as a group. All of this test design information is attached to the unit test plan. Test cases, test procedures and test suites may be reused from the past projects if the organization has been careful to store them so that they can be easily retrievable and reusable
- Test case design at unit level can be based on the use of black and white box design strategies. Both of these approaches are useful for designing test cases for functions and procedures.

The Test Harness

- ***The auxiliary code developed to support testing of units and components is called as test harness***
- ***The harness consist of drivers that call the target code and stubs that represent modules it calls.***
- The development of drivers and stubs requires testing resources. The drivers and stubs must be tested themselves to insure they are working properly and that they are reusable for subsequent releases of the software
- Drivers and stubs can be developed at several levels of functionality
- **Eg:-** a driver could have the following options and combinations of options:

i. Call the target unit

ii. Do 1, and pad pass input parameters from the table

iii. Do 1,2, and display parameters

iv. Do 1,2,3 and display result (output parameters) The stub should also exhibit different levels of functionality. For example a stub could

i. Display a message that it has been called the target unit

ii. Do1, and display any input parameters passes from the target units

iii. Do1,2, and pass back result from a table

iv. Do1,2,3 and display result from table

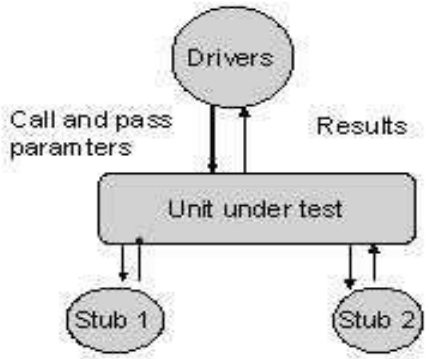


FIG. 3.5 THE TEST HARNESS

Running a Unit tests and Recording Results

Unit test can begin when

- the unit become available from the developers
- the test cases have been designed and reviewed
- the test harness and any other supplemental to supporting tools

The status of the test efforts for a unit, and a summary of test results must be recorded in a unit test worksheet.

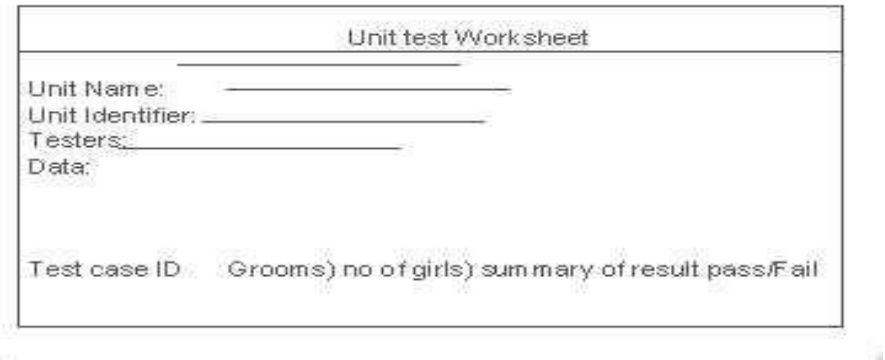


FIG. 3.6 SUMMARY WORK SHEET FOR UNIT TEST RESULTS

- The tester must determine from the results whether the unit has passed or failed the test
- If the test is failed, the nature of the problem should be recorded in what is sometimes called the test incident report.
- Differences from expected behavior should be described in detail.
- When a unit fails a test there may be several reasons for the failure. The most likely reason for the failure is a fault in the unit implementation

i. A fault in the test case

ii. A fault in test procedures execution

iii. A fault in the test environment

iv. A fault in the unit design

- When a unit has been completely tested and finally passes all of the

required tests it is ready for integration.

INTEGRATION TESTING

Testing the interaction between the modules and interaction with other systems externally.

Integration Test-Goals

- Integration test for procedural code has two major goals:
  i. To detect defects that occur on the interfaces of units
  ii. To assemble the individual unit into working subsystem and finally a complete system that is ready for system test.
- In unit test the tester attempts to detect defects that are related to the functionality and structure of the unit.
- Integration test should only be performed on unit that have been reviewed and have successfully passed unit testing.
- Integration testing works best as an iterative process procedural oriented system.
- One unit at a time integrated into a set of previously integrated modules which have passed a set of integration tests.
- The interface and functionality of the new unit is combination with the previously integrated units is tested
- When a subsystem is built from units integrated in the stepwise manner, then performance, security and stress test can be performed in this subsystem.

Designing Integration tests

- Integration test for procedural software can be designed using a black or white box approach.

- Some unit test can be reused.
- Since many error occur at module interfaces, test designers need to focus on exercising all input/output parameter pairs and all calling relationships
- The tester needs to insure the parameters are of the correct type and in the correct order.

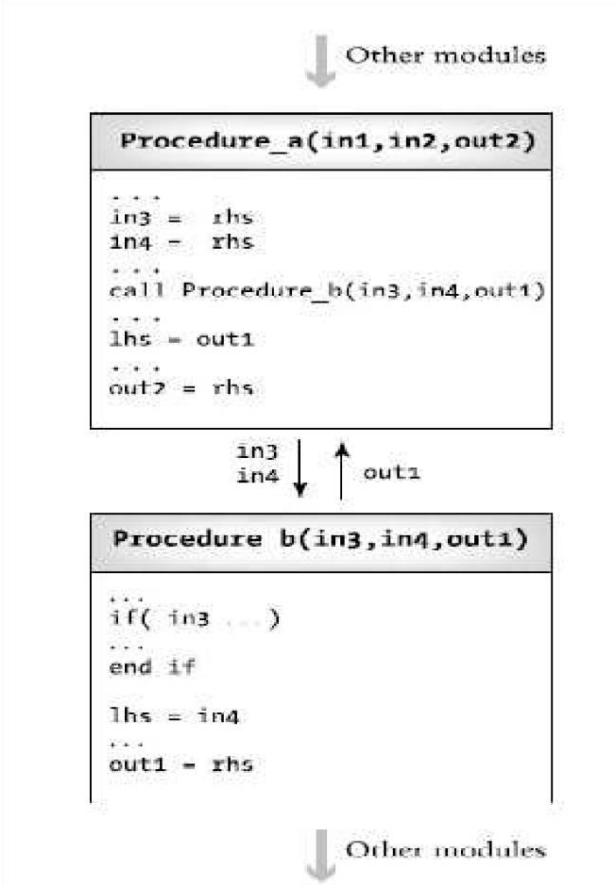


FIG.3.7 EXAMPLE INTEGRATION OF TWO PROCEDURES

- In the example above, Procedure_b has two input parameters int3,int4.
- Procedure_b uses those parameters and then returns a value for the output parameter out1.
- The terms such as *rhs* and *lhs* could be any variable or expression.
- The parameter could be involved in a number of *def* and/or *use* data flow patterns
- The actual usage patterns of the parameters must be checked at

integration time.

- Some black box test used for module integration may be reusable from unit testing.
- When units are integrated and subsystems are to be tested as a whole, new tests will a have to be designed to cover the functionality tests at the integration level are the requirements document and the user manual.
- Tester need to work with requirement analyst to insure that the requirements are testable, accurate and complete.
- Black Box tests should be developed to insure proper functionally and ability to handle subsystem stress.
- Integration Testing of clusters of classes also involves building test harness which in this case are special classes of objects built for testing
- In class testing we evaluated intra class method interaction, and at the cluster level we test inter class method interaction as well
- We want to insure that message are being passed properly to interfacing objects, object state transition are correct when specific events occur , and that the cluster are performing their required functions.

Integration test Planning

- Integration test must be planned.
- Planning can begin when high level design is complete so that the system architecture is defined.
- Documents relevant to integration test planning are the requirement document, the user manual and usage scenarios.
- These documents contain structure charts, the state charts and data dictionary , cross reference table, module interface description
    - The strategy for integration of the unit must be defined .

- The detailed description is given :-
- Cluster this cluster is dependent on
- A natural language description of the functionality of the cluster to be retested.
- List a classes in the cluster
- A set of cluster test cases

What is Integration Testing:-

- ✓ **Integration testing is both a type of testing and phase of testing**.
- ✓ Integration is defined to be a set of interactions, all defined interaction among the components need to be tested.

Integration Testing As a Type of Testing

:✓ Integration testing means testing of interfaces.

- ✓ They are ***Internal*** Interfaces and ***Exported*** or ***External Interfaces***
- ✓ Internal Interfaces are those that provide communication across two modules within a project or product, internal to the product, and not exposed to the customer or external developers.
- ✓ Exported interfaces are those that are visible outside the product to third party developers and solution providers.

"Integration Testing Type Focuses on testing interfaces that are "Implicit and Explicit" and "Internal and External"

Methodologies for deciding the order of integration testing

There are several methodologies available, to in decide the order for integration testing.

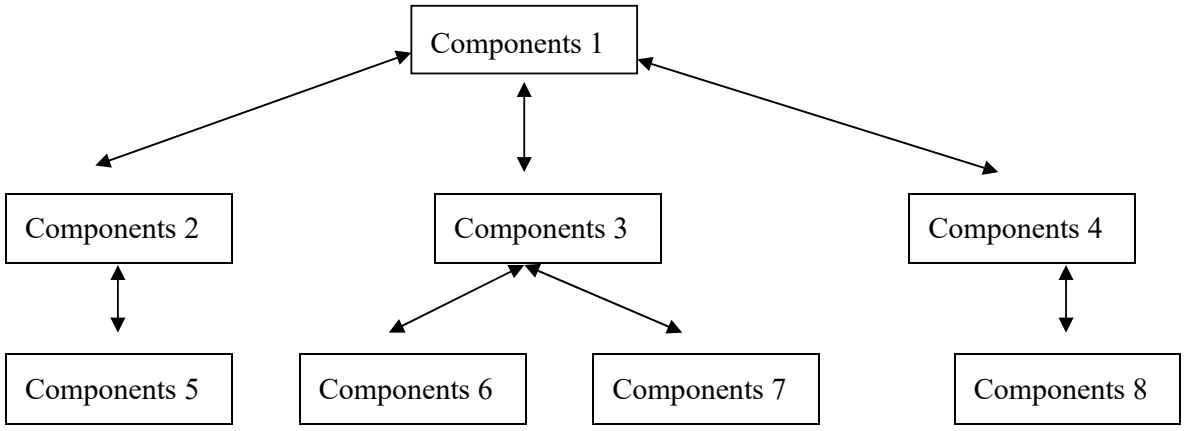
These are as follows:-

1. **Top Down Integration**
2. **Bottom up Integration**
3. **Bi-Directional Integration**
4. **System Integration**

Top-Down Integration:

Integration Testing involves testing the topmost component interface with other components in same order as you navigate from top to bottom, till we cover all the components.

To understand this methodology, we will assume that a new product/ software development where components become available one after another in the order of component number specified .The integration starts with testing the interface between Component 1 and Component 2 .To complete the integration testing all interfaces mentioned covering all the arrows, have to be tested together. The order in which the interfaces are to be tested is depicted in the table below. In an incremental product development, where one or two components gets added to the product in each increment, the integration testing methodology pertains to only those new interfaces that are added .



Order of testing Interfaces

Steps	***Interfaces Tested***
1	***1-2***

2	***1-3***
3	***1-4***
4	***1-2-5***
5	***1-3-6***
6	***1-3-6-(3-7)***
7	***(1-2-5)-(1-3-6-(3-7))***
8	***1-4-8***
9	***(1-2-5)-(1-3-6-(3-7))-(1-4-8)***

Bottom-Up Integration:-

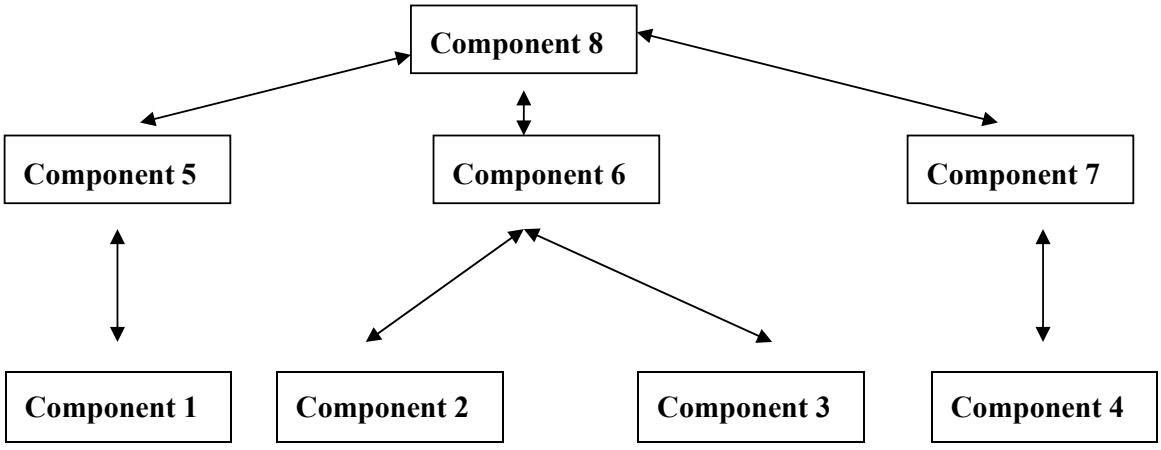
Bottom-up integration is just the opposite of top-down integration, where the components for a new product development become available in reverse order, starting from the bottom.

Testing takes place from the bottom of the control flow upwards.

Components or systems are substituted by drivers.

Logic Flow is from top to bottom and integration path is from bottom to top. Navigation in bottom-up integration starts from component 1 converting all sub systems , till components 8 is reached. The order is listed below. The number of steps in the bottom up can be optimized into four steps.

By combining step2 and step3 and by combining step 5-8 in the previous table.



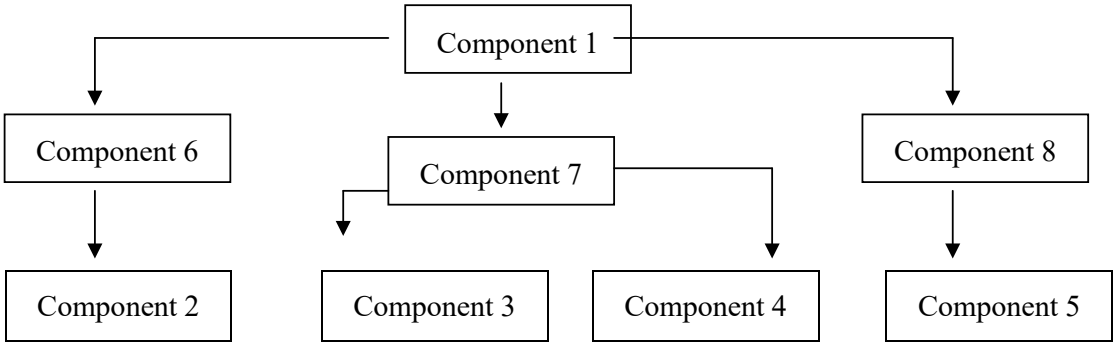
Order of Interface tested using Bottom Up Integration

Steps	*Interfaces Tested*
1	*1-5*
2	*2-6,3-6*
3	*2-6-(3-6)*
4	*4-7*
5	*1-5-8*
6	*2-6-(3-6)-8*
7	*4-7-8*
8	*(1-5-8)-(2-6-(3-6)-8)-(4-7-8)*

Integration:

Bi directional integration is a combination of the top-down and bottom –up integration approaches used together to derive integration steps.

This approach is also called as "*Sandwich Integration*".



Steps for Integration Using Sandwich Testing :-

Steps	*Integration Tested*
1	*6-2*
2	*7-3-4*
3	*8-5*

4	*(1-6-2)-(1-7-3-4)-(1-8-5)*

System Integration:

System Integration means that all the components of the system are integrated and tested as a single unit.

Integration testing, which is testing of interface, can be divided into two types:-

- ☞ **Components or Sub-System Integration**
- ☞ **Final Integration testing or system Integration**

Big bang Integration is deal for a product where the interfaces are stable with less number of defects.

Choosing Integration Methods:-

Sno	*Factors*	*Suggested Integration Methods*
1	**Clear Requirement and Design**	***Top Down***
2	**Dynamically, Changing Requirements, Design, Architecture**	***Bottom-Up***
3	**Changing Architecture, Stable Design**	***Bi-Directional***
4	**Limited Changes to existing Architecture with less Impact**	***Big Bang***
5	**Combination of all the above**	***Select one of the above after careful analysis***

Integration Testing As a Phase of testing :-

"All testing activities that are conducted from the point where two components are integrated to the point where all system components work together are considered a part of the integration testing phase."

The Integration testing phases focuses on finding defects which predominantly arise because of combining various components for testing, and should not be focused on for component or few components .Integration testing as a type focuses on testing the interfaces. This is a subnet of the integration testing phase.

SCENARIO TESTING

Scenario testing is defined as a "set of realistic user activities that are used for evaluating the products" .It is also defined as testing involving customer scenarios.

There are two methods to evolve scenario

1. **System Scenario**
2. **Use case Scenario/ Role Based Scenarios.**

System Scenario:-

System Scenario is a method where by the set of activities used for scenario testing covers several components in the system.

The **following approaches** can be used to develop system scenarios.

Story-line:

Develop a story-line that combines various activities of the product **Life-cycle / state transitions**:

Consider an object, derive the different transitions / modification that happen to the object and derive scenarios to cover them

Deployment / implementation details from customer:

Develop a scenario from a known customer Deployment / implementation details and create a set of activities by various users in that implementation **Business verticals**:

Visualizing how a product / software will be applied to different business verticals and create a set of activities as scenarios (e.g., insurance, life sciences)

Battle-ground scenarios:

Create some scenarios to justify "the product works" and some scenarios to "try and break the system" to justify "the product doesn't work."

Use Case Scenarios:-

- ✓ ***A use case Scenario is a stepwise procedure on how a user intends to use a system, with different user roles and associated parameters.***
- ✓ A use case scenario can include stories, pictures and deployment details.
- ✓ Use cases are useful for explaining customer problems and how the software can solve those problems without any ambiguity

Example:-

The scenario below is an example of withdrawing a cash from a bank. A customer fills up a cheque and gives it to an official in the bank. The official verifies the balance in the account from the computer and gives the required cash to the customer .The customer in this example is a actor, the clerk the agent , and the response given by the computer which gives the balance in the account , is called the system response.

Actor	System Response
User would like to withdraw cash and inserts the card in ATM machine	Request for Password or PIN (Personal identification number)
User Fill-in the Password or PIN	
	Validate the password or PIN
	Give a list containing types of accounts
User selects an account type	
	Ask the user for amount
User Fill-in the amount of cash required	
	Check availability of funds Update account balance Prepare receipt Dispense cash
Retrieve cash from ATM	
	Print receipt

Actor and System Response in Use Case for ATM cash withdrawal

DEFECT BASH

- **Defect bash is an *ad hoc* testing, done by people performing different roles to bring out all types of defects.**
- It is very popular among application development companies, where the products can be used by people who perform different roles.
- The testing by all the participants during the defect bash is not based on written test cases.

- Defect bash brings together plenty of **good practices** that are popular in testing industry. They are as follows :-
    1. Enabling people to "***cross boundaries and test beyond assigned area***"
    2. Bringing different people performing different roles together in the organization for testing - "***Testing isn't for testers alone***"
    3. Let everyone in organization use the product before delivery - "***Eat your own dog food***"
    4. Bringing fresh pairs of eyes to uncover new defects – "***Fresh eyes have less bias***"
    5. Bringing in people who have different levels of product understanding, to test the product together randomly – "***Users of software are not the same***"
    6. Testing doesn't wait for the time taken for documentation – "***Does testing wait till all documentation is done?***"
    7. Enabling people to say the "system works" as well as enabling them to "break the system" – "***Testing isn't to conclude that the system works or doesn't work***"

Even though it is said that defect bash is an ad hoc testing, not all activities of defects bash are unplanned. All the activities in the defect bash are planned activities, except for what to be tested .

It involves **several steps:-**

1. **Choosing the frequency and duration of defect bash.**
2. **Selecting the right product build.**
3. **Communicating the objectives of each defect bash to everyone**
4. **Setting up and monitoring the lab for defect bash.**

5. **Taking action and fixing issues.**
6. **Optimizing the effort involved in defect bash.**

1. **Choosing frequency and duration**
    - Too frequent or too few rounds may not meet objective
    - Optimize duration involved
2. **Selecting right product build**
    - Good-quality product
    - Regression tested build
    - Too many defects spoil confidence
3. **Communication objective of defect bash**
    - Purpose & objective has to be clear
    - Areas of focus to be communicated
    - Defects that can be found easily by test team shouldn't be objective
4. **Setting up and monitoring lab**
    - Right configuration and resources
    - Easy install & set-up help
    - Optimized for both functional & non-functional defects
    - Monitor all resources (RAM, disk, CPU, network)
5. **Taking actions and fixing issues**
    - Duplicate defects
    - Not possible to look at each defect alone due to volume
    - Code reviews and inspections
    - Communication to all users on defects and their resolution

SYSTEM TESTING

System Test- The Different Types

- When integration tests are completed, a software system has been assembled and its major subsystems have been tested.
- System test planning should begin at the requirement based (black box) test.
- System test planning is a complicated task. There are many components of the plan that need to be prepared such as test approaches, costs, schedules, test cases and test procedures.
- System testing itself requires large amount of resources.
- **The goal is to ensure that the system performs according to its requirements.**
- **System test evaluates both functional behavior and quality requirement such as reliability, usability, performance and security.**
- **The phase of testing is especially useful for detecting external hardware and software interface defects.** Eg:- those causing race conditions, deadlocks, problems with interrupts and exception handling.

- There are several types of system tests
- Functional testing
- Performance Testing
- Stress testing
- Configuration testing
- Security Testing
- Recovery testing
- Two other types of system testing called reliability and usability testing.

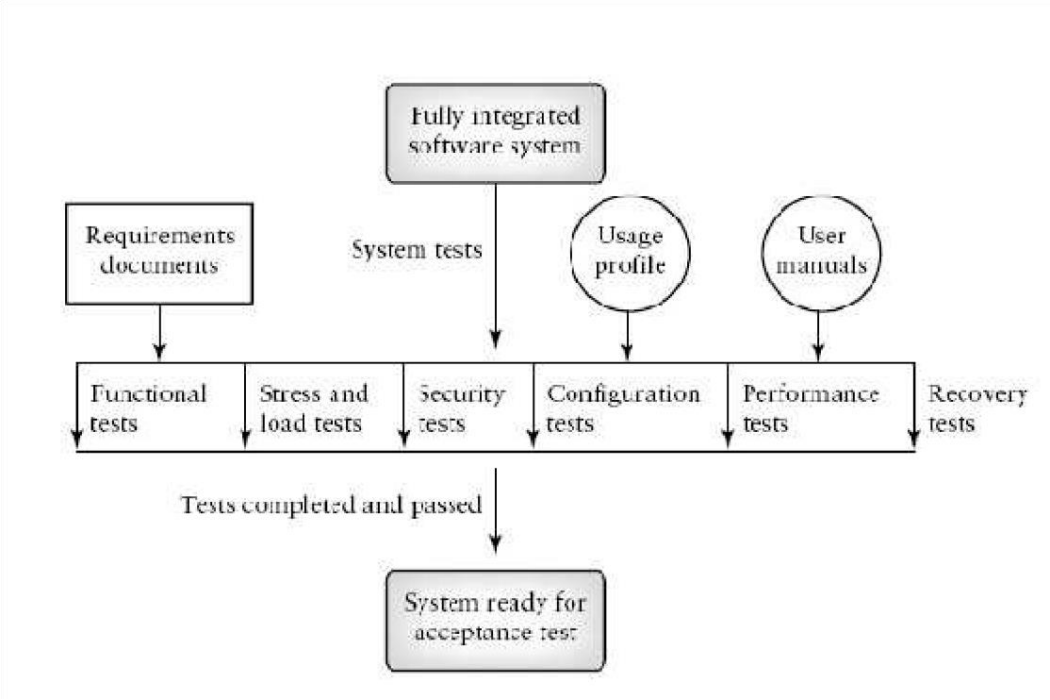


FIG 3.8 TYPES OF SYSTEM TESTS

An important tool for implementing system tests is a load generator. A load generator is essential for testing quality requirements such as performance and stress.

A load is a serious of input that stimulated a group of transaction

- A transaction is a unit of work seen from the system user's view. A transaction consist of set of operations that may be performed by a person , software system or a device that is outside the system.
- A use case can be used to describe a transaction. If you were system testing a telecomm system you would need a load that simulated a series of phone calls (transactions) of particular types and lengths arriving from different locations
- A load can be a real load, which is we can put the system under test to real usage by having actual telephone users connected to it.
- Loads can also produced by tools called load generators; they will generate test input data from system test. Load generators can be simple tools that outputs a fixed set of predetermined transaction

1.Performance Testing

- ℘ To evaluate the time taken by the system to perform its required function in comparison with different versions of the same product

2.Scalability testing

- ℘ To find out the maximum capability of the system parameters

3.Reliability testing

- ℘ To evaluate the ability of the system to perform its required functions repeatedly for a specified period of time

4.Stress testing

- ℘ Evaluating a system beyond the limits of the specified requirements to ensure the system does not break down unexpectedly

4.Interoperability testing

- To ensure that two or more products can exchange information, use the information and work closely

5.Localization testing

- Testing conducted to verify that the localized product works in different languages

Why system testing done?

- Helps in identifying as many defects as possible before the customer finds them in the deployment
- Last chance for the testing team to find any remaining defects before the product is handed over to the customer
- Objective to find product level defects
- Test both functional & non functional aspects of the product
- Build confidence in the product
- Test the product behavior in a complete and realistic environment
- Ensure all the requirements are met and ready the product for acceptance testing

FUNCTIONAL TESTING

- **Functional tests at system level are used to ensure that the behavior of the system adheres to the requirements specification.**
- All functional requirements for the system must be achievable by the system.
- For example , if a personal finance system is required to allow users to set up account, add, modify and delete entries in the accounts, and print reports, the function based system and acceptance test must ensure that the system can perform these tasks

- **Functional test are black box in nature**
- **The focus is on the inputs and proper output for each function**
  - **Improper and illegal inputs must also be handled by the system**
  - System behavior under the latter circumstances.

The test should focus on the following goals

- ✓ All types or classes of legal input must be accepted by the software
- ✓ All classes of illegal inputs must be rejected
- ✓ All possible classes of system output must exercised and examined
- ✓ All effective system states and state transition must be exercised and examined
- ✓ All functions must be exercised

PERFORMANCE TESTING

- ℘ **The goal of system performance tests is to see if the software meets performance requirements.**
- ℘ **Confirms whether there are any hardware or software factors that impact on the systems requirement.**
- ℘ **Resources for the performance testing must be allocated in the system test plan**

- There are two major requirements:
- **Functional Requirement**: Users describe what function the software should perform. We test for compliance of these requirements at the system level with the functional based system test.
- **Quality Requirement**:- There are nonfunctional in nature but describes quality levels expected for the software. One example of a quality requirement

is performance level, the users may have objectives for the software system in terms of memory use, response time, throughput and delays

- Performance testing allows the testers to tune the system, ie to optimize the allocation of system resource.
- Performance objectives must be articulated clearly by the users/client in the requirement documents , and stated clearly in the system test plans

- Objective must be quantified
- Example of resources are given below in a diagram

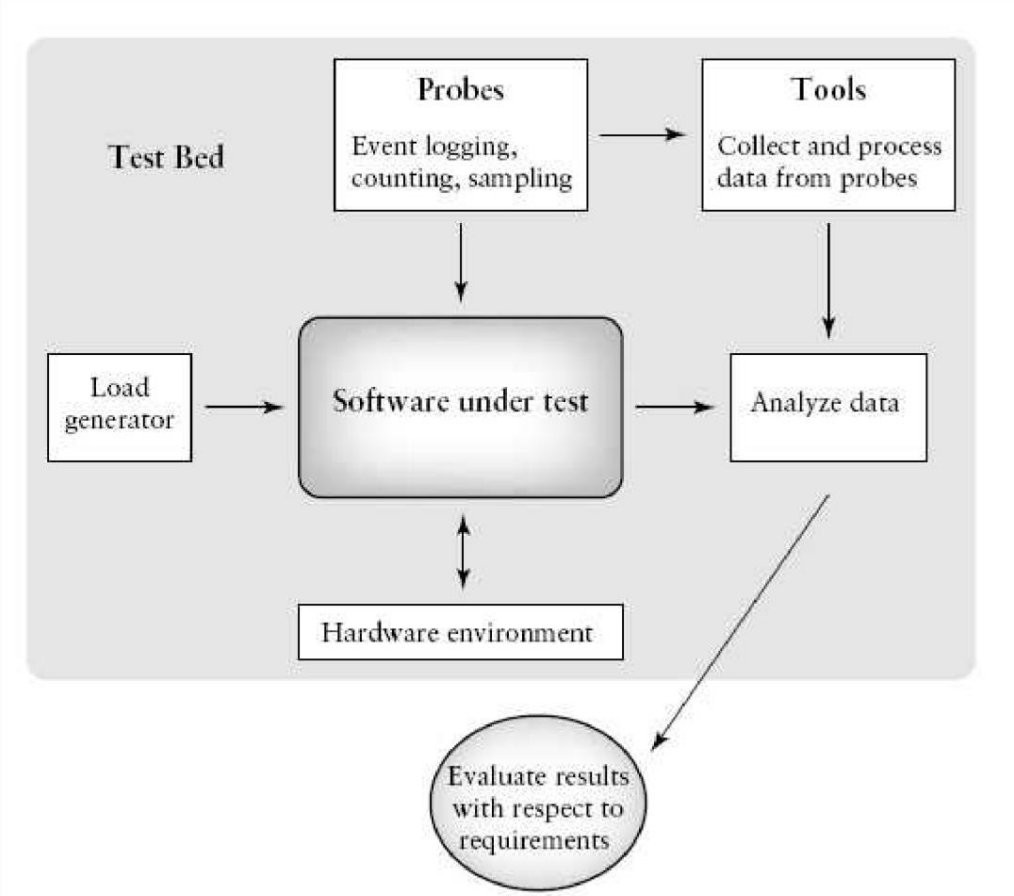


FIG. 3.9 EXAMPLES OF SPECIAL RESOURCES NEEDED FOR A PERFORMANCE TEST

- A source of transaction to drive the experiments. For example if you were performance testing an operating system you need a stream of data that represent typical user interactions.
- Typically the source of transaction for many systems is load generator.

-
- An experimental test bed that includes hardware and software the system under test interacts with. The test bed requirement sometimes includes special laboratory equipment and space that must be reserved for the tests. Instruments or probes that help to collect the performance data, probes may be hardware or software in nature.
- Some probe tasks are event counting and event duration measurement.
- Eg:- if you are investigating memory requirements for your software you could use a hardware probe that collected information on memory usage as the system executes
- The tester must keep in mind that the probes themselves may have an impact on system performance
- A set of tools to collect, store, process and interpret the data.

Very often, large volume of data are collected, and without tools the testers may have difficulty in processing and analyzing the data in order to evaluate true performance levels.

STRESS TESTING

- **The goal of stress test is to try to break the system; find the circumstance which it will crash, this is sometimes called "*breaking the system".***
- **Stress testing often uncovers race conditions, deadlocks**
- **The goal of stress test is to try to break the system; find the circumstance which it will crash, this is sometimes called "*breaking the system*".**

- **Stress testing is important because it can reveal defects in real time** and other types of systems, as well as weak areas where poor design could cause unavailability of services.
- **Stress testing often uncovers race conditions, deadlocks, depletion of resource in unusual or un planned patterns, and upset in normal operation of the software system.**
- System limits and threshold values are exercised
  Stress testing is important from the user/client point of view. When system operate correctly under conditions of stress then client have confidence that the software can perform as required.

CONFIGURATION TESTING

- **Allows testers to evaluate system performance and availability when hardware exchanges and reconfigurations occur.**
- To test the software using configuration testing, many resources such as multiple hardware devices are needed
- Software Systems interact with hardware devices such as disc drivers, tape drivers and printers. Many Software system also interact with multiple CPU some of which are redundant
- Eg:- a printer of type X should be substitutable for a printer of type Y,
  CPU A should be removable from a system composed of several other CPUs
- Sensor A should be replaced with Sensor B
- **Configuration testing has the following objectives**
  i. **Show that all configuration changing commands and menus work properly**

- ii. **Show that all interchangeable and that they each enter the proper states for the specified conditions**
- iii. **Shows that the system performance level is maintained when devices are interchanged, or when they fail**

- Several types of operations should be performed during configuration test, some sample operations for tester are:-

a. Rotate and Permutate the position of devices to ensure

physiological/logical device permutations work for each device

b. Induce malfunctions in each devices , to see if the system properly handles the malfunction

c. Induce multiple device malfunctions to see how the system reacts

- These operation will help to reveal problems (defects) relating to hardware and software when hardware exchange, and the reconfiguration occur.

SECURITY TESTING

- **Security testing is a process used to reveal defects in the security mechanisms of a software system**
- Computer Software and data can be compromised by

i. Criminals, intent on doing damages, stealing data and information, causing denial of service , invading privacy

ii. Errors on the part of honest developers/ maintainers who modify, destroy or compromise data because of misinformation , misunderstanding , and/or lack of knowledge

Attacks can be random or systematic.

 Damage can be done through various means such as:-

a. Viruses

b.Trojan Horses

c. Trap Doors

d.Illicit channels

- The effect of security breaches could be extensive and can cause
- Loss of information
- Corruption of information
- Privacy violations
- Denial of service

●

Key areas of security testing

1. Password Checking:-

Test the password checker to insure that users will select a password that meets the condition descried in the password checker specification. Equivalence class partitioning and boundary value analysis based on the rules and conditions that specify a valid password can be used to design the tests.

2. Legal and Illegal Entry with password:-

Test for legal and illegal system/data access via legal and illegal passwords.

3. Password Expiration:-

If it is decided that password will expire after certain time period, tests should be designed to insure the expiration period is properly supported and that users can enter a new and appropriate password.

4. Encryption:-

Design test cases to evaluate the correctness of both encryption and decryption algorithms for systems where data/message are encoded

5. Browsing:-

Evaluate browsing privileges to insure that unauthorized browsing doesn't occur. Tester should attempt to browse illegally and observe system responses. They should determine what types of private information can be inferred by both legal and illegal browsing

6. Trap Doors:-

Identify any unprotected entries into the system that may allow access through unexpected channel (trap doors) .Design test cases that attempt to gain illegal entry and observe results. tester will need to support of designer and developers for this task

7. Viruses:-

Design test to insure that system virus checkers prevent or curtail entry of viruses into the system. Tester may attempt to infect the system with various viruses and observer the system response.

RECOVERY TESTING

- **Recovery testing is a type of nonfunctional testing technique performed in order to determine how quickly the system can recover after it has gone through system crash or hardware failure**
- **Recovery testing is especially important for transaction system**

  Eg:- on line banking software
- Beizer advises that tester focus on the following areas during recovery testing
  - ☞ ***Restart:-*** The current system state and transaction state are discarded The most recent checkpoint record retrieved an d the system initialized to the state in the checkpoint record. Tester must insure that all transaction have been reconstructed correctly and that all devices are in proper state. The system should then be able to begin to process new transaction
  - ☞ ***Switchover:-*** The ability of the system to switch to a anew processor must be tested .Switch over is the result of a command or detection of faulty processor by a monitor
- All transaction and processes must be carefully examined to detect:-
- Loss of transaction
- Merging of transaction
- Incorrect Transactions
- An unnecessary duplication of transaction

REGRESSION TESTING

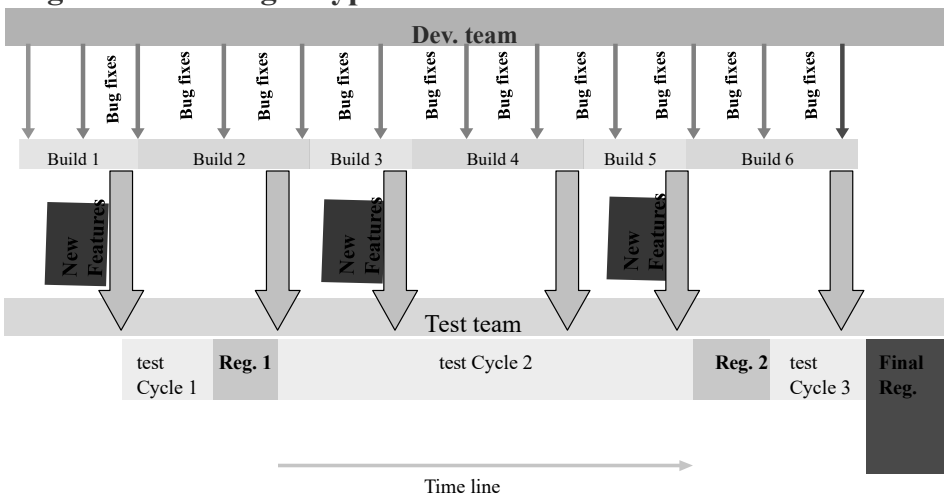
- **Regression testing is retesting of software that occur when changes are made to ensure that new version of the software has retained the capability of the old version and no new defects has been introduced due to the changes.**
- Regression Testing can occur at any level of test.
- Ex:- When unit tests are run the unit may pass a number of these tests until one of the test does reveal a defect. The unit is repaired and then retested with all the old test cases to ensure that the changes have not affected its functionality

Doctor: Congratulations! The stomach ulcer that was bothering you and preventing digestion is now completely cured!	*Patient:* That is fine doctor, but I have got such a bad mouth ulcer that I can't eat anything and hence there is nothing to digest!

- Regression testing is **selective re-testing** of the system with an objective to ensure that the bug fixes work and those bug fixes have not caused any un-intended effects in the system
- This testing is done to ensure that:
  - The bug-fixes work

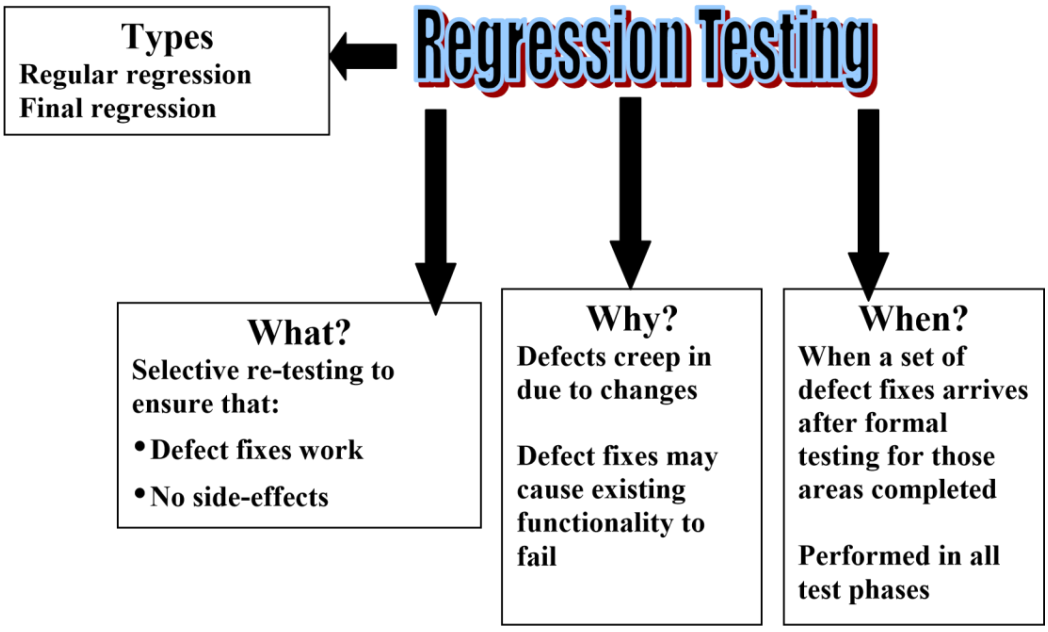
– The bug-fixes do not create any side-effects

Regression Testing – Types



Regression Testing – Types I. Final regression testing

- To ensure that "**the same build of the product that was tested reaches the customer**"
- Used to get a comfort feeling on the product prior to release **II. Regression testing**
- To validate the product builds between test cycles
- Unchanged build is recommended but not mandatory
- Used to get a comfort feeling on the bug fixes, and to carry on with next cycle of testing



ACCEPTANCE TESTING

- After the software has passed all the system tests and defect repairs have been made, the users take a more active role in the testing process.
- Developers/testers must keep in mind that the software is being developed to satisfy the users requirements, and no matter how elegant its design it will not be accepted by the users unless it helps them to achieve their goals as specified in the requirements.
- Alpha, beta, and acceptance tests allow users to evaluate the software in terms of their expectations and goals.
- **When software is being developed for a specific client, acceptance tests are carried out after system testing.**
- The acceptance tests must be planned carefully with input from the client/users. Acceptance test cases are based on requirements.
- The user manual is an additional source for test cases. System test cases may be reused. The software must run under real-world conditions on operational hardware and software. The software-under-test should be stressed.

- Acceptance tests are a very important milestone for the developers. At this time the clients will determine if the software meets their requirements. Contractual obligations can be satisfied if the client is satisfied with the software. Development organizations will often receive their final payment when acceptance tests have been passed.
- Acceptance tests must be prepared by the developers/testers.
- Clients should be provided with documents and other material to help them participate in the acceptance testing process, and to evaluate the results.
- **After acceptance testing the client will point out to the developers which requirement have/have not been satisfied.**
- **If the client is satisfied that the software is usable and reliable, and they give their approval, then the next step is to install the system at the client's site.**
- **If the client's site conditions are different from that of the developers, the developers must set up the system so that it can interface with client software and hardware. Retesting may have to be done to insure that the software works as required in the client's environment. This is called installation test.**

Acceptance tests are run to verify both functional and non functional requirements. **Acceptance Criteria**:

i. **Acceptance Criteria – Product Acceptance**: During the requirement phase, each requirement is associated with acceptance criteria. It is possible that one or more requirements may be mapped to form acceptance criteria. Whenever there are changes to requirements, the acceptance criteria are accordingly modified and maintained.

ii. **Acceptance Criteria – procedure acceptance:** Acceptance criteria can be defined based on the procedures followed for delivery. Some examples of acceptance criteria are as follows:

a. User, administration and troubleshooting documentation should be part of the release.

b. A minimum of 20 employees are trained on the product usage prior to deployment.

iii. **Acceptance Criteria – Service level agreements:** Service level agreements can become part of acceptance criteria. Service level agreements are generally part of a contract signed by the customer and product organization.

Selecting Test Cases for Acceptance Testing:

The test cases for acceptance testing are selected from the existing set of test cases from different phases of testing.

Some guidelines on what test cases can be included for acceptance testing are given below:

1. **End-to-end functionality verification**: Test cases that include the end-toend functionality of the product are taken up for acceptance testing. This ensures that all the business transactions are tested as a whole.
2. **Domain Tests**: Since acceptance tests focus on business scenarios the product domain tests are included. Test cases that reflect business domain knowledge are included.
3. **User Scenario Tests**: Acceptance tests reflect the real-life user scenario verification.
4. **Basic Sanity tests**: Tests that verify the basic existing behavior of the product are included. These tests ensure that the system performs the basic operations that it was intended to do.
5. **New Functionality**: When the product undergoes modifications or changes, the acceptance test cases focus on verifying the new features.

Executing acceptance tests:

- ✓ An acceptance test team comprises members who are involved in the dayto-day activities of the product usage or are familiar with such scenarios.
- ✓ The product management, support, and consulting team, who have good knowledge of the customers contribute to the acceptance testing.
- ✓ Test team members help the acceptance members to get the required test data, select and identify test cases, and analyze the acceptance test results.
- ✓ During test execution, the acceptance test team reports its progress regularly.
- ✓ The defect reports are generated on a periodic basis.
- ✓ Defect reported during acceptance tests could be of different priorities. Test teams help acceptance test team report defects.
- ✓ When the defect fixes point to scope or requirement changes, then it may either result in the extension of the release due to include the feature in the current release or get postponed to subsequent releases.

INTERNATIONALIZATION TESTING

Internationalization ($I_{18}n$) is used as an umbrella term to mean all activities that are required to make the software available for international market.

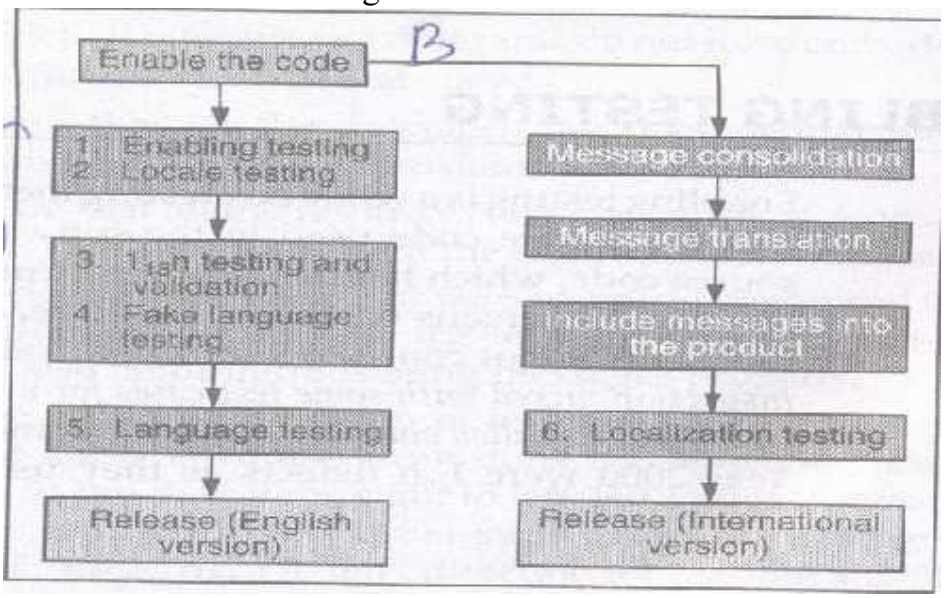
(18 is used to mean that there are 18 characters between "I" and the last "n" in the word Internationalization.

The testing that is done in various phases to ensure that all those activities are done right is called internationalization testing or $I_{18}n$ testing.

Test Phases for Internationalization Testing:

Testing for internationalization requires a clear understanding of all activities involved and their sequence.

The figure below depicts the various major activities involved in internationalization testing.



The testing for internationalization is done in multiple phases in the project life cycle.

Some important aspects of internationalization testing are:

a. Testing the code for how it handles input, strings, and sorting items

b. Display of messages for various languages and

c. Processing of messages for various languages and conventions

1. Enabling Testing:

Enabling testing is a white box testing methodology, which is done to ensure that the source code used in the software allows internationalization.

An activity of code review or code inspection mixed with some test cases for unit testing, with an objective to catch $I_{18}n$ defects is called enabling testing.

Enabling testing uses a checklist. Some items to be kept in the review checklist for enabling testing are as follows:

- Check the code for APIs/function calls that are not part of the $I_{18}n$.
- Check the code for hard-coded date, currency formats, ASCII code, or character constants.

- Ensure that adequate size is provided for buffers and variables to contain translated messages.

2. Locale Testing:

Once the code is verified for I18n and the enabling test is completed, the next step is to validate the effects of locale change in the product.

A local change affects date, currency format, and the display of items on screen, in dialog boxes and text.

Changing the different locales using the system settings or environment variables, and testing the software functionality, number, date, time and currency format is called locale testing.

Some of the items to be checked in locale testing are as follows:

1. All features that are applicable to $I_{18}n$ are tested with different locales of the software for which they are intended.
2. Function keys and help screens are tested with different applicable locales.
3. Date and time format are in line with the defined locale of the language.
4. Time zone information and daylight saving time calculations are consistent and correct.

3. Internationalization Validation:

$I_{18}n$ validation is different from $I_{18}n$ testing. $I_{18}n$ testing is the superset of all types of testing.

$I_{18}n$ validation is performed with the following objectives.

a. The software is tested for functionality with ASCII and European Characters.
b. The software handles string operations, sorting, sequence operations as per the language and characters selected.
c. The software display is consistent with characters which are non-ASCII in GUI and menus.
d. The software messages are handled properly.

A Checklist for the I18n validation includes the following:

- The functionality in all languages and locales are the same.
- Sorting and sequencing the items to be as per the conventions of language and locale.
- The software functions correctly with different languages.
- The documentation contains consistent documentation style,
  punctuations, and all language/locale conventions are followed for every target audience.

4. Fake Language Testing:

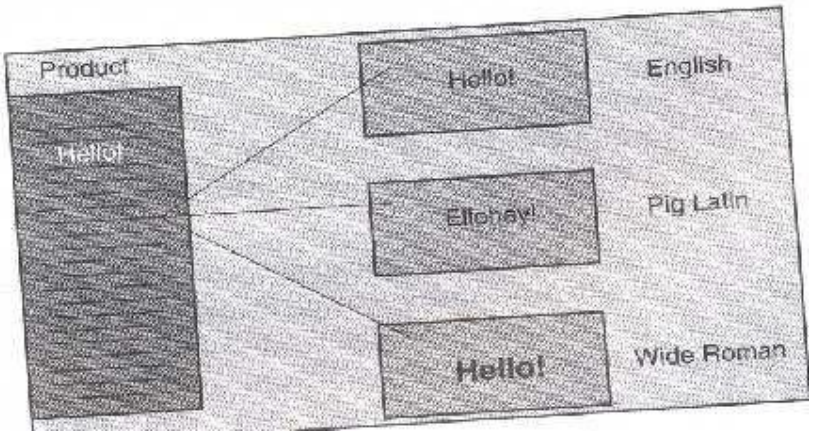
Fake language testing uses software translators to catch the translation and localization issues early.

This also ensures that switching between languages work properly and correct messages are picked from proper directories that have the translated messages.

Fake language testing helps in identifying the issues proactively before the product is localized.

The fake language translators use English-like target language, which are easy to understand and test.

This type of testing helps English testers to find the defects that may otherwise be found only by language experts during localization testing.



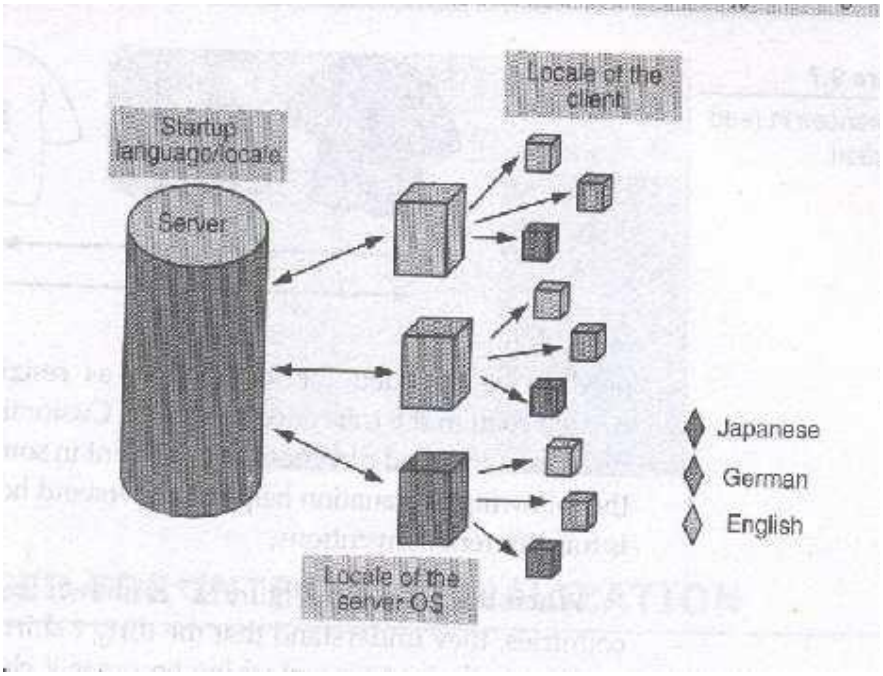
The following items in the checklist can be used for fake language testing.

a. Ensure software functionality is tested for atleast one of the European single-byte fake language.(Latin)
b. Ensure software functionality is tested for atleast one double-byte language (Roman)

5. Language Testing:

Language testing focuses on testing the English product with a global environment of products and services functioning in non English. It is the short form of "language compatibility testing".

This ensure that software created in English can work with platforms and environments that are English and Non-English.



The figure above illustrates the language testing and various combinations of locales that have to be tested in client-server architecture.

While testing, it is important to look for locale-related issues, as some of the defects that escaped from locale testing may show up in this testing.

Language testing should have the following checklists:

a. Check the functionality on English, one non-English and one double-byte language platform combination.
b. Check the performance of key functionality on different language platforms and across different machines connected in the network.

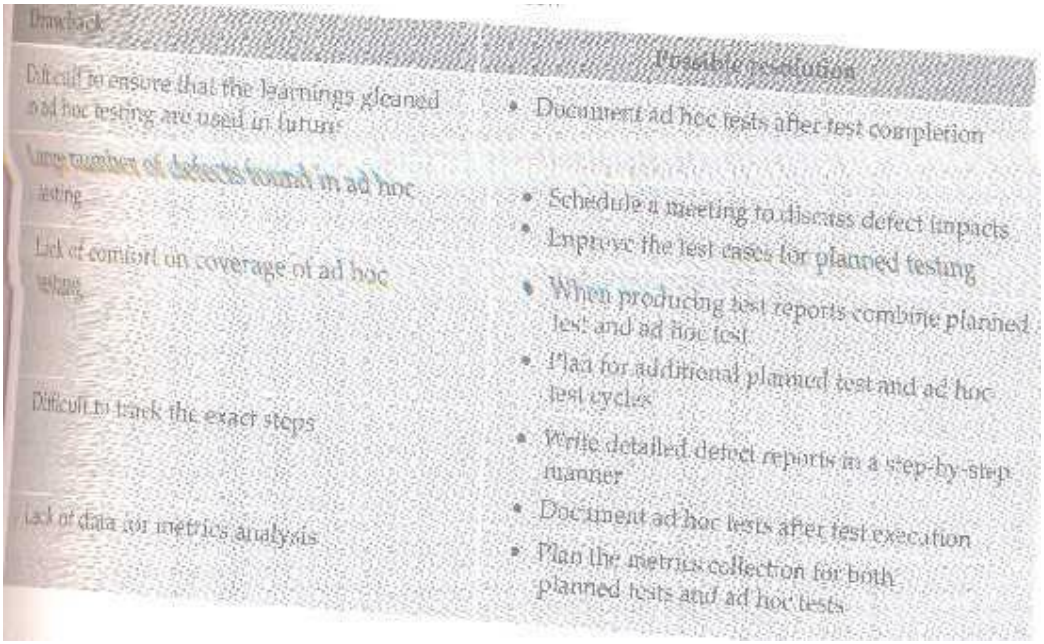
6. <u>Localization Testing:</u>

- ✓ When the software is approaching the release date, messages are consolidated into a separate file and sent to multilingual experts for translation.
- ✓ A set of build tools consolidates all the messages and other resources automatically and puts them in separate files.
- ✓ The multilingual experts may not be aware of the software or their target customers.
- ✓ Localization is a very expensive and labour-intensive process.
- ✓ While translating, not only the messages but also resources such as GUI screens, dialog boxes, icons and bitmaps need to be included for localization.
- ✓ Customization is important as scroll directions and read directions are different in some languages.
- ✓ The following checklist may help in doing localization testing.
    - All the messages, documents, pictures, screens are localized to reflect the native users and the conventions of the country, locale and language.
    - Sorting and case conversions are right as per language convention.
    - Font sizes and hot keys are working correctly in the translated messages, documents and screens.
    - Filtering and searching capabilities of the software work as per the language and locale conventions.

ADHOC TESTING

Testing done without using any formal testing technique is called adhoc testing.

- Adhoc testing is done to explore the undiscovered areas in the product by using intuition, previous experience in working with the product, expert knowledge of the platform or technology and experience of testing a similar product.
- Adhoc testing does not make use of any of the test case design techniques.



There are different types of adhoc testing. They are:

1. **Buddy Testing:**

A developer and tester working as buddies to help each other on testing and in understanding the specialization is called buddy testing.

- Two team members are identified as buddies.
- The buddies mutually help each other, with a common goal of identifying defects early and correcting them.
- A developer and tester become buddies.
- Buddies should not feel mutually threatened or get a feeling of insecurity during buddy testing.
- They stay close together to be able to follow the agreed plan.
- Buddy testing is normally done at the unit test phase, where there are both coding and testing activities.

2. Pair Testing:

Pair Testing is testing done by two testers working simultaneously on the same machine to find defects in the product.

- Two testers pair up to test a product's feature on the same machine.
- The objective of this testing is to maximize the exchange of ideas between the two testers.
- When one person is executing the tests, the other person takes notes.
- The other person suggests an idea or helps in providing additional perspectives.
- Pair testing can be done during any phase of testing.
- It encourages idea generation right from the requirements analysis phase, taking it forward to the design, coding and testing phases.
- Testers can pair together during the coding phase to generate various ideas to test the code and various components.

3. **Exploratory Testing:**

- Exploratory testing is a technique to find defects by exploring the product, covering more depth and breadth.
- It can be done during any phase of testing.
- Exploratory testers may execute their tests based on their past experiences in testing a similar product.
- Since there is large creative element to exploratory testing, similar test cases may result in different kinds of defects when done by two different individuals.
- **Exploratory test techniques are:**

a. Guesses

b. Architecture diagram, use cases

c. Past defects

d. Error Handling

e. Discussions

f. Questions and checklists.

4. **Iterative Testing:**

- In an iterative model, the requirements keep coming and the product is developed iteratively for each requirement. The testing associated with this process is called iterative testing.
- Regression tests may be repeated at least every alternative iteration so that the current functionality is preserved.
- Since iterative testing involves repetitive test execution of tests that were run for the previous iterations, it becomes a tiresome exercise for the testers.

5. **Agile and Extreme Testing:**

- Agile and extreme models take the processes to the extreme to ensure that customer requirements are met in timely manner.

- In this model, customers partner with the project teams to go step by step in bringing the project to completion in a phased manner. The customer becomes part of the project team so as to clarify any doubts/questions.
- Agile and extreme methodology emphasizes the involvement of the entire team, and their interaction with each other, to produce a workable software that can satisfy a given set of features.

USABILITY AND ACCESSIBILITY TESTING USABILITY TESTING:

The testing that validates the ease of use, speed, and aesthetics of the product from the users' point of view is called usability testing.

A right approach for usability is to test every artifact that impacts users – such as product binaries, documentation, messages, media-covering usage patterns through both graphical and command user interfaces as applicable.

Usability testing can be done is two phases:

1. Design validation Phase
2. Component and integration testing phase.

Usability design is verified through several means. Some of them are as follows:

a. Style sheets
b. Screen prototypes
c. Paper designs
d. Layout Design.

Table 12.1 Development and testing of client applications and web application.

Client application	Web application
Step 1: Design for functionality	Step 1: Design for user interface
Step 2: Perform coding for functionality	Step 2: Perform coding for user interface (prototype)
Step 3: Design for user interface	Step 3: Test user interface (Phase 1)
Step 4: Perform coding for user interface	Step 4: Design for functionality
Step 5: Integrate user interface with functionality	Step 5: Perform coding for functionality
Step 6: Test the user interface along with functionality (Phase 1 and 2)	Step 6: Integrate user interface with functionality
	Step 7: Test user interface along with functionality (Phase 2)

ACCESSIBILITY TESTING:

Verifying the product usability for physically challenged users is called accessibility testing.

Accessibility to the product can be provided by two means:

a. Making use of accessibility features provided by the underlying infrastructure called basic accessibility and

b. Providing accessibility in the product through standards and guidelines, called product accessibility.

Basic Accessibility:

- It is provided by the hardware and operating system. All the input and output devices of the computer and their accessibility options are categorized under basic accessibility.
- Some of the basic accessibility options are:

i. Keyboard accessibility
ii. Screen accessibility **Product Accessibility:**

- A good understanding of the basic accessibility features is needed while providing accessibility to the product.
- A product should do everything possible to ensure that the basic accessibility features are utilized by it.
- **Sample Requirements are:**

i. Providing detailed text equivalent for multimedia files ensures the captions feature is utilized by the product.
ii. Documents and fields should be organized so that they can be read without requiring a particular resolution of the screen, and templates.
iii. User interfaces should be designed so that all information conveyed with color is also available without color.
iv. Reduce flicker rate, speed of moving text; avoid flashes and blinking text.
v. Reduce physical movement requirements for the users when designing the interface and allow adequate time for user responses. The sample requirements given above are some examples to improve accessibility.

ALPHA AND BETA TESTING

ALPHA TESTING

If the software has been developed for the mass market, then testing it for individual clients/users is not practical in most cases. Very often this type of software undergoes two stages of acceptance test.

The first is called alpha test. This test takes place at the developer's site. A cross-section of potential users and members of the developer's organization are invited to use the software. Developers observe the users and note problems.

- Alpha testing is testing of an application when development is about to complete. Minor design changes can still be made as a result of alpha testing.
- Alpha testing is typically performed by a group that is independent of the design team, but still within the company, e.g. in-house software test engineers, or software QA engineers.
- Alpha testing is final testing before the software is released to the general public. It has two phases:
- In the **first phase** of alpha testing, the software is tested by in-house developers. They use either debugger software, or hardware-assisted debuggers. The goal is to catch bugs quickly.
- In the **second phase** of alpha testing, the software is handed over to the software QA staff, for additional testing in an environment that is similar to the intended use.

BETA TESTING

Beta test sends the software to a cross-section of users who install it and use it under real world working conditions. The users send records of problems with the software to the development organization where the defects are repaired sometimes in time for the current release.

The goal of beta testing is to place your application in the hands of real users outside of your own engineering team to discover any flaws or issues from the user's perspective that you would not want to have in your final, released version of the application.

Advantages of beta testing

- You have the opportunity to get your application into the hands of users prior to releasing it to the general public.
- Users can install, test your application, and send feedback to you during this beta testing period.
- Your beta testers can discover issues with your application that you may have not noticed, such as confusing application flow, and even crashes.
- Using the feedback you get from these users, you can fix problems before it is released to the general public.
- The more issues you fix that solve real user problems, the higher the quality of your application when you release it to the general public.
- Having a higher-quality application when you release to the general public will increase customer satisfaction.
- These users, who are early adopters of your application, will generate excitement about your application.

WEBSITE TESTING

Website Testing is checking your web application for potential bugs before its made live or before code is moved into the production environment.

Some or all of the following testing types may be performed depending on your web testing requirements.

1.Functionality Testing:

- This is used to check if your product is as per the specifications you intended for it as well as the functional requirements you charted out for it in your developmental documentation.
- Testing Activities Included:

Test all **links** in your webpages are working correctly and make sure there are no broken links.

It can be **carried out by testers** like you **or a small focus group** similar to the target audience of the web application.

Test the site **Navigation**:

Menus, buttons or Links to different pages on your site should be easily visible and consistent on all webpages **Test** the **Content**:

Content should be legible with no spelling or grammatical errors.

Images if present should contain an "alt" text

2.Interface Testing:

Three areas to be tested here are - Application, Web and Database Server **Application:** Test requests are sent correctly to the Database and output at the client side is displayed correctly. Errors if any must be caught by the application and must be only shown to the administrator and not the end user.

Web Server: Test Web server is handling all application requests without any service denial.

Database Server: Make sure queries sent to the database give expected results.

3.Database Testing:

Database is one critical component of your web application and stress must be laid to test it thoroughly. Testing activities will include Test if any errors are shown while executing queries

- Data Integrity is maintained while creating, updating or deleting data in database.
- Check response time of queries and fine tune them if necessary.
- Test data retrieved from your database is shown accurately in your web application

4.Compatibility testing.

Compatibility tests ensures that your web application displays correctly across different devices. This would include-

Browser Compatibility Test: Same website in different browsers will display differently. You need to test if your web application is being displayed correctly across browsers, JavaScript, AJAX and authentication is working fine. You may also check for Mobile Browser Compatibility.

5.Performance Testing:

This will ensure your site works under all loads.

6. Security testing:

Security Testing is vital for e-commerce website that store sensitive customer information like credit cards. Testing Activities will include-

- Test unauthorized access to secure pages should not be permitted
- Restricted files should not be downloadable without appropriate access

TESTING OO SYSTEMS

Testing is a continuous activity during software development. In objectoriented systems, testing encompasses three levels, namely, unit testing, subsystem testing, and system testing.

Unit Testing

In unit testing, the individual classes are tested.

Smallest testable unit is the encapsulated class

It is seen whether the class attributes are implemented as per design and whether the methods and the interfaces are error-free. Unit testing is the responsibility of the application engineer who implements the structure.

Subsystem Testing

This involves testing a particular module or a subsystem and is the

responsibility of the subsystem lead. It involves testing the associations within the subsystem as well as the interaction of the subsystem with the outside. Subsystem tests can be used as regression tests for each newly released version of the subsystem.

System Testing

System testing involves testing the system as a whole and is the responsibility of the quality-assurance team. The team often uses system tests as regression tests when assembling new releases.

Object-Oriented Testing Techniques

Grey Box Testing

The different types of test cases that can be designed for testing object-oriented programs are called grey box test cases. Some of the important types of grey box testing are:

State model based testing : This encompasses state coverage, state transition coverage, and state transition path coverage.

Use case based testing : Each scenario in each use case is tested.

Class diagram based testing : Each class, derived class, associations, and aggregations are tested.

Sequence diagram based testing : The methods in the messages in the sequence diagrams are tested.

Techniques for Subsystem Testing

The two main approaches of subsystem testing are:

Thread based testing : All classes that are needed to realize a single use case in a subsystem are integrated and tested.

Use based testing : The interfaces and services of the modules at each level of hierarchy are tested. Testing starts from the individual classes to the small modules comprising of classes, gradually to larger modules, and finally all the major subsystems.

Categories of System Testing

Alpha testing : This is carried out by the testing team within the organization that develops software.

Beta testing : This is carried out by select group of co-operating customers.

Acceptance testing : This is carried out by the customer before accepting the deliverables.

TESTING THE DOCUMENTATION

Documentation testing is a non-functional type of **software testing**.

- It is a type of **non-functional testing**.
- Any written or pictorial information describing, defining, specifying, reporting, or certifying activities, requirements, procedures, or results'. Documentation is as important to a product's success as the product itself. If the documentation is poor, non-existent, or wrong, it reflects on the quality of the product and the vendor.
- As per the IEEE Documentation describing plans for, or results of, the testing of a system or component, Types include test case specification, test incident report, test log, test plan, test procedure, test report. Hence the testing of all the above mentioned documents is known as documentation testing.
- This is one of the most cost effective approaches to testing. If the documentation is not right: there will be major and costly problems. The documentation can be tested in a number of different ways to many different degrees of complexity. These range from running the documents through a spelling and grammar checking device, to manually reviewing the documentation to remove any ambiguity or inconsistency.
- Documentation testing can start at the very beginning of the software process and hence save large amounts of money, since the earlier a **defect** is found the less it will cost to be fixed.