UNIT III
LEVELS OF TESTING

The Need for Levels 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

3.1 THE NEED FOR LEVELS OF TESTING

❖ Execution-based software testing, especially for large systems, is usually carried out at different levels.
❖ The software is tested at different levels. There are four levels of software testing, namely,
  ✓ **Unit Testing**
  ✓ **Integration Testing**
  ✓ **System Testing**
  ✓ **Acceptance Testing**

❖ At each level there are specific testing goals.
❖ At unit test a single component is tested. A principal goal is to detect functional and structural defects in the unit.
❖ At the integration level several components are tested as a group, and the tester investigates component interactions.
❖ At the system level the system as a whole is tested and a principle goal is to evaluate attributes such as usability, reliability, and performance.
❖ The major testing levels for both types of system are similar. The nature of the code that results from each developmental approach demands different testing strategies, to identify individual components, and to assemble them into subsystems.
❖ For both types of systems the testing process begins with the smallest units or components to identify functional and structural defects.
❖ Testers check for defects and adherence to specifications. Proper interaction at the component interfaces is of special interest at the integration level.
❖ White box tests are used to reveal defects in control and data flow between the integrated modules.
❖ System test begins when all of the components have been integrated successfully. It usually requires the bulk of testing resources.
❖ At the system level the tester looks for defects, but the focus is on evaluating performance, usability, reliability, and other quality-related requirements.
❖ If the system is being custom made for an individual client then the next step following system test is acceptance test. This is a very important testing stage for the developers.

- During acceptance test the development organization must show that the software meets all of the client's requirements.
- A successful acceptance test provides a good opportunity for developers to request recommendation letters from the client.
- Software developed for the mass market goes through a series of tests called alpha and beta tests.
- Alpha tests bring potential users to the developer's site to use the software. Developers note any problems.
- Beta tests send the software out to potential users who use it under real-world conditions and report defects to the developing organization. Implementing all of these levels of testing require a large investment in time and organizational resources.

3.1.1 Levels 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.
- There are two major approaches to system development
  - ✓ Bottom-up,
  - ✓ Top-down.
- These approaches are supported by two major types of programming languages
  - ✓ Procedure-oriented,
  - ✓ Object-oriented.
- 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 in order to reveal defects. Object-oriented systems are viewed as being composed of active data along with allowed operations on that data.

Procedural System

- The lowest level of abstraction is described as a function or a procedure that performs some simple task. The next higher level of abstraction is a group of procedures (or functions) that call one another and implement a major system requirement. These are called subsystems. Combining subsystems finally produces the system as a whole, which is the highest level of abstraction.

Object-Oriented System

- The lowest level is viewed as the method or member function. The next highest level is viewed as the class that encapsulates data and methods that operate on the data. The system level is a combination of all the clusters and any auxiliary code needed to run the system. Object-oriented development key beneficial features are encapsulation, inheritance, and polymorphism.

3.2 UNIT TEST

- A unit is the smallest possible testable software component. An unit in procedure-oriented software system:
  - ✓ Performs a single cohesive function;
  - ✓ Can be compiled separately;
  - ✓ Is a task in a work breakdown structure;
  - ✓ Contains code that can fit on a single page or screen.

- In object-oriented systems both the method and the class/object have been suggested by researchers as the choice for a unit.
- A unit may also be a small-sized COTS component purchased from an outside vendor that is undergoing evaluation by the purchaser, or a simple module retrieved from an in-house reuse library.
- Since the software component being tested is relatively small in size and simple in function, it is easier to design, execute, record, and analyse test results.
- If a defect is revealed by the tests it is easier to locate and repair since only the one unit is under consideration.

The Need for Preparation

- The principal goal for unit testing is insure that each individual software unit is functioning according to its specification.
- Good testing practice is based on unit tests that are planned and public.
- Planning includes designing tests 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 (someone other than the developer) and the test results and defects found should be recorded as a part of the unit history.
- 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.

3.3 UNIT TESTING PLANNING

- A general unit test plan should be prepared. It may be prepared as a component of the master test plan or as a stand-alone plan.
- It should be developed in conjunction with the master test plan and the project plan for each project
- Documents that provide inputs for the unit test plan are the project plan, as well the requirements, specification, and design documents that describe the target units.
- The phases allow a steady evolution of the unit test plan as more information becomes available.

Phase 1: Describe Unit Test Approach and Risks

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

- ✓ Identifies test risks;
- ✓ Describes techniques to be used for designing the test cases for the units;
- ✓ Describes techniques to be used for data validation and recording of test results;
- ✓ Describes the requirements for test harnesses and other software that interfaces with the units to be tested, for example, any special objects needed for testing object-oriented units.

Phase 2: Identify Unit Features to be tested

- ❖ 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 requirements, states, and state transitions, control structures, messages, and data flow patterns.

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.

3.4 DESIGNING THE UNIT TEST

- ❖ Part of the preparation work for unit test involves unit test design. It is important to specify the test cases and, the test procedures. Test case data should be tabulated for ease of use, and reuse.
- ❖ As part of the unit test design process, developers/testers should also describe the relationships between the tests.
- ❖ Test suites can be defined that bind 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 past projects.
- ❖ Test case design at the unit level can be based on use of the black and white box test design strategies.
- ❖ Both of these approaches are useful for designing test cases for functions and procedures.
- ❖ They are also useful for designing tests for the individual methods (member functions) contained in a class.
- ❖ White-Box method can be used because the size is small. This approach gives the tester the opportunity to exercise logic structures and/or data flow sequences, or to use mutation analysis, all with the goal of evaluating the structural integrity of the unit.
- ❖ Some black box–based testing is also done at unit level; however, the bulk of black box testing is usually done at the integration and system levels and beyond.
- ❖ In the case of a smaller-sized COTS component selected for unit testing, a black box test design approach may be the only option.
- ❖ It should be mentioned that for units that perform mission/safely/business critical functions, it is often useful and prudent to design stress, security, and performance tests at the unit level if possible.
- ❖ This approach may prevent larger scale failures at higher levels of test.

3.5 THE TEST HARNESS

- ❖ The auxiliary code developed to support testing of units and components is called a test harness.
- ❖ The harness consists 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.
- For example, a driver could have the following options and combinations of options:
  - ✓ Call the target unit
  - ✓ Do 1, and pass inputs parameters from a table
  - ✓ Do 1, 2, and display parameters
  - ✓ Do 1, 2, 3 and display results (output parameters)
- The stubs could also exhibit different levels of functionality. For example a stub could:
  - ✓ Display a message that it has been called by the target unit
  - ✓ Do 1, and display any input parameters passed from the target unit
  - ✓ Do 1, 2, and pass back a result from a table
  - ✓ Do 1, 2, 3, and display result from table

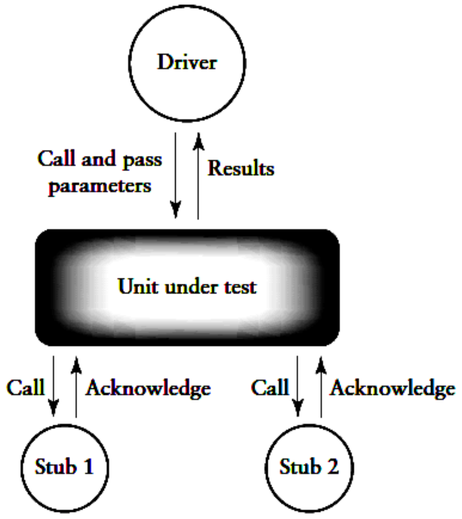


Figure 3.1 : Test Harness

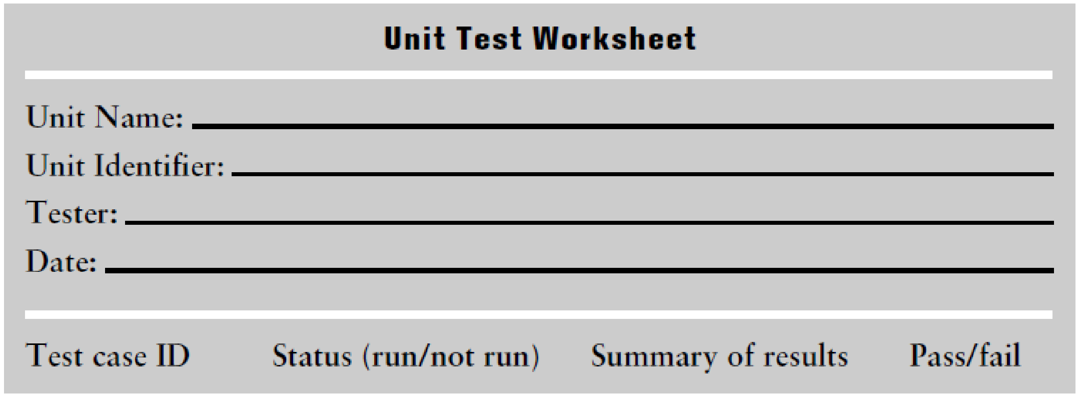
- Drivers and stubs as shown in the figure are developed as procedures and functions for traditional imperative-language based systems.
- For object-oriented systems, developing drivers and stubs often means the design and implementation of special classes to perform the required testing tasks.
- The test harness itself may be a hierarchy of classes. The test planner must realize that, the higher the degree of functionally for the harness, the more resources it will require to design, implement, and test.
- Developers/testers will have to decide depending on the nature of the code under test, just how complex the test harness needs to be.

3.6 RUNNING THE UNIT TEST AND RECORDING RESULT

- Unit tests can begin when
  - ✓ The units becomes available from the developers,

- ✓ The test cases have been designed and reviewed,
- ✓ The test harness, and any other supplemental supporting tools, is available.

- ❖ The testers then proceed to run the tests and record results. The status of the test efforts for a unit, and a summary of the test results, could be recorded in a simple format as shown below.



Unit Test Worksheet

Unit Name: ____

Unit Identifier: ____

Tester: ____

Date: ____

Test case ID	Status (run/not run)	Summary of results	Pass/fail

Figure 3.2: Summary of Worksheet for Unit Test results

- ❖ Differences from expected behaviour should be described in detail. During testing the tester may determine that additional tests are required.
- ❖ The test set will have to be augmented and the test plan documents should reflect these changes.
- ❖ 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 (the code).
- ❖ Other likely causes that need to be carefully investigated by the tester are the following:
  - ✓ A fault in the test case specification
  - ✓ A fault in test procedure execution
  - ✓ A fault in the test environment
  - ✓ 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.
- ❖ Under some circumstances a unit may be given a conditional acceptance for integration test.
- ❖ This may occur when the unit fails some tests, but the impact of the failure is not significant.
- ❖ When testing of the units is complete, a test summary report should be prepared.
- ❖ This is a valuable document for the groups responsible for integration and system tests.
- ❖ It is also a valuable component of the project history.
- ❖ Finally, the tester should insure that the test cases, test procedures, and test harnesses are preserved for future reuse.

3.7 INTEGRATION TEST

- ❖ Integration test for procedural code has two major goals:

  - ✓ To detect defects that occur on the interfaces of units;
  - ✓ To assemble the individual units into working subsystems and finally a complete system that is ready for system test.

- In unit test the testers attempt to detect defects that are related to the functionality and structure of the unit.
- The interfaces are more adequately tested during integration test when each unit is finally connected to a full and working implementation of those units it calls, and those that call it.
- As a consequence of this assembly or integration process, software subsystems and finally a completed system is put together during the integration test.
- The completed system is then ready for system testing.
- A unit tested in isolation may not have been tested adequately for the situation where it is combined with other modules.
- Integration testing works best as an iterative process in procedural oriented system.
- One unit at a time is integrated into a set of previously integrated modules which have passed a set of integration tests.
- The interfaces and functionally of the new unit in combination with the previously integrated units is tested.
- Integrating one unit at a time helps the testers in several ways.
- It keeps the number of new interfaces to be examined small, so tests can focus on these interfaces only.
- Experienced testers know that many defects occur at module interfaces.
- Another advantage is that the massive failures that often occur when multiple units are integrated at once are avoided.
- The integration process in object-oriented systems is driven by assembly of the classes into cooperating groups.
- The cooperating groups of classes are tested as a whole and then combined into higher-level groups.

3.8 DESIGNING INTEGRATION

- Integration tests for procedural software can be designed using a black or white box approach.
- Both are recommended. Some unit tests can be reused.
- The tester needs to insure the parameters are of the correct type and in the correct order.
- The tester must also insure that once the parameters are passed to a routine they are used correctly.
- Procedure_b is being integrated with Procedure_a.
- Procedure_a calls Procedure_b with two input parameters in3, in4.
- Procedure_b uses those parameters and then returns a value for the output parameter out1.
- Terms such as lhs and rhs could be any variable or expression.
- The parameters 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.

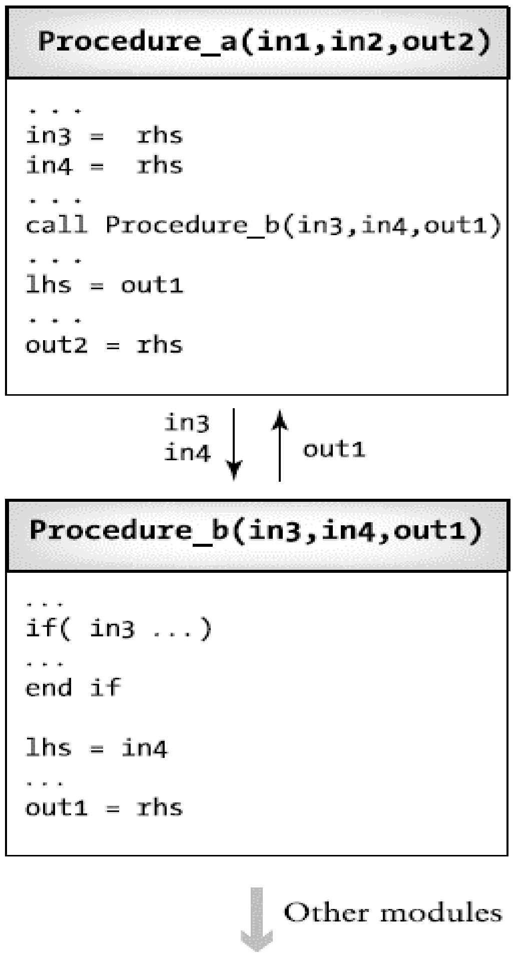


Figure 3.3: Integration of two procedures

- For conventional systems, input/output parameters and calling relationships will appear in a structure chart built during detailed design.
- Testers must insure that test cases are designed so that all modules in the structure chart are called at least once, and all called modules are called by every caller.
- Some black box tests 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 have to be designed to cover their functionality and adherence to performance and other requirements.
- Integration testing of clusters of classes also involves building test harnesses which in this case are special classes of objects built especially for testing.
- Whereas in class testing we evaluated intra-class method interactions, at the cluster level we test interclass method interaction as well.
- We want to insure that messages are being passed properly to interfacing objects, object state transitions are correct when specific events occur, and that the clusters are performing their required functions.

3.9 TESTS: INTEGRATION TEST PLANNING

- Integration test must be planned. Planning can begin when high-level design is complete so that the system architecture is defined.
- Other documents relevant to integration test planning are the requirements document, the user manual, and usage scenarios.
- These documents contain structure charts, state charts, data dictionaries, cross-reference tables, module interface descriptions, data flow descriptions, messages and event descriptions, all necessary to plan integration tests.
- For object-oriented systems a working definition of a cluster or similar construct must be described, and relevant test cases must be specified.
- In addition, testing resources and schedules for integration should be included in the test plan.
- The plan must include the following items
  - ✓ Clusters this cluster is dependent on;
  - ✓ A natural language description of the functionality of the cluster to be tested;
  - ✓ List of classes in the cluster;
  - ✓ A set of cluster test cases.

- One of the goals of integration test is to build working subsystems, and then combine these into the system as a whole.
- When planning for integration test the planner selects subsystems to build based upon the requirements and user needs.
- Very often subsystems selected for integration are prioritized. Developers may want to show clients that certain key subsystems have been assembled and are minimally functional.

3.10 SCENARIO TESTING

- Scenario testing is defined as a set of realistic user activities that are used for evaluating the product. It is also defined as the testing involving customer scenario
- There are two methods to evolve scenarios
  1. System scenarios
  2. Use case scenarios

System Scenarios

System scenario is a method whereby 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 that may be executed by an end user

Life cycle/state transition:

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

Deployment / Implementation:

- ❖ Stories 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:

- ❖ Visualize how a product/software will be applied to different verticals and create a set of activities as scenarios to address specific vertical businesses.

Battle ground:

- ❖ Create some scenarios to justify that "the product works" and some scenarios to "try and break the system" to justify "the product does not work"

- ❖ The set of scenarios developed will be more effective if the majority of the approaches mentioned above are used in combination, not in isolation.
- ❖ Scenario should not be a set of disjoined activities which have no relation to each other. Any activity in scenario is always a continuation of the previous activity and depends on or is impacted by results of previous activities.
- ❖ Effective scenarios will have a combination of current customer implementation, foreseeing future use of product, and developing ad hoc test cases. Considering only one aspect would make scenarios ineffective.

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.
- ❖ A use case can involve several roles or class of users who typically perform different activities based on the role.
- ❖ There are some activities that are common across roles and these are some activities that are very specific and can be performed only by the user belonging to a particular role.
- ❖ Use case scenario term the users with different roles as "actors" what the product should do for particular activity is termed as "system behaviour".
- ❖ Users with a specific role to interact between the actors and the system are called `agent`.

Example Withdrawing cash from a bank

- ❖ A customer fills up a check 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 the actor, the clerk the agent, and the response given by the customer by the computer, which gives the balance in the account, is called the system response.
- ❖ The actor [who is the customer] need not know what the official is doing and what command he is using to interact with the computer. The actor is only concerned about getting the cash.
- ❖ The agent [who is the official] is not concerned about the logic of how the computer works.
- ❖ He or she is only interested in knowing from the computer whether he or she can give cash or not.
- ❖ However, the system behaviour [computer logic] needs to be tested before applying the sequence of agent activities and actor activities.

3.11 DEFECT BASH ELIMINATION

- Defect is an ad hoc testing where people performing different roles in an organization test the product together at the same time.
- The testing by all the participants during defect bash is not based on written test cases. What is to tested is left to an individual`s decision and creativity.
- Defect bash brings together plenty of good.
- Practices that are popular in testing industry are
    1. Enabling people “cross boundaries and test beyond assigned areas”.
    2. Bringing different people performing different roles together in the organization for testing – “Testing is not for tester alone”
    3. Letting everyone in the 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 same”.
    6. Let testing does not wait for lack of time taken for documentation- “Does testing wait till all documentation is done?”
    7. Enabling people to say “system works” as well as enabling them to “break the system”- “Testing is not to conclude the system works or does not work”.
- Even though it is said that defect bash is an ad hoc testing, not all activities of defect bash are unplanned.
- All the activities in the defect bash are planned activities, except for what to be tested. It involves several steps.

Step 1: Choosing the frequency and duration of defect bash.

Step 2: Selecting the right product build.

Step 3: communicating the objective of each defect bash to everyone.

Step 4: Setting up and monitoring the lab for defect bash.

Step 5: Taking actions and fixing issues.

Step 6: Optimizing the effort involved in defect bash.

3.12 SYSTEM TESTING

- When integration tests are completed, a software system has been assembled and its major subsystems have been tested.
- At this point the developers/ testers begin to test it as a whole. System test planning should begin at the requirements phase with the development of a master test plan and requirements-based (black box) tests.
- 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 a large amount of resources. The goal is to ensure that the system performs according to its requirements.
- System test evaluates both functional behaviour and quality requirements such as reliability, usability, performance and security.
- This phase of testing is especially useful for detecting external hardware and software interface defects, for example, those causing race conditions, and deadlocks, problems with interrupts and exception handling, and ineffective memory usage.
- System test often requires many resources, special laboratory equipment, and long test times; it is usually performed by a team of testers.
- The best scenario is for the team to be part of an independent testing group.

3.13 TYPES OF SYSTEM TESTING

There are several types of system testing,

1. Functional testing
2. Performance testing
3. Stress testing
4. Configuration testing
5. Security testing
6. Recovery testing

- Not all software systems need to undergo all the types of system testing. Test planners need to decide on the type of tests applicable to a particular software system. Decisions depend on the characteristics of the system and the available test resources.
- During system test the testers can repeat these tests and design additional tests for the system as a whole.
- The repeated tests can in some cases be considered regression tests since there most probably have been changes made to the requirements and to the system itself since the initiation of the project. 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 series of inputs that simulates a group of transactions.

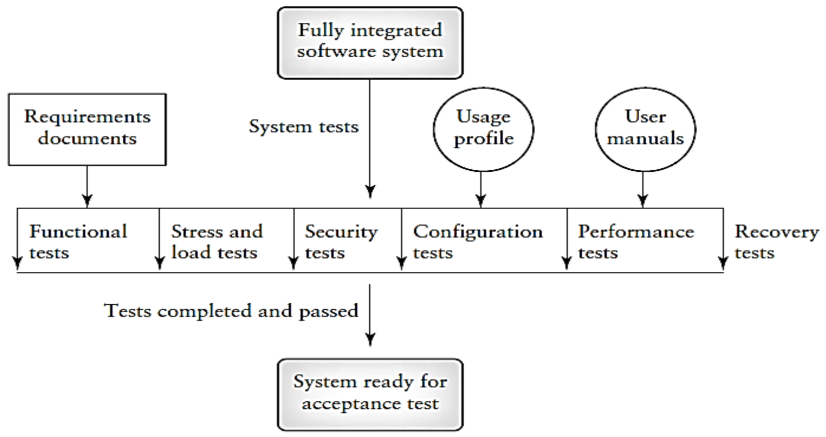


Figure 3.4: Types of System Tests

1. Functional Testing

- ❖ Functional Tests may overlap with acceptance tests. Functional tests at the 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.
- ❖ Functional tests are black box in nature. The focus is on the inputs and proper outputs for each function.
- ❖ Improper and illegal inputs must also be handled by the system. The tests should focus on the following goals.

    1. All types or classes of legal inputs must be accepted by the software.
    2. All classes of illegal inputs must be rejected (however, the system should remain available).
    3. All possible classes of system output must exercise and examined.
    4. All effective system states and state transitions must be exercised and examined.
    5. All functions must be exercised.

2. Performance Testing

- ❖ An examination of a requirements document shows that there are two major types of requirements:

    a) **Functional requirements.** Users describe what functions the software should perform.

    b) **Quality requirements.** There are non-functional in nature but describe quality levels expected for the software.

- ❖ The goal of system performance tests is to see if the software meets the performance requirements.
- ❖ Testers also learn from performance test whether there are any hardware or software factors that impact on the system's performance.
- ❖ Resources for performance testing must be allocated in the system test plan.
- ❖ Examples of such resources are shown in the below figure.

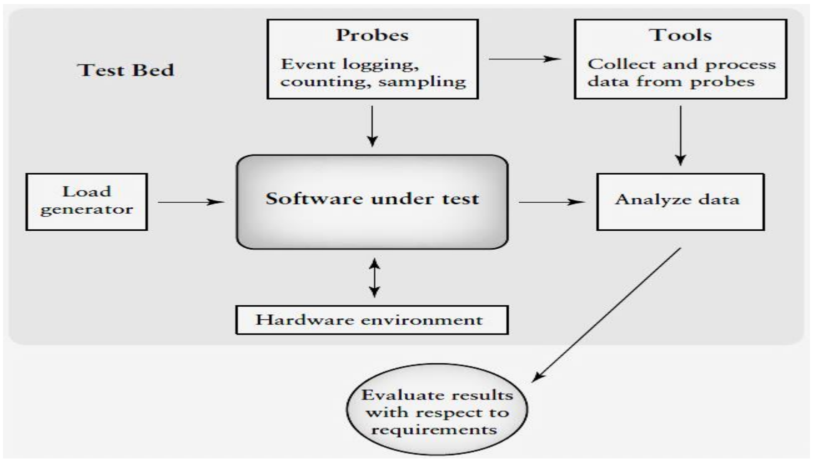

Test Bed
Probes
Event logging, counting, sampling
Tools
Collect and process data from probes
Load generator
Software under test
Analyze data
Hardware environment
Evaluate results with respect to requirements

Figure 3.5: Resources needed for Performance Test

3. Stress Testing

- When a system is tested with a load that causes it to allocate its resources in maximum amounts, this is called stress testing.
- 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 service.
- Stress testing is important from the user/client point of view.
- When system operates correctly under conditions of stress then clients have confidence that the software can perform as required.

4. Configuration Testing

- Configuration testing allows developers/testers to evaluate system performance and availability when hardware exchanges and reconfigurations occur.
- Configuration testing also requires many resources including the multiple hardware devices used for the tests.
- If a system does not have specific requirements for device configuration changes then large-scale configuration testing is not essential.
- Configuration testing has the following objectives

  - ✓ Show that all the configuration changing commands and menus work properly.
  - ✓ Show that all interchangeable devices are really interchangeable, and that they each enter the proper states for the specified conditions.
  - ✓ Show that the systems' performance level is maintained when devices are interchanged, or when they fail.

5. Security Testing

- Designing and testing software systems to insure that they are safe and secure is a big issue facing software developers and test specialists.
- Security testing is a testing technique to determine if an information system protects data and maintains functionality as intended. It also aims at verifying 6 basic principles as listed below:
  - ✓ Confidentiality
  - ✓ Integrity
  - ✓ Authentication
  - ✓ Authorization
  - ✓ Availability
  - ✓ Non-repudiation
- Security testing evaluates system characteristics that relate to the availability, integrity, and confidentially of system data and services.
- Users/clients should be encouraged to make sure their security needs are clearly known at requirements time, so that security issues can be addressed by designers and testers.
- Both criminal behaviour and errors that do damage can be perpetuated by those inside and outside of an organization.
- Attacks can be random or systematic. Damage can be done through various means such as
  - ✓ Viruses
  - ✓ Trojan horses
  - ✓ Trap doors
  - ✓ Illicit channels
- The effects of security breaches could be extensive and can cause:
  - ✓ Loss of information
  - ✓ Corruption of information
  - ✓ Misinformation
  - ✓ Privacy violations
  - ✓ Denial of service
- **FOCUS AREAS** : There are four main focus areas to be considered in security testing (Especially for web sites/applications):
  - ✓ **Network security:** This involves looking for vulnerabilities in the network infrastructure (resources and policies).
  - ✓ **System software security:** This involves assessing weaknesses in the various software (operating system, database system, and other software) the application depends on.
  - ✓ **Client-side application security:** This deals with ensuring that the client (browser or any such tool) cannot be manipulated.
  - ✓ **Server-side application security:** This involves making sure that the server code and its technologies are robust enough to fend off any intrusion.
- There are seven main types of security testing as per Open Source Security Testing methodology manual. They are explained as follows:

- ✓ **Vulnerability Scanning**: This is done through automated software to scan a system against known vulnerability signatures.
- ✓ **Security Scanning:** It involves identifying network and system weaknesses, and later provides solutions for reducing these risks. This scanning can be performed for both Manual and Automated scanning.
- ✓ **Penetration testing**: This kind of testing simulates an attack from malicious hacker. This testing involves analysis of a particular system to check for potential vulnerabilities to an external hacking attempt.
- ✓ **Risk Assessment:** This testing involves analysis of security risks observed in the organization. Risks are classified as Low, Medium and High. This testing recommends controls and measures to reduce the risk.
- ✓ **Security Auditing:** This is internal inspection of Applications and Operating systems for security flaws. Audit can also be done via line by line inspection of code
- ✓ **Ethical hacking:** It's hacking Organization Software systems. Unlike malicious hackers, who steal for their own gains, the intent is to expose security flaws in the system.
- ✓ **Posture Assessment:** This combines Security scanning, Ethical Hacking and Risk Assessments to show an overall security posture of an organization.

6. Recovery Testing

- ❖ Recovery testing subjects a system to losses of resources in order to determine if it can recover properly from these losses.
- ❖ This type of testing is especially important for transaction systems.
- ❖ Tests would determine if the system could return to a well-known state, and that no transactions have been compromised. Systems with automated recovery are designed for this purpose.
- ❖ Testers focus on the following areas during recovery testing,
  1. **Restart**. The current system state and transaction states are discarded.
  2. **Switchover**. The ability of the system to switch to a new processor must be tested.

3.14 ACCEPTANCE TESTING

Acceptance testing, a testing technique performed to determine whether or not the software system has met the requirement specifications.

- ❖ The main purpose of this test is to evaluate the system's compliance with the business requirements and verify if it is has met the required criteria for delivery to end users.
- ❖ Acceptance Testing is performed after System Testing and before making the system available for actual use.
- ❖ There are various forms of acceptance testing:
  - ✓ User acceptance Testing
  - ✓ Business acceptance Testing
  - ✓ Alpha Testing
  - ✓ Beta Testing

Who performs it?

- **Internal Acceptance Testing** (Also known as **Alpha Testing**) is performed by members of the organization that developed the software but who are not directly involved in the project (Development or Testing). Usually, it is the members of Product Management, Sales and/or Customer Support.

- **External Acceptance Testing** is performed by people who are not employees of the organization that developed the software.

  - ✓ **Customer Acceptance Testing** is performed by the customers of the organization that developed the software. They are the ones who asked the organization to develop the software.

  - ✓ **User Acceptance Testing** (Also known as **Beta Testing**) is performed by the end users of the software. They can be the customers themselves or the customers' customers.

- The acceptance test cases are executed against the test data or using an acceptance test script and then the results are compared with the expected ones.
- The acceptance test activities are designed to reach at one of the conclusions:
  - ✓ Accept the system as delivered
  - ✓ Accept the system after the requested modifications have been made
  - ✓ Do not accept the system

Usually, Black Box Testing method is used in Acceptance Testing. Testing does not normally follow a strict procedure and is not scripted but is rather ad-hoc.

3.15 PERFORMANCE TESTING

Performance is a basic requirement for any product and is fast becoming a subject of great interest in the testing community.

Factors Governing Performance Testing

Many factors govern the performance testing. They are:

1. Throughput
2. Response time
3. Latency
4. Tuning
5. Benchmarking
6. Capacity planning

1. Throughput

- The capability of a system or the product in handling multiple transactions is determined by a factor called throughput.
- Throughput represents the number of requests/business transactions processed by the product in specified time duration.

2. Response Time

- ❖ Response time can be defined as the delay between the point of request and the first response from the product.

3. Latency

- ❖ Latency is a delay caused by the application, operating system, and by the environment that are calculated separately.

4. Tuning

- ❖ Tuning is a procedure by which the product performance is enhanced by setting different values to the parameters of the product, operating system, and other components.
- ❖ Tuning improves the product performance without having to touch the source code of the product.

5. Benchmarking

- ❖ Comparison of competitive product is called benchmarking.
- ❖ No tow products are the same in features, cost and functionality. Hence, it is not easy to decide which parameters must be compared across two products.

6. Capacity Planning

- ❖ The exercise to find out what resources and configurations are needed is called capacity planning.
- ❖ The purpose of a capacity planning exercise is to help customers plan for the set off hardware and software resources prior to installation or upgrade of the product.

Methodology for Performance Testing

- ❖ Performance testing is complex and expensive due to large resource requirements and the time it takes.
- ❖ Performance test is ambiguous because of the different people who are performing the various roles having different expectations.
- ❖ If defect found in performance change, then it may require design and architecture change.
- ❖ A methodology for performance testing involves the following steps.
    1. Collecting requirements
    2. Writing test cases
    3. Automating performance test cases
    4. Executing performance test cases
    5. Analyzing performance test cases
    6. Performance tuning
    7. Performance benchmarking
    8. Capacity planning

1. Collecting Requirements

- ❖ Performance testing needs elaborate documentation and environment setup and the expected results may not well known in advance.
- ❖ Challenges in collecting requirements for performance testing are
    1. The performance testing requirements should be testable-not all features/functionality can be tested.
    2. The performance testing requirements needs to clearly state what factors needs to be measured and improved.
    3. Performance testing requirements needs to be associated with the actual number or percentage of improvement that is denied.

- ❖ Sources for deriving performance requirements are
  1. Performance compared to the previous release of the same product.
  2. Performance compared to the competitive product.
  3. Performance compared to absolute numbers derived from actual need.
  4. Performance numbers derived from architecture and design.
- ❖ Two types of requirements employed in performance testing are
  1. Generic requirement
  2. Specific requirements

Generic Requirements

- ❖ Generic requirements are common across all products in the product domain area. Examples are Time taken to load a page, initial response when a mouse is clicked, and time taken to navigate between screens.

Specific Requirements

- ❖ Specific requirements are those that depend on implementation for a particular product and differ from one product to another in a given domain. Eg.The time taken to withdraw cash in an ATM.
- ❖ The performance values that are in acceptable limits when the load increases are denoted by a term called "graceful performance degradation".

2. Writing Test Cases

- ❖ A test case for performance testing should have
  1. List of operations or business transactions to be tested.
  2. Steps for executing those operations/transactions.
  3. List of products OS parameters that impact the performance testing, and their values.
  4. Loading pattern.
  5. Resource and their configuration [eg. Hardware]
  6. The expected results. [eg. Latency]
  7. The product versions/competitive products to be compared with and related information such as their corresponding fields.
- ❖ Performance test cases are repetitive in nature All test cases are part of performance testing have to be assigned different priorities so that high priority test cases can be completed before others.

3. Automating Performance Test Cases

- ❖ Performance testing is repetitive. Performance test cases cannot be effective without automation. The result of performance testing needs to be accurate.
- ❖ Performance testing takes in to account several factors. It will be difficult to remember all these and use them if the tests are done manually.
- ❖ The analysis of performance results and failures needs to take in to account related information such as resource utilization, log files, trace files, and so on that are collected at regular intervals.
- ❖ It is impossible to do this testing and perform the book-keeping of all related information and analysis manually. End-to-end automation is required for performance testing.

4. Executing Performance Test Cases

- ❖ Performance testing involves less effort for execution but more effort for planning, data collection and analysis.
- ❖ While executing performance tests, the following data are collected.
    1. Start and end time of test case execution.
    2. Log and trace/audit files of the product and operating system.
    3. Utilization of resources on a periodic basis.
    4. Configuration of all environmental factors [hardware, software and other components].
    5. The response time, throughput, latency and so on as specified in the test case documentation at regular interval

Scenario Testing

- ❖ A set of transaction/operations that are usually performed by the user forms the scenario for performance testing.
- ❖ His particular testing is done to ensure whether the mix of operations/transaction concurrently by different users/machine meets the performance criteria.

Configuration Performance Test

- ❖ What performance a product delivers for different configurations of hardware and network setup, is another aspect that needs to be included during execution.
- ❖ Configuration performance test ensures that the performance of the product is compatible with different hardware, utilizing the special nature of those configurations and yielding the best performance possible.

5. Analyzing the Performance Test Results

- ❖ Analyzing the performance test results require multidimensional thinking, product knowledge, analytical thinking, and statistical background are essential.
- ❖ Before analyzing the data, some calculations of data and organization of the data are required.
    1. Calculating the mean of the performance test result data.
    2. Calculating the standard deviation.
    3. Removing the noise and re-plotting and re-calculating the mean and standard deviation.
    4. The data coming from the cache need to be differentiated from the data that gets processed by the product, and presented.
    5. Differentiating the performance data when the resources are available completely as against when some background activities were going on.
- ❖ When there are set of performance numbers that came from multiple runs of the same test, there could be situations where in a few of the iterations, some errors were committed by the scripts, software, or a human.
- ❖ The process of removing some unwanted values in a set is called “noise removal”. When some values are removed from the set, the mean and standard deviation needs to be re-calculated.

- The majority of the server-client, internet, and database applications store the data in a local high speed buffer when a query is made. This enables them to present the data quickly when the same request is made again. This is called "**caching**".
- Once the data sets are organized, the analysis of performance data is carried out to conclude the following
    1. Whether performance of the product is consistent when tests are executed multiple times.
    2. What performance can be expected for what type of configuration resources?
    3. What parameters impact performance and how they can be used to derive better performance?
    4. What is the effect of scenarios involving several mix of operations for the performance factors.
    5. What is the effect of product technologies such as caching on performance improvements?
    6. What is the optimum throughput/response time of the product for a set of factors such as load, resources, and parameters?

6. Performance Tuning

- Performance tuning needs a high degree of skill in identifying the list of parameters and their contribution to performance.
- Two steps involved in getting the optimum mileage performance tuning.
    1. Tuning the product parameters.
    2. Tuning operating system parameters.

Tuning the Product Parameters

- The product parameters in isolation as well as in combination have an impact on product performance.
- Repeat the performance tests for different values of each parameter that impact performance.
- Some times when a particular parameter value is changed, it needs changes in other parameters. Repeat the performance test for a group of parameters and their different values.
- Repeat the performance test for low and high values of all parameters called "factory setting tests".
- Performance tuning provides better results only for a particular configuration and for certain transactions.

Tuning the Operating System Parameters

- The parameters in the operating system are grouped under different categories.
    - ✓ File system related parameter. [Number of open file permitted]
    - ✓ Disk management parameters [simultaneous disk reads/writes]
    - ✓ Memory management parameters[virtual memory page size and number of pages]
    - ✓ Processor management parameters enabling/disabling processors in multiprocessor environment]
    - ✓ Network parameters [setting TCP/IP time out]
- While repeating the tests, the OS parameters need to be tuned before application/product tune is done.

- The machine, on which the parameter is tuned, may have multiple products and applications that are running. Hence tuning as OS parameter may give better results for the product under test, but may heavily impact the other products that are running on the same machine.
- OS parameters need to be tuned only when the complete impact is known to all applications running in the machine or they need to tune only when it is absolutely necessary, giving big performance advantage.
- The results of performance tuning are normally published in the form of a guide called the "performance tuning guide" for customers so that they can benefit.
- The guide explains in detail the effect of each product and OS parameter on performance. It also gives a set of guidance values for the combination of parameter must be tuned in which situation along with associated warnings of any wrong tuning exercise.

7. Performance Benchmarking

- Performance Benchmarking is about comparing the performance of product transaction with that of competitors.
- End-user transactions/scenarios could be one approach for comparison.
- An independent test team of an independent organization not related to the organizations of the products being compared is performance benchmarking.
- The steps involved in performance benchmarking are
  - ✓ Identifying the transactions/scenarios and the test of configuration.
  - ✓ Comparing the performance of different products.
  - ✓ Tuning the parameters of the products being compared fairly to deliver the best performance.
  - ✓ Publishing the results of performance benchmarking.

- Once the tests are executed, the next step is to compare the results. In performance benchmarking all products should be tune to the same degree.
- Three outcomes from performance benchmarking are positive, neutral and negative.
- The results of performance benchmarking are published. Two types of publications are
  - ✓ Internal- confidential publication to product team contains outcomes and recommended set of actions.
  - ✓ Marketing collateral-contains positive outcomes of performance benchmarking which helps as sales tools for the product.

8. Capacity Planning

- In capacity planning, the performance requirements and performance results are taken as input requirements and the configuration needed to satisfy that set of requirements are derived.
- Capacity planning necessitates a clear understanding of the resource requirements for transactions/ scenarios.
- Since the load pattern changes according to future requirements, it is critical to consider those requirements during capacity planning. Capacities planning corresponding to short, medium and long-term requirements are called
  - ✓ Minimum required configuration
  - ✓ Typical configuration
  - ✓ Special configuration

Minimum Required Configuration

- It denotes that with anything less than this configuration, the product may not even work.

Typical Configuration

- It denotes that under configuration, the product will work fine for meeting the performance requirements of the required local pattern and can also handle a slight increase in load pattern.

Special Configuration

- It denotes that capacity planning was done considering all future requirements.
- Two techniques play major role in capacity planning.
    a) Load balancing
    b) High availability

Load Balancing

- Load balancing ensures that the multiple machines available are used equally to service the transactions.
- This ensures that by adding more machines, more loads can be handled by the product.

High Availability

- Machine clusters are used to ensure availability.
- In a cluster these are multiple machines with shared data so that in case one machine goes down, the transactions can be handled by another machine in the cluster.

Tools for Performance Testing

- Two types of tools can be used for performance testing
    a. Functional performance tools
    b. Load tools

Functional Performance Tools

- Functional performance tools help in recording and playing back the transactions and obtaining performance numbers.
- This test generally involves very few machines.

  Eg: Win runner from Mercury, QA partner from Compuware, silk test from segue.

Load Tools

- Load testing tools simulate the load condition for performance testing without having to keep that many users or machines.
- It simplifies the complexities involved in creating the load.

  Eg: load runner from Mercury, QA load from Compuware, Silk performer from Segue.

Process for Performance Testing

- The effort involved in performance testing is more and tests are generally repeated several times.
- Resources needed for performance testing is quite high.

- ❖ Test plan needs the following details.
  - ✓ Resource requirements
  - ✓ Test bed [simulated and real life], test-lab setup.
  - ✓ Responsibilities.
  - ✓ Setting up product traces, audit and traces.
  - ✓ Entry and exit criteria keeping a strong process for performance testing provides a high return on investment.

Challenges

- ✓ Availability of skills is a major problem facing performance testing. [product knowledge, tools usage automation, knowledge on statistics and analytical skills.]
- ✓ Performance testing requires a large number and amount of resources such as hardware, software, effort, time tools and people.
- ✓ Performance test results need to reflect real-life environment and expectations.
- ✓ Interfacing with different teams that includes a set of customer is another challenge.

3.16 REGRESSION TESTING

- ❖ Regression testing is not a level of testing, but it is the retesting of software that occurs when changes are made to ensure that the new version of the software has retained the capabilities of the old version and that no new defects have been introduced due to the changes.
- ❖ Regression testing can occur at any level of test.
- ❖ When unit tests are run the unit may pass a number of these tests until one of the tests 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.
- ❖ Regression tests are especially important when multiple software releases are developed.
- ❖ Users want new capabilities in the latest releases, but still expect the older capabilities to remain in place. This is where regression testing plays a role.

Advantages	Disadvantages
• Ensures that the unchanged parts of software work properly. • Ensures that all errors that have occurred in software due to modifications are corrected and are not affecting the working of software.	• Time consuming activity. • Considered to be expensive.

Need for Regression Testing:

- ❖ Change in requirements and code is modified according to the requirement

- New feature is added to the software
- Defect fixing
- Performance issue fix
- Regression Testing can be carried out using following techniques:

Retest All

- This is one of the methods for regression testing in which all the tests in the existing test bucket or suite should be re-executed. This is very expensive as it requires huge time and resources.

Regression Test Selection

- Instead of re-executing the entire test suite, it is better to select part of test suite to be run
- Test cases selected can be categorized as 1) Reusable Test Cases 2) Obsolete Test Cases.
- Re-usable Test cases can be used in succeeding regression cycles.
- Obsolete Test Cases can't be used in succeeding cycles.

Prioritization of Test Cases

- Prioritize the test cases depending on business impact, critical &frequently used functionalities. Selection of test cases based on priority will greatly reduce the regression test suite.
- Effective Regression Tests can be done by selecting following test cases -
  - ✓ Test cases which have frequent defects
  - ✓ Functionalities which are more visible to the users
  - ✓ Test cases which verify core features of the product
  - ✓ Test cases of Functionalities which has undergone more and recent changes
  - ✓ All Integration Test Cases
  - ✓ All Complex Test Cases
  - ✓ Boundary value test cases
  - ✓ Sample of Successful test cases
  - ✓ Sample of Failure test cases

Challenges in Regression Testing

- Following are the major testing problems for doing regression testing:
  - ✓ With successive regression runs, test suites become fairly large. Due to time and budget constraints, the entire regression test suite cannot be executed
  - ✓ Minimizing test suite while achieving maximum test coverage remains a challenge
  - ✓ Determination of frequency of Regression Tests, i.e., after every modification or every build update or after a bunch of bug fixes, is a challenge.
- Regression testing will be considered after a bug fixed or when any area of functionality changed. During bug fixing method some part of coding may be changed or even functionality may be also manipulated so due to this change we have to perform Regression Testing.

Types of Regression Tests:

- **Final Regression Tests: -** A "final regression testing" is performed to validate the build that hasn't changed for a period of time. This build is deployed or shipped to customers.
- **Regression Tests: -** A normal regression testing is performed to verify if the build has NOT broken any other parts of the application by the recent code changes for defect fixing or for enhancement.

3.17 INTERNATIONALIZATION TESTING

- It is a type of non-functional testing.
- Internationalization testing is the process of verifying the application under test to work uniformly across multiple regions and cultures.
- Internationalization testing is the process, which ensures that product's functionality is not broken and all the messages are properly externalized when used in different languages and locale.
- Internationalization testing is also called **I18N testing**, because there are 18 characters between I and N in Internationalization.
- The main purpose of internationalization is to check if the code can handle all international support without breaking functionality that might cause data loss or data integrity issues.
- Globalization testing verifies if there is proper functionality of the product with any of the locale settings.
- Internationalization Checklists:
  - ✓ Testing to check if the product works across settings.
  - ✓ Verifying the installation using various settings.
  - ✓ Verify if the product works across language settings and currency settings.

- In I18N testing, first step is to identify all the textual information in the system. This includes all the text present on the application's GUI, any text/messages that application is producing including error message/warning and help/documentation etc.
- Main focus of the I18N testing is not to find functional defects, but to make sure that product is ready for the global market.
- As in other non-functional testing it is assumed that functional testing has been completed and all the functionality related defects are identified and removed.
- I18N testing can be divided in to two parts.
  - ✓ First, to make sure that application's GUI or functionality will not be broken with the translated text.
  - ✓ Second to make sure that translation of all the strings have happened properly. This activity is called Translation Verification Testing and is normally conducted by person who knows the language every well.

- To make sure that application's functionality or GUI will not be broken after the translation a popular technique known as pseudo-translation is used.
- In pseudo-translation instead of translating it completely, it is translated in a pseudo manner.
- For example an externalized string "Bad Command" can be translated in Japanese as [JA XXXXX Bad Command XXXXXX JA].

- Now if the product is launched with locale as Japanese it should show the externalized string as given above instead of "Bad Command". There are utilities to do this job for us, to do pseudo-translation of all the externalized strings of our application.
- During pseudo-translation we need to make sure that you are doing it roughly according to the rule. For example, width is normally expanded up to forty percent for the pseudo-translated strings as compare to the English.
- In I18N testing focus is not on the functionality but on the translation and locale related issues.

3.18 AD-HOC TESTING

- Ad hoc test 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.
- It is generally done to uncover defects that are not covered by planned testing.
- Ad hoc testing does make use of any of the test case design technique like equivalence partitioning, boundary value analysis and so on.
- Product defects get tuned to planned test cases and those test cases may not uncover defects in the next test cycle unless new perspectives are added.
- Planned test cases require constant update, sometimes even on a daily basis, incorporating the new learning.
- Updating test cases very frequently may become a time consuming and tedious job. In such a situation ad hoc testing comes handy-to test those perspectives without requiring to formally and immediately updating the test cases.
- One of the most fundamental differences between planned testing and ad-hoc testing is that test execution and test report generation takes place before test case design in ad hoc testing.
- Since ad hoc tests require better understanding of the product, it is important to stay connected. Ad hoc testing is a planned activity. Only the test cases are not documented to start with.
- Ad hoc testing can be performed on a product at any time, but the returns from the ad hoc testing are more if they are run after running planned test cases.
- Ad hoc testing can be planned in one of two ways.
    1. After a certain number of planned test cases are executed.
    2. Prior to planned testing.

- Ad hoc testing does not require the test cases to be documented. But after the test execution, ad hoc testing requires all the perspectives that were tested to be documented as a set of test cases.
- Ad hoc testing can be used to switch the context of the software usage frequently to cover more functionality in less time. Ad hoc testing is also known as **random sampling test** or **monkey testing**.
- Ad hoc testing is normally performed in conjunction with planned testing.
- Ad hoc testing is done as a confidence measure just before the release, to ensure there are no areas that got missed out in testing. Ad hoc testing is applicable for all testing phases.
- Ad hoc testing is performed during unit testing to improve requirements clarity, identify missing codes and to detect defects early.
- Ad hoc testing is performed during component and integration testing to uncover defects not caught earlier by planned test cases.
- During the system and acceptance test phase, ad hoc testing is performed to gain confidence in the product and to catch costly defects that may have been missed.

Drawbacks

- ❖ Difficult to ensure that learning learned in ad hoc testing are used in future.
- ❖ Large number of defects found in ad hoc testing.
- ❖ Lack of comfort on coverage of ad hoc testing
- ❖ Difficult to track the exact steps.
- ❖ Lack of data for metrics and analysis.

Buddy Testing

- ❖ **Buddy testing uses the "buddy system" practice wherein 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 a tester usually become buddies.
- ❖ Buddying people with good working relationships yet having diverse backgrounds is a kind of safety measure that improves the chance of detecting errors in the program very early.
- ❖ Buddies should not feel mutually threatened or get a feeling of insecurity during buddy testing. They are trained on the philosophy and objective of buddy training.
- ❖ The code is unit tested to ensure what it is supposed to do before buddy testing starts.
- ❖ The buddy can check for compliance to coding standards, appropriate variable definitions, missing code, sufficient inline code documentation, and error checking and so on.
- ❖ Buddy testing **uses** both **white-box** and **black box testing approaches**.
- ❖ The buddy, after testing, generates specific review comments and points out specific defects.
- ❖ The buddy may suggest ideas to fix the code when pointing out an error in the work product.
- ❖ The developer reviews the comments, and, if the buddies agree, the appropriate changes are implemented, or else both of them discuss the comments and come to a conclusion.
- ❖ Buddy testing is normally done at the unit test phase, where there are both coding and testing activities.

Pair Testing

- ❖ In pair testing, two tester pair up to test a product's feature on the same machine. The objective of this exercise is to maximize the exchange of ideas between 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.
- ❖ It may not be mandatory for one person to stick one role continuously for an entire session. They can swap roles of "tests" and "scribes" during a session.
- ❖ One person can pair with multiple persons during a day at various points of time for testing.
- ❖ Pair testing is usually a focused session for about an hour or two. It is up to the pair to decide on the different ways of testing this functionality.
- ❖ The presence of one senior member can also help in pairing. This can cut down on the time spent on the learning curve of the product. Pair testing can be done during any phase of testing.
- ❖ When the product is in new domain and not many people have desired domain knowledge, pair testing is useful.
- ❖ Pair testing helps in getting feedback on their abilities from each other. This testing can be used to coach the inexperienced members in the team by pairing them with experienced testers.

- Pair testing can track elusive defects that are not caught by a single person testing. A defect found during the pair testing may be explained better by representation from two members.

Situations when pair testing becomes ineffective

- During pairing, teaming up individual high performers may lead to problems. It may be possible that during the course of the session, one person takes the lead and other has a laidback attitude.
- When one member is working on the computer and the other is playing the role a scribe, if their speed of understanding and execution does not match, it may result in loss of attention.
- Sometimes pairing up juniors with experienced members may result in the former doing tasks that the senior may not want to.

Exploratory Testing

- Another technique to find defects in ad hoc testing is to keep exploring the product, covering more depth and breadth.
- Exploratory testing tries to do that with specific objectives, tasks, and plans. Exploratory testing can be done during any phase of testing.
- Exploratory testers may execute their tests based on their past experience in testing a similar product, or a product of similar domain, or a product in a technology area.
- They use their past experience of finding defects in the previous product release and check if the same problem persists in the current version.
- Exploratory testing can be used to test software that is untested unknown, or unstable.
- Exploring can happen not only for functionality but also for different environments, configuration parameters, test data, and so on.

Exploratory test techniques
There are several ways to perform exploratory testing.

- **Guesses**: Guesses are used to find the part of the program that is likely to have more errors.
- **Architecture Diagrams**, **Use Cases**: Architecture diagrams depict the interactions and relationships between different components and modules. Use cases give an insight of the product s usage from the end users perspective.
- **Past Defects**: studying the defects reported in the previous releases helps in understanding of the error prone functionality in a product development environment.
- **Error Handling**: Error handling is a portion of the code which prints appropriate messages or provides appropriate actions in case of failures.
- **Discussions**: Understanding of the product from discussions. Exploration may be planned base on the understanding of the system during project discussions or meetings. Plenty of information can be picked up during these meetings regarding implementation of different requirements for the product.
- **Questions and Checklists**: questions like “what, when, how, who and why” can provide leads to explore in the product. Eg “what does this module do”, “how is the input processed” “who are the users of this module”.

- ❖ For exploratory testing, a detailed plan can be created specifying the areas to be tested, objectives, and time and effort to be spent.
- ❖ During test execution, areas are identified where there may be more problems and they are probed further. The exploration technique can consider various combinations of inputs, environments, or system configuration parameters.

Iterative Testing

- ❖ The iterative model is where the requirements keep coming and the product is developed iteratively for each requirement. The testing associated for this process is called **iterative testing**.
- ❖ Iterative testing requires repetitive testing. When a new requirement or a defect fix is done, it may have an impact on other requirements that have been already been tested.
- ❖ Majority of these tests are executed manually. Iterative testing aims at testing the product for all requirements, irrespective of the phase they belong to in the spiral model. Customers have a usable product at the end of every iteration
- ❖ Customers and management can notice the impact of defects and the product functionality at the end of each iteration. They can take a call to proceed to the next level or not, based on the observations made in the last iteration.
- ❖ Some types of tests that are performed in later iteration may not be possible to perform during earlier iteration.
- ❖ Test plan gets updated after each iteration since the scope of testing, type of testing and effort involved vary. Developers create unit test cases to ensure that the program developed goes through complete testing.
- ❖ After each iteration, unit test cases are added, edited, or deleted to keep up with the revised requirement for the current phase.
- ❖ In order to avoid the monotony and to increase test efficiency, test that need to be carried forward to all the iterations may be automated, wherever possible.
- ❖ A defect found in one iteration may be fixed in the same build or carried forward, based on the priority decided by the customer.
- ❖ The defect found in second iteration may no longer valid or could have become void due to revised requirements during the third, fourth, and fifth iterations. The functionality that worked in third iteration may fail during the fifth iteration.

3.19 ALPA, BETA TESTING

- ❖ Users have been involved in requirements analysis and reviews, and have played a role in test planning. This is true for acceptance test planning if the software is being custom made for an organization.
- ❖ The clients along with test planners design the actual test cases that will be run during acceptance test.
- ❖ 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 user’s 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.

- 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.
- 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.
- If the software has been developed for the mass market, then testing it for individual clients/users is not practical or even possible in most cases.
- This type of software undergoes two stages of acceptance test.
    1. **Alpha test**
    2. **Beta test**
- **Alpha 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.
- **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. In many cases the repairs are delayed until the next release.

3.20 TESTING OO SYSTEM

- Testing of software developed using OO paradigm has to deal with the new problems also.
- Note that object-oriented testing can be used to test the object-oriented software as well as conventional software.
- OO system should be tested at different levels to uncover all the errors. At the algorithmic level, each module (or method) of every class in the program should be tested in isolation. For this, white-box testing can be applied easily.
- As classes form the main unit of object-oriented program, testing of classes is the main concern while testing an OO program.
- At the class level, every class should be tested as an individual entity. At this level, programmers who are involved in the development of class conduct the testing.
- Test cases can be drawn from requirements specifications, models, and the language used. In addition, structural testing methods such as boundary value analysis are extremely used.

- After performing the testing at class level, cluster level testing should be performed.
- As classes are collaborated (or integrated) to form a small subsystem (also known as cluster), testing each cluster individually is necessary.
- At this level, focus is on testing the components that execute concurrently as well as on the interclass interaction.
- Hence, testing at this level may be viewed as integration testing where units to be integrated are classes.
- Once all the clusters in the system are tested, system level testing begins. At this level, interaction among clusters is tested.
- Usually, there is a misconception that if individual classes are well designed and have proved to work in isolation, then there is no need to test the interactions between two or more classes when they are integrated.
- However, this is not true because sometimes there can be errors, which can be detected only through integration of classes.
- Also, it is possible that if a class does not contain a bug, it may still be used in a wrong way by another class, leading to system failure.

Developing Test Cases in Object-Oriented Testing

- The methods used to design test cases in OO testing are based on the conventional methods. However, these test cases should encompass special features so that they can be used in the object-oriented environment.
- The points that should be noted while developing test cases in an object-oriented environment are listed below:
    - ✓ It should be explicitly specified with each test case which class it should test.
    - ✓ Purpose of each test case should be mentioned.
    - ✓ External conditions that should exist while conducting a test should be clearly stated with each test case.
    - ✓ All the states of object that is to be tested should be specified.
    - ✓ Instructions to understand and conduct the test cases should be provided with each test case.

OBJECT-ORIENTED TESTING METHODS

1. **State based testing**
2. **Fault based testing**
3. **Scenario based testing**

State based testing

- **State-based testing** is used to verify whether the methods (a procedure that is executed by an object) of a class are interacting properly with each other.
- This testing seeks to exercise the transitions among the states of objects based upon the identified inputs.
- For this testing, finite-state machine (FSM) or state-transition diagram representing the possible states of the object and how state transition occurs is built. In addition, state-based testing generates test cases, which check whether the method is able to change the state of object as expected. If any method of the class does not change the object state as expected, the method is said to contain errors.
- To perform state-based testing, a number of steps are followed, which are listed below:

- ✓ Derive a new class from an existing class with some additional features, which are used to examine and set the state of the object.
- ✓ Next, the test driver is written. This test driver contains a main program to create an object, send messages to set the state of the object, send messages to invoke methods of the class that is being tested and send messages to check the final state of the object.
- ✓ Finally, stubs are written. These stubs call the untested methods.

Fault-based Testing

- ❖ Fault-based testing is used to determine or uncover a set of plausible faults. In other words, the focus of tester in this testing is to detect the presence of possible faults.
- ❖ Fault-based testing starts by examining the analysis and design models of OO software as these models may provide an idea of problems in the implementation of software.
- ❖ With the knowledge of system under test and experience in the application domain, tester designs test cases where each test case targets to uncover some particular faults.
- ❖ The effectiveness of this testing depends highly on tester experience in application domain and the system under test. This is because if he fails to perceive real faults in the system to be plausible, testing may leave many faults undetected.
- ❖ However, examining analysis and design models may enable tester to detect large number of errors with less effort.
- ❖ As testing only proves the existence and not the absence of errors, this testing approach is considered to be an effective method and hence is often used when security or safety of a system is to be tested.
- ❖ Integration testing applied for OO software targets to uncover the possible faults in both operation calls and various types of messages (like a message sent to invoke an object).
- ❖ These faults may be unexpected outputs, incorrect messages or operations, and incorrect invocation.
- ❖ The faults can be recognized by determining the behavior of all operations performed to invoke the methods of a class.

Scenario-based Testing

- ❖ Scenario-based testing is used to detect errors that are caused due to incorrect specifications and improper interactions among various segments of the software.
- ❖ Incorrect interactions often lead to incorrect outputs that can cause malfunctioning of some segments of the software.
- ❖ The use of scenarios in testing is a common way of describing how a user might accomplish a task or achieve a goal within a specific context or environment.
- ❖ Note that these scenarios are more context- and user specific instead of being product-specific. Generally, the structure of a scenario includes the following points.
    - ✓ A condition under which the scenario runs.
    - ✓ A goal to achieve, which can also be a name of the scenario.
    - ✓ A set of steps of actions.
    - ✓ An end condition at which the goal is achieved.
    - ✓ A possible set of extensions written as scenario fragments.

- ❖ Scenario- based testing combines all the classes that support a use-case (scenarios are subset of use-cases) and executes a test case to test them.
- ❖ Execution of all the test cases ensures that all methods in all the classes are executed at least once during testing.

- However, testing all the objects (present in the classes combined together) collectively is difficult. Thus, rather than testing all objects collectively, they are tested using either top-down or bottom-up integration approach.
- This testing is considered to be the most effective method as scenarios can be organized in such a manner that the most likely scenarios are tested first with unusual or exceptional scenarios considered later in the testing process.

Challenges in Testing Object-Oriented Programs

- Traditional testing methods are not directly applicable to OO programs as they involve OO concepts including encapsulation, inheritance, and polymorphism.
- These concepts lead to issues, which are yet to be resolved. Some of these issues are listed below.
- Encapsulation of attributes and methods in class may create obstacles while testing.
- As methods are invoked through the object of corresponding class, testing cannot be accomplished without object. In addition, the state of object at the time of invocation of method affects its behavior.
- Hence, testing depends not only on the object but on the state of object also, which is very difficult to acquire.
- Inheritance and polymorphism also introduce problems that are not found in traditional software. Test cases designed for base class are not applicable to derived class always (especially, when derived class is used in different context). Thus, most testing methods require some kind of adaptation in order to function properly in an OO environment.

3.21 USABILITY AND ACCESSIBILITY TESTING

- Usability testing attempts to characterize the "look and feel" and usage aspects of a product, from point of view of users. Some characteristics of usability testing are
    1. Ease of use
    2. Speed
    3. Pleasantness and aesthetics

- Usability testing addresses these from the point of view of a user.

Approach to Usability

- When doing usability testing, certain human factors can be represented in a quantifiable way and can be tested objectively.
- The number of mouse clicks, number of sub-menus to navigate, number of keystrokes, number of commands to perform a task can all measured and checked as part of usability testing.
- Usability improvements sometimes can be very marginal but can give huge benefits, with the number of users for the product increasing.
- Usability testing is not only for product binaries or executable. It also applies to documentation and other deliverables that are shipped along with a product. The release media should also be varied for suitability
- The people suited to perform usability testing are
    - ✓ Representative of actual user segments.
    - ✓ People who are new to the product, so that they can start without any bias and be able to identify usability problems.

- Another aspect of usability is with respect to messages that a system gives to its users.
- Messages are classified into three types.
  - ✓ Informal message
  - ✓ Warning message
  - ✓ Error message

Informal Message

- Informal message is verified to find out whether an end-user can understand the message and associate it with the operation done and the context.

Warning Message

- Warning message is checked for why it happened and what to do to avoid the warning.

Error Message

- Wherever there is an error message, three things are looked for, what is the error, why that error happened, and what to do to avoid or work around that error.

- Usability should go through both positive and negative testing-that is both correct and incorrect usage of the product.
- Usability should also consider Command Line Interface {CLI} and other interfaces that are used by the users.

When to Do Usability Testing

- Usability testing is performed in two phases
  - ✓ Design validation
  - ✓ Usability testing done as a part of component and integration testing phases of test cycle.

- When planning for testing, the usability requirements should be planned in a parallel, upfront in the development cycle, similar to any other type of testing.
- When there are two defects one on functionality and the on usability the functionality defect is usually given precedence.
- Postponing usability defects testing in a testing cycle can prove to be very expensive , as a large number of usability defects may end up as needing changes in design and needing fixes in more than one screen, affecting different code paths. All these situations can be avoided if usability testing is planned upfront.
- In the first phase of usability testing usability design is validated. Usability design is verified through several means. Some of them are
  - ✓ Style sheets
  - ✓ Screen prototypes
  - ✓ Paper design
  - ✓ Layout design

- In the second phase tests are run to test the product for usability. When to do usability also depends on the type of the product that is being developed.

Development and Testing Of Client Application

Step 1: Design for functionality

Step 2: Perform coding for functionality

Step 3: Design for user interface

Step 4: Perform coding for user interface

Step 5: Integrate user interface with functionality

Step 6: Test the user interface along with functionality [phase 1 and phase 2]

Development and Testing Of Web Application

Step 1 : Design the user interface

Step 2 : Perform coding for user interface

Step 3 : Test user interface [phase 1]

Step 4 : Design for functionality

Step 5 : Perform coding for functionality

Step 6 : Integrate user interface with functionality

Step 7 : Test user interface along with functionality [phase 2]

How to Achieve Usability

- Involving the customer to give Feedback on all the user interface requirements upfront or during the design phase is essential. Different categories of users are
  - ✓ Experts
  - ✓ Beginners
  - ✓ Novices

Expert Users

- Expert users do no report usability problems. Instead they invent workarounds and adapt themselves to the product though they hope for a better product

Beginners

- Beginners get impacted because of lack of usability but still they do not report usability problems as they are eager to learn more about the product.

Novice Users

- Novice users report plenty of usability issues but some of them may get ignored because they do not have adequate skills to use the product. Irrespective of the category of users the product is expected to be usable by all.

- The product is expected to be usable to another category of challenged users such as those who are hearing impaired, vision impaired, and mobility impaired. This type of usability that deals with overall visual and motion challenged user is called “accessibility testing”.
- When the users are using the product during usability testing their activities are closely monitored and all observations are recorded and defects are raised by the test team rather than expecting every problem to be reported. In order to create a product that is usable for categories of the users, at least one user from each category must be involved in usability testing.

- Recording the operating sequence screens and user reactions and making observations needs some equipment to be set up for "usability".

Quality Factors for Usability

- Some of the quality for usability is
  - ✓ Comprehensibility
  - ✓ Consistency
  - ✓ Navigation
  - ✓ Responsiveness

Comprehensibility

- The product should have simple and logical structure of features and documentation. They should be grouped on the basis of user scenarios and usage.The most frequent operations that are performed early in a scenario should be presented first, using the user interfaces.

Consistency

- A product needs to be consistent with any applicable standards, platform look-and –feel, base infrastructure, and earlier versions of the same product.

Navigation

- This helps in determining how easy it is to select the different operations of the product.The number of mouse clicks, or menu navigations that is required to perform an operation should be minimized to improve usability.

- When users get stuck or get lost, there should be an easy option to abort or go back to the previous screen or to the main menu so that the user can try a different route.

Responsiveness

- How fast the product responds to the user request is another important aspect of usability. Whenever the product is processing some information, the visual display should indicate the progress and also the amount of time left so that the users can wait patiently till operation is completed.

Aesthetics Testing

- Important aspect in usability is making the product beautiful. Performing aesthetic testing helps in improving usability further
- Adequate care for the aesthetic aspect of the product can ensure that product is beautiful, at least product must not end up being termed ugly. Beauty sets the first impression for any product.
- In many product companies, aesthetic takes a back seat. Aesthetics is not in the external look alone. It is in all the aspect s such as messages, screens, colours, and images.
- A pleasant look for menus, leasing colours, nice icons, and so on can improve aesthetics
- Some of the aesthetics aspects must be done during the design phase and should be taken as the last and low-priority activity before the release.
- Aesthetics testing can be performed by anyone who appreciates beauty, which means everyone

Accessibility Testing

- There are large numbers of people who are challenged with vision, hearing, and mobility related problems- partial or complete.
- Product usability that does not look into their requirements would result in lack of acceptance.
- Accessibility tools are available to help them with alternatives.

- Accessibility testing involves testing these alternative methods of using the product and testing the product along with the accessibility tools.
- Accessibility is a subset of usability and should be included as part of usability test planning.
- Accessibility to product can be provided by two means
  - ✓ Basic accessibility
  - ✓ Product accessibility

Basic Accessibility

- Basic accessibility 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.

Keyboard Accessibility

- An example of hardware accessibility improvement is the little projection on can find in any key board on top of the F and J keys. This little projections help vision impaired users to get the feel and align their fingers for typing.
- Key board keys with different sizes and providing short cut function keys are other examples of improving accessibility, provided at the hardware level.
- The operating system vendors came up with some more improvements in the key board.
- Some of the improvements are
  - ✓ Sticky keys
  - ✓ Toggle keys
  - ✓ Arrow keys
  - ✓ Filter keys
  - ✓ Sound keys
  - ✓ Narrator

Sticky Keys

- One of the most complex sequenced for vision impaired and mobility impaired users is <CTRL><ALT><DEL>. This key board sequence is used for various purpose such as log in, log out, locking and unlocking machines, shutdown and bring up task manager.
- When sticky keys feature is enabled, <CTRL>, <ALT> keys are pressed once and released by the user before pressing <DEL> key.

  This allows a single finger operation to complete the sequence.

Toggle Keys

- When toggle keys are enabled the information typed may be different from what the user desires.Eg: INSERT key, NUM LOCK key
- Vision impaired users find it difficult to know the status of these toggle keys. To solve this problem sound is enabled, and the different tones are played when enabling and disabling toggle keys.

Arrow Keys

- ❖ Mobility impaired users have problems moving the mouse. Such users will be able to use the key board arrow keys for mouse movement.
- ❖ The two buttons of the mouse and their operations too can be directed from the key board.

Filter Keys

- ❖ When keys are pressed for more than a particular duration, they are assumed to be repeated. Filter keys help in either stopping the repetition completed or slowing down the repetition.

Sound Keys

- ❖ To help vision impaired users; there is one more mechanism that pronounces each character as and when they are hit on the key board.

Narrator

- ❖ Narrator is a utility which provides auditory feedback. For example, it may pronounce the events when they are executed by the users read out the characters typed, notify the system events by distinguishing sounds, and so on.

Screen Accessibility

- ❖ Hearing impaired users require extra visual feedback on the screen. Some accessibility features that enhance usability using the screen are as follows:

Visual Sounds

- ❖ Visual sounds are the “waveform” or “graph form” of the sound.

Enabling Captions for Multimedia

- ❖ All multimedia speech and sound can be enabled with text equivalents and they are displayed on the screen when speech and sound are played.

Soft Key Board

- ❖ A soft key board helps mobility impaired users by displaying the key board on the screen.

3.22 CONFIGURATION TESTING

- ❖ Configuration testing is the process of checking the operation of the software we are testing with all these various types of hardware.
- ❖ This is usually a dynamic white-box testing and programmer-debugging effort.
- ❖ A configuration problem can occur for several reasons, all requiring someone to carefully examine the code while running the software under different configurations to find the bug:
  - ✓ Software may have a bug that appears under a broad class of configurations.
  - ✓ An example is if your greeting card program works fine with laser printers but not with inkjet printers.
  - ✓ Software may have a bug specific only to one particular configuration—it doesn’t work on the OkeeDoKee Model BR549 Ink Jet Deluxe printer.
  - ✓ The hardware device or its device drivers may have a bug that only your software.

- ✓ Maybe your software is the only one that uses a unique display card setting. When your software is run with a specific video card, the PC crashes.
- ✓ The hardware device or its device drivers may have a bug that can be seen with lots of other software—although it may be particularly obvious with yours.
- ✓ An example would be if a specific printer driver always defaulted to draft mode and your photo printing software had to set it to high-quality every time it printed.

Sizing up the Job

- ❖ The job of configuration testing can be a huge undertaking. Suppose that we are testing a new software game that runs under Microsoft Windows. The game is very graphical, has lots of sound effects, allows multiple players to compete against each other over the phone lines, and can print out game details for strategy planning.
- ❖ We need to consider configuration testing with different graphics cards, soundcards, modems, and printers.
- ❖ There are approximately 336 possible display cards, 210 sound cards, 1500 modems, and 1200 printers. The number of test combinations is 336 × 210 ×1500 ×1200, for a total in the billions.
- ❖ The answer to this is equivalence partitioning.

Approaching the Task

- ❖ The decision-making process that goes into deciding what devices to test with and how they should be tested is a fairly straightforward equivalence partition project. What's important, and what makes the effort a success or not, is the information you use to make the decisions.
- ❖ The following sections show the general process that we should use when planning your configuration testing.
    - ✓ Decide the Types of Hardware Needed
    - ✓ Decide What Hardware Brands, Models, and Device Drivers Are Available
    - ✓ Decide Which Hardware Features, Modes, and Options Are Possible
    - ✓ Pare Down the Identified Hardware Configurations to a Manageable Set
    - ✓ Identify Software's Unique Features That Work withthe Hardware Configurations
    - ✓ Design the Test Cases to Run on Each Configuration
    - ✓ Execute the Tests on Each Configuration
    - ✓ Rerun the Tests Until the Results Satisfy the Team

Obtaining the Hardware

- ❖ Here are a few ideas for overcoming this problem:
    - ✓ Buy only the configurations that can or will use most often.
    - ✓ Contact the hardware manufacturers and ask if they will lend or even give the hardware.
    - ✓ Send a memo or email to everyone in your company asking what hardware they have in their office or even at home and if they would allow you to run a few tests on it.

Identifying Hardware Standards

- ❖ If interested in static black-box analysis, that is, reviewing the specifications that the hardware companies use to create their products, knowing some details of the hardware specifications can help us make more informed equivalence partition decisions.
- ❖ For Apple hardware, visit the Apple Hardware Web site at http://developer.apple.com/ hardware/.

- For PCs, the best link is http://www.pcdesignguide.org/.
- Microsoft publishes a set of standards for software and hardware to receive the Windows logo. That information is at http://msdn.microsoft.com/certification/ and http://www.microsoft.com/hwtest.

Configuration Testing Other Hardware

- If we are testing software for an industrial controller, a network, medical devices, or a phone system, ask questions as:
  - ✓ What external hardware will operate with this software?
  - ✓ What models and versions of that hardware are available?
  - ✓ What features or options does that hardware support?
- Create equivalence partitions of the hardware based on input from the people who work with the equipment, your project manager, or your sales people.
- Develop test cases, collect the selected hardware, and run the tests.

3.23 COMPATIBILITY TESTING

- ***Software compatibility testing*** means checking that our software interacts with and shares information correctly with other software.
- This interaction could occur between two programs simultaneously running on the same computer or even on different computers connected through the Internet thousands of miles apart.
- The interaction could also be as simple as saving data to a floppy disk and hand-carrying it to another computer across the room.
- Examples of compatible software are
  - ✓ Cutting text from a Web page and pasting it into a document opened in your word processor
  - ✓ Saving accounting data from one spreadsheet program and then loading it into a completely different spreadsheet program
  - ✓ Having photograph touch up software work correctly on different versions of the same operating system
  - ✓ Having your word processor load in the names and addresses from your contact management program and print out personalized invitations and envelopes
  - ✓ Upgrading to a new database program and having all your existing databases load in and work just as they did with the old program
- If we are assigned the task of performing software compatibility testing on a new piece of software, we need to get the answers to few questions:
  - ✓ What other *platforms* (operating system, Web browser, or other operating environment) and other application software is your software designed to be compatible with? If the software you're testing is a platform, what applications are designed to run under it?
  - ✓ What compatibility standards or guidelines should be followed that define how your software should interact with other software?
  - ✓ What types of data will your software use to interact and share information with other platforms and software?
- Gaining the answers to these questions is basic static testing - both black-box and white-box.

Platform and Application Versions

- ❖ Selecting the target platforms or the compatible applications is really a program management or a marketing task.
- ❖ For example, we seen notices such as these on software packages or start up screens:
  - ✓ Works best with Netscape 4.0
  - ✓ Requires Windows 95 or greater
  - ✓ For use with Linux kernel 2.2.16
- ❖ This information is part of the specification and tells the development and test teams what they're aiming for.
- ❖ Each platform has its own development criteria and it's important, from a project management standpoint, to make this platform list as small as possible but still fill the customer's needs.

Backward and Forward Compatibility

- ❖ If something is backward compatible, it will work with previous versions of the software.
- ❖ If something is forward compatible, it will work with future versions of the software.

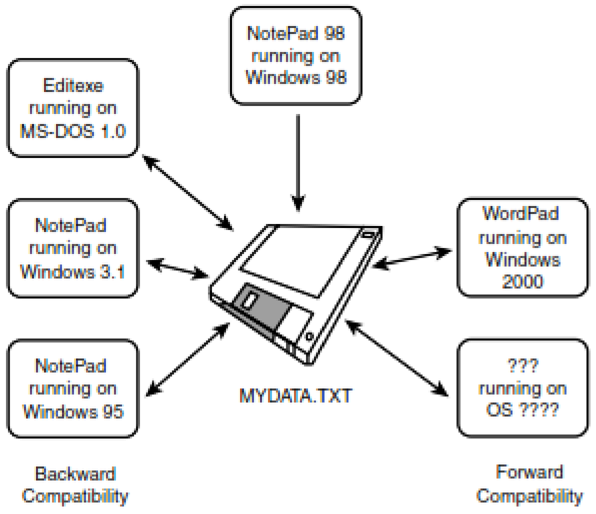

Editexe running on MS-DOS 1.0
NotePad 98 running on Windows 98
NotePad running on Windows 3.1
WordPad running on Windows 2000
NotePad running on Windows 95
MYDATA.TXT
??? running on OS ????
Backward Compatibility
Forward Compatibility

Figure 3.6 : Backward and forward compatibility define what versions will work with your software or data files.

The Impact of Testing Multiple Versions

- ❖ Testing that multiple versions of platforms and software applications work properly with each other can be a huge task
- ❖ Consider the situation of having to compatibility test a new version of popular operating system.
- ❖ The programmers have made numerous bug fixes and performance improvements and have added many new features to the code.
- ❖ There could be tens or hundreds of thousands of existing programs for the current versions of the OS. The project's goal is to be100 percent compatible with them.

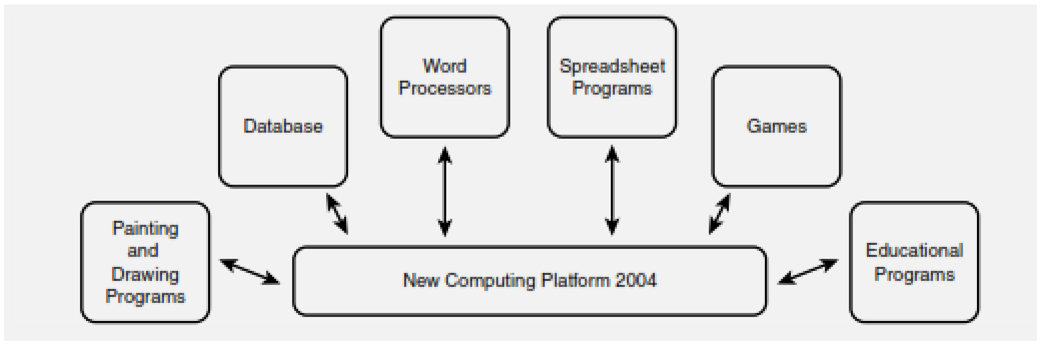


Figure 3.7 : If you compatibility test a new platform, you must check that existing software applications work correctly with it.

- ❖ We need to decide which ones are the most important to test. The key word is important. Thecriteria that might go into deciding what programs to choose could be
    - ✓ **Popularity:** Use sales data to select the top 100 or 1,000 most popular programs.
    - ✓ **Age:** You might want to select programs and versions that are less than three years old.
    - ✓ **Type:** Break the software world into types such as painting, writing, accounting, databases, communications, and so on. Select software from each category for testing.
    - ✓ **Manufacturer:** Another criterion would be to pick software based on the company that created it.

Standards and Guidelines

- ❖ There are really two levels of these requirements: **high-level** and **low-level.**
- ❖ High-level standards are the ones that guide your product's general compliance, its look and feel, its supported features, and so on.
- ❖ Low-level standards are the nitty-gritty details, such as the file formats and the network communications protocols.
- ❖ Both are important and both need to be tested to assure compatibility.

Data Sharing Compatibility

- ❖ The most familiar means of transferring data from one program to another is saving and loading disk files.
- ❖ Adhering to the low-level standards for the disk and file formats makes this sharing possible

  Examples
    - ✓ ***Files save and file load*** are the data sharing methods that everyone is aware of. We save our data to a floppy disk (or some other means of network, magnetic, or optical storage) and then transfer it over to another computer running different software. The data format of the files needs to meet standards for it to be compatible on both computers.
    - ✓ ***File export and file import*** are the means that many programs use to be compatible with older versions of themselves and with other programs.
    - ✓ ***Cut, copy,* and *paste*** are the most familiar methods for sharing data among programs without transferring the data to a disk. In this case, the transfer happens in memory through an intermediate program called the *Clipboard*.

- ✓ ***DDE* (pronounced D-D-E) and *OLE* (pronounced oh-lay)** are the methods in Windows of transferring data between two applications. DDE stands for Dynamic Data Exchange and OLE stands for Object Linking and Embedding.

3.24 TESTING THE DOCUMENTATION

Types of Software Documentation

- ❖ **Packaging text and graphics:** This includes the box, carton, wrapping, and so on. The documentation might contain screen shots from the software, lists of features, system requirements, and copyright information.
- ❖ **Marketing material, ads, and other inserts:** These are all the pieces of paper you usually throw away, but they are important tools used to promote the sale of related software, add-on content, service contracts, and so on. The information for them must be correct for a customer to take them seriously.
- ❖ **Warranty/registration:** This is the card that the customer fills out and sends in to register the software. It can also be part of the software and display onscreen for the user to read, acknowledge, and even complete online.
- ❖ **EULA.** Pronounced "you-la," it stands for *End User License Agreement*. This is the legal document that the customer agrees to that says, among other things, that he won't copy the software nor sue the manufacturer if he's harmed by a bug. The EULA is sometimes printed on the envelope containing the media—the floppy or CD. It also may pop up onscreen during the software's installation.
- ❖ **Labels and stickers.** These may appear on the media, on the box, or on the printed material. There may also be serial number stickers and labels that seal the EULA envelope. Figure shows an example of a disk label and all the information that needs to be checked.
- ❖ **Installation and setup instructions:** Sometimes this information is printed on the media, but it also can be included as a separate sheet of paper or, if it's complex software, as an entire manual.
- ❖ **User's manual.** The usefulness and flexibility of online manuals has made printed manuals much less common than they once were. Most software now
  comes with a small, concise "getting started"–type manual with the detailed information moved to online format. The online manuals can be distributed on the software's media, on a Web site, or a combination of both.
- ❖ **Online help:** Online help often gets intertwined with the user's manual, sometimes even replacing it. Online help is indexed and searchable, making it much easier for users to find the information they're looking for. Many online help systems allow natural language queries so users can type **Tell me how to copy text from one program to another** and receive an appropriate response.
- ❖ **Tutorials, wizards, and CBT (Computer Based Training):** These tools blend programming code and written documentation. They're often a mixture of both content and highlevel, macro-like programming and are often tied in with the online help system. A user can ask a question and the software then guides him through the steps to complete the task. Microsoft's Office Assistant, sometimes referred to as the "paper clip guy" is an example of such a system.

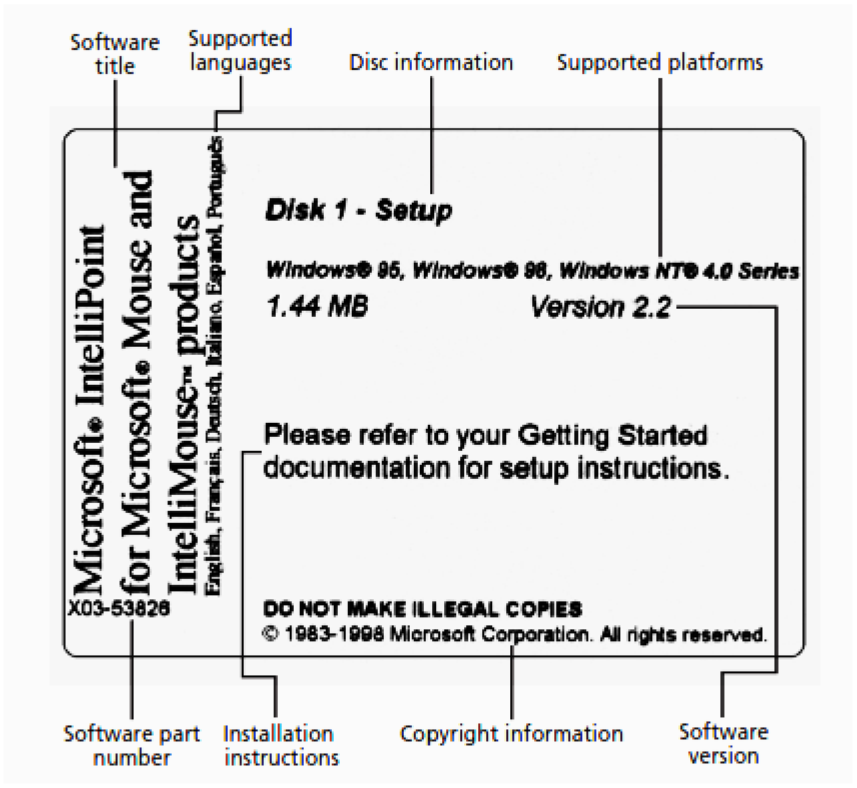

Software title
Supported languages
Disc information
Supported platforms
Microsoft® IntelliPoint for Microsoft® Mouse and IntelliMouse™ products
English, Français, Deutsch, Italiano, Español, Português
Disk 1 - Setup
Windows® 95, Windows® 98, Windows NT® 4.0 Series
1.44 MB
Version 2.2
Please refer to your Getting Started documentation for setup instructions.
X03-53826
DO NOT MAKE ILLEGAL COPIES
© 1983-1998 Microsoft Corporation. All rights reserved.
Software part number
Installation instructions
Copyright information
Software version

Figure 3.8: Documentation on a disk label for the software tester to check.

- **Samples, examples, and templates:** An example of these would be a word processor with forms or samples that a user can simply fill in to quickly create professional-looking results. A compiler could have snippets of code that demonstrate how to use certain aspects of the language.
- **Error messages:** These have already been discussed a couple times in this book as an often neglected area, but they ultimately fall under the category of documentation.

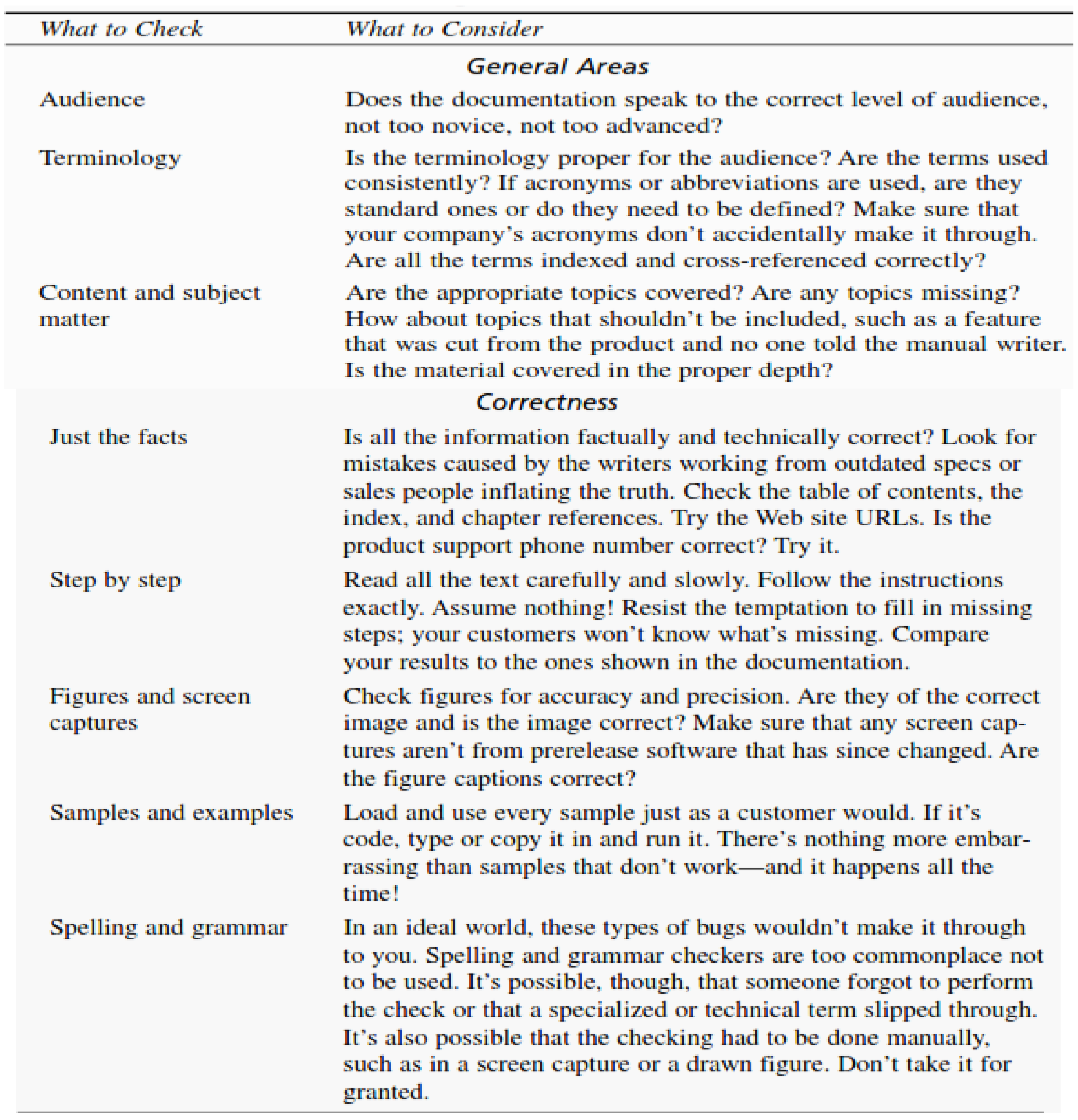
The Importance of Documentation Testing

- Software users consider all these individual non-software components parts of the overall software product. They don't care whether the pieces were created by a programmer, a writer, or a graphic artist. What they care about is the quality of the entire package.
- Good software documentation contributes to the product's overall quality in three ways:
  - ✓ **It improves usability:**"Usability Testing," all the issues related to a product's usability? Much of that usability is related to the software documentation.
  - ✓ **It improves reliability:** Reliability is how stable and consistent the software is.
  - ✓ **It lowers support costs.**

What to Look for When Reviewing Documentation

- The following Table is a simple checklist to use as a basis for building your documentation test cases.

Table 3.1: Checklist for Building Documentation test case



What to Check	*What to Consider*
	General Areas
Audience	Does the documentation speak to the correct level of audience, not too novice, not too advanced?
Terminology	Is the terminology proper for the audience? Are the terms used consistently? If acronyms or abbreviations are used, are they standard ones or do they need to be defined? Make sure that your company's acronyms don't accidentally make it through. Are all the terms indexed and cross-referenced correctly?
Content and subject matter	Are the appropriate topics covered? Are any topics missing? How about topics that shouldn't be included, such as a feature that was cut from the product and no one told the manual writer. Is the material covered in the proper depth?
	Correctness
Just the facts	Is all the information factually and technically correct? Look for mistakes caused by the writers working from outdated specs or sales people inflating the truth. Check the table of contents, the index, and chapter references. Try the Web site URLs. Is the product support phone number correct? Try it.
Step by step	Read all the text carefully and slowly. Follow the instructions exactly. Assume nothing! Resist the temptation to fill in missing steps; your customers won't know what's missing. Compare your results to the ones shown in the documentation.
Figures and screen captures	Check figures for accuracy and precision. Are they of the correct image and is the image correct? Make sure that any screen captures aren't from prerelease software that has since changed. Are the figure captions correct?
Samples and examples	Load and use every sample just as a customer would. If it's code, type or copy it in and run it. There's nothing more embarrassing than samples that don't work—and it happens all the time!
Spelling and grammar	In an ideal world, these types of bugs wouldn't make it through to you. Spelling and grammar checkers are too commonplace not to be used. It's possible, though, that someone forgot to perform the check or that a specialized or technical term slipped through. It's also possible that the checking had to be done manually, such as in a screen capture or a drawn figure. Don't take it for granted.

The Realities of Documentation Testing

- Documentation often gets the least attention, budget, and resources.
- It's possible that the people writing the documentation aren't experts in what the software does.
- Printed documentation takes time to produce, sometimes weeks or even months.

3.25 WEBSITE TESTING

- ❖ Web testing is the name given to software testing that focuses on web applications.
- ❖ Complete testing of a web-based system before going live can help address issues before the system is revealed to the public.
- ❖ Issues such as the security of the web application, the basic functionality of the site, its accessibility to handicapped users and fully able users, as well as readiness for expected traffic and number of users and the ability to survive a massive spike in user traffic, both of which are related to load testing.

Web testing checklist:

1) Functionality Testing
2) Usability testing
3) Interface testing
4) Compatibility testing
5) Performance testing
6) Security testing

1. Functionality Testing:

- ❖ Test for – all the links in web pages, database connection, forms used in the web pages for submitting or getting information from user, Cookie testing.

Check all the links:

- ✓ Test the outgoing links from all the pages from specific domain under test.
- ✓ Test all internal links.
- ✓ Test links jumping on the same pages.
- ✓ Test links used to send the email to admin or other users from web pages.
- ✓ Test to check if there are any orphan pages.
- ✓ Lastly in link checking, check for broken links in all above-mentioned links.

Test forms in all pages:

- ❖ Forms are the integral part of any web site. Forms are used to get information from users and to keep interaction with them. So what should be checked on these forms?

  - ✓ First check all the validations on each field.
  - ✓ Check for the default values of fields.
  - ✓ Wrong inputs to the fields in the forms.
  - ✓ Options to create forms if any, form delete, view or modify the forms.

- ❖ Example: search engine project
- ❖ In this project we have advertiser and affiliate signup steps. Each sign up step is different but dependent on other steps. So sign up flow should get executed correctly. There are different field validations like email Ids, User financial info validations. All these validations should get checked in manual or automated web testing.

Cookies testing:

- Cookies are small files stored on user machine. These are basically used to maintain the session mainly login sessions. Test the application by enabling or disabling the cookies in your browser options.
- Test if the cookies are encrypted before writing to user machine. If you are testing the session cookies (i.e. cookies expire after the sessions ends) check for login sessions and user stats after session end. Check effect on application security by deleting the cookies.

Validate HTML/CSS:

- If we are optimizing our site for Search engines then HTML/CSS validation is very important. Mainly validate the site for HTML syntax errors. Check if site is crawlable to different search engines.

Database testing:

- Data consistency is very important in web application. Check for data integrity and errors while you edit, delete, modify the forms or do any DB related functionality.
- Check if all the database queries are executing correctly, data is retrieved correctly and also updated correctly.

2.Usability Testing:

Test for navigation:

- Navigation means how the user surfs the web pages, different controls like buttons, boxes or how user using the links on the pages to surf different pages.
- Usability testing includes:
    - ✓ Web site should be easy to use. Instructions should be provided clearly. Check if the provided instructions are correct means whether they satisfy purpose.
    - ✓ Main menu should be provided on each page. It should be consistent.

Content checking:

- Content should be logical and easy to understand. Check for spelling errors. Use of dark colors annoys users and should not be used in site theme. We can follow some standards that are used for web page and content building. These are common accepted standards like as I mentioned above about annoying colours, fonts, frames etc.
- Content should be meaningful. All the anchor text links should be working properly. Images should be placed properly with proper sizes.
- These are some basic standards that should be followed in web development. Your task is to validate all for UI testing

Other user information for user help:

- Like search option, sitemap, help files etc. Sitemap should be present with all the links in web sites with proper tree view of navigation. Check for all links on the sitemap.
- "Search in the site" option will help users to find content pages they are looking for easily and quickly. These are all optional items and if present should be validated.
- Common issues on web design:
    - ✓ Gratuitous Use of Bleeding-Edge Technology.
    - ✓ Scrolling Text, Marquees, and Constantly Running Animations.
    - ✓ Long Scrolling Pages.
    - ✓ Non-Standard Link Colors.

- ✓ Outdated Information.
- ✓ Overly Long Download Times.
- ✓ Lack of Navigation Support.
- ✓ Orphan Pages.
- ✓ Complex Web Site Addresses (URLs).
- ✓ Using Frames.

3. Interface Testing:

- ❖ The main interfaces are:
  - ✓ Web server and application server interface
  - ✓ Application server and Database server interface.

- ❖ Check if all the interactions between these servers are executed properly. Errors are handled properly. If database or web server returns any error message for any query by application server then application server should catch and display these error messages appropriately to users. Check what happens if user interrupts any transaction in-between? Check what happens if connection to web server is reset in between?

4) Compatibility Testing:

- ❖ Compatibility of your web site is very important testing aspect. See which compatibility test to be executed:
  - ✓ Browser compatibility
  - ✓ Operating system compatibility
  - ✓ Mobile browsing
  - ✓ Printing options

Browser compatibility:

- ❖ Some applications are very dependent on browsers. Different browsers have different configurations and settings that your web page should be compatible with. Your web site coding should be cross browser platform compatible. If we are using java scripts or AJAX calls for UI functionality, performing security checks or validations then give more stress on browser compatibility testing of your web application.
- ❖ Test web application on different browsers like Internet explorer, Firefox, Netscape navigator, AOL, Safari, Opera browsers with different versions.

OS compatibility:

- ❖ Some functionality in your web application is may not be compatible with all operating systems. All new technologies used in web development like graphics designs, interface calls like different API's may not be available in all Operating Systems.
- ❖ Test our web application on different operating systems like Windows, Unix, MAC, Linux, Solaris with different OS flavours.

Mobile browsing:

- ❖ This is new technology age. So in future Mobile browsing will rock. We must Test our web pages on mobile browsers. Compatibility issues may be there on mobile.

Printing options:

- ❖ If we are giving page-printing options then make sure fonts, page alignment, page graphics getting printed properly. Pages should be fit to paper size or as per the size mentioned in printing option.

5. Performance Testing:

- ❖ Web application should sustain to heavy load. Web performance testing should include:
  - ✓ Web Load Testing
  - ✓ Web Stress Testing
- ❖ Test application performance on different internet connection speed.
- ❖ In **web load testing** test if many users are accessing or requesting the same page. Can system sustain in peak load times? Site should handle many simultaneous user requests, large input data from users, Simultaneous connection to DB, heavy load on specific pages etc.
- ❖ **Stress testing**: Generally stress means stretching the system beyond its specification limits. Web stress testing is performed to break the site by giving stress and checked how system reacts to stress and how system recovers from crashes.
- ❖ Stress is generally given on input fields, login and sign up areas.
- ❖ In **web performance testing** web site functionality on different operating systems, different hardware platforms is checked for software, hardware memory leakage errors,

6.Security Testing:

- ❖ Following are some test cases for web security testing:
  - ✓ Test by pasting internal url directly into browser address bar without login. Internal pages should not open.
  - ✓ If you are logged in using username and password and browsing internal pages then try changing url options directly. I.e. If you are checking some publisher site statistics with publisher site ID= 123. Try directly changing the url site ID parameter to different site ID which is not related to logged in user. Access should denied for this user to view others stats.
  - ✓ Try some invalid inputs in input fields like login username, password, input text boxes. Check the system reaction on all invalid inputs.
  - ✓ Web directories or files should not be accessible directly unless given download option.Test the CAPTCHA for automates scripts logins.
  - ✓ Test if SSL is used for security measures. If used proper message should get displayed when user switch from non-secure http:// pages to secure https:// pages and vice versa.
  - ✓ All transactions, error messages, security breach attempts should get logged in log files somewhere on web server.

QUESTION BANK

PART – A

The Need for Levels of Testing

1. What are the various skills needed by a test specialist. [Nov 2009][IT - Nov / Dec 2014 -8M]

- ✓ Experience and interests in field of software tests
- ✓ Planning and prioritizing tasks
- ✓ Resolving problems in a timely manner

2. What are the various levels of testing?

- ✓ Unit test
- ✓ Integration test
- ✓ System test
- ✓ Acceptance test.

Unit Test

3. Define unit test. Give example. [May / Jun 2012] [Nov 2017]

Unit testing is a test (often automated) that validates that individual units of source code are working properly.
For example you are testing a function; whether loop or statement in a program is working properly or not than this is called as **unit testing**

4. Write the workable definition for a software unit and characterize it.[Nov / Dec 2012]

A unit is the smallest possible testable software component. A unit in procedure-oriented software system:

- Performs a single cohesive function;
- Can be compiled separately;
- Is a task in a work breakdown structure;
- Contains code that can fit on a single page or screen.

5. List the components suitable for unit test.

- ✓ Procedures and functions
- ✓ Classes/objects and methods
- ✓ Procedure-sized reusable components.

6. What is the need for preparation to perform unit test?

Unit test can be done effectively by planning properly. Planning includes designing tests 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.

7. Mention the approach used for designing unit test case.

Test case design at the unit level can be based on use of the black and white box test design strategies. Both of these approaches are useful for designing test cases for functions and procedures. They are also useful for designing tests for the individual methods (member functions) contained in a class.

Unit Test Planning

8. List the phases of unit test Planning.

- ✓ Phase 1: Describe unit test approach and risks
- ✓ Phase 2: Identify unit features to be tested.
- ✓ Phase 3: Add levels of detailed to the plan.

Designing the Unit Tests

9. List the issues in the unit test.

- ✓ Issue 1: Adequately testing classes.
- ✓ Issue 2: Observation of objects states and state changes.
- ✓ Issue 3: The retesting of classes-I
- ✓ Issue 4: The retesting of classes-II

The Test Harness

10. Define test harness.[Nov / Dec 2012][IT Nov 2016]

The auxiliary code developed to support to testing of units and components is called a test harness. The harness consists of drivers that call the target code and stubs that represent modules it calls.

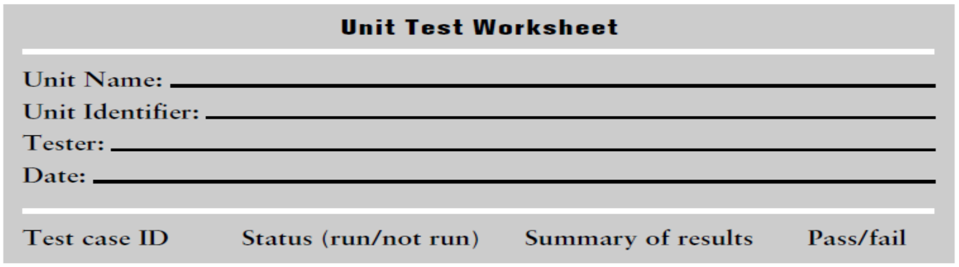
11. Why is it so important to design a test harness for reusability?[Nov 2014][May 2017] [Nov 2017]

- ✓ Testing can occur at times that the office is not staffed (e.g. at night)
- ✓ Increased quality of software components and application.
- ✓ Increased productivity due to automation of the testing process.

Running the Unit tests and Recording results

12. How are the results of unit test recorded?

The status of the test efforts for a unit, and a summary of the test results, could be recorded in a simple format as shown below:



Unit Test Worksheet

Unit Name: ____

Unit Identifier: ____

Tester: ____

Date: ____

Test case ID	Status (run/not run)	Summary of results	Pass/fail

Integration tests

13. List the major goals of Integration test.

- ✓ To detect defects that occurs on the interfaces of units.
- ✓ To assemble the individual units into working subsystems and the finally a Complete system that is ready for system test

Designing Integration Tests

14. What are the factors involved in designing for integration test?

- ✓ The parameters could be involved in a number of def and/or use data flow patterns.
- ✓ For conventional systems, input/output parameters and calling relationships will appear in a structure chart built during detailed design.
- ✓ Testers must insure that test cases are designed so that all modules in the structure chart are called at least once, and all called modules are called by every caller.

Integration Test Planning

15. What are the documents necessary for integration test planning?

Documents relevant to integration test planning are the requirements document, the user manual, and usage scenarios. These documents contain structure charts, state charts, data dictionaries, cross-reference tables, module interface descriptions, data flow descriptions, messages and event descriptions, all necessary to plan integration tests.

16. What are the items need to be included in the integration test plan?

The plan must include the following items

- ✓ Clusters this cluster is dependent on;
- ✓ A natural language description of the functionality of the cluster to be tested;
- ✓ List of classes in the cluster;
- ✓ A set of cluster test cases.

Scenario Testing

17. Define Scenario testing.

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

18. What are the methods used to evolve scenarios?

- ✓ System scenarios
- ✓ Use case scenarios

19. What is system scenario?

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

20. Mention the approaches used for developing system scenario.

- ✓ Story Line
- ✓ Life cycle/state transition
- ✓ Deployment / Implementation
- ✓ Business verticals
- ✓ Battle ground

21. What is use case scenario?

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.

Defect Bash Elimination

22. List out the steps involved in defect bash.

Step 1: Choosing the frequency and duration of defect bash.
Step 2: Selecting the right product build.
Step 3: communicating the objective of each defect bash to everyone.
Step 4: Setting up and monitoring the lab for defect bash.
Step 5: Taking actions and fixing issues.
Step 6: Optimizing the effort involved in defect bash.

System Testing

23. What is the need of system testing?

The goal of system testing is to ensure that the system performs according to its requirements.
System test evaluates both functional behavior and quality requirements such as reliability, usability, performance and security. This phase of testing is especially useful for detecting external hardware and software interface defects, for example, those causing race conditions,deadlocks, problems with interrupts and exception handling, and ineffective memory usage.

24. List the types of system testing.

1. Functional testing
2. Performance testing
3. Stress testing
4. Configuration testing
5. Security testing
6. Recovery testing

25. List the effect of security breaches.

- ✓ Loss of information
- ✓ Corruption of information
- ✓ Misinformation
- ✓ Privacy violations
- ✓ Denial of service

26. Give the examples of security testing.

- ✓ Password checking
- ✓ Legal and illegal entry with password
- ✓ Password Expiration
- ✓ Encryption
- ✓ Browsing
- ✓ Trap doors
- ✓ Viruses.

27. What are the objectives of configuration testing?

Configuration testing has the following objectives

- ✓ Show that all the configuration changing commands and menus work properly.
- ✓ Show that all interchangeable devices are really interchangeable, and that they each enter the proper states for the specified conditions.
- ✓ Show that the systems' performance level is maintained when devices are interchanged, or when they fail.

28. What are the goals of functional testing?

- ✓ All types or classes of legal inputs must be accepted by the software.
- ✓ All classes of illegal inputs must be rejected (however, the system should remain available).
- ✓ All possible classes of system output must exercise and examined.
- ✓ All effective system states and state transitions must be exercised and examined.
- ✓ All functions must be exercised.

29. What are the areas to be focused during recovery testing?

Testers focus on the following areas during recovery testing,
Restart. The current system state and transaction states are discarded.
Switchover. The ability of the system to switch to a new processor must be tested.

30. What are the factors that govern performance testing?

- ✓ Throughput
- ✓ Response time
- ✓ Latency
- ✓ Tuning
- ✓ Benchmarking
- ✓ Capacity planning

Acceptance testing

31. What are the criteria for acceptance testing?

- ✓ Product acceptance
- ✓ Procedure acceptance
- ✓ Service level agreements

Performance Testing

32. What are the two major requirements of Performance testing?

- ✓ Functional requirements
- ✓ Quality requirements.

Regression Testing

33. Define Regression testing.

Regression testing is not a level of testing, but it is the retesting of the software that occurs when the changes are made to ensure that the new version of the software has retained the capabilities of the old version and that has no defect have been introduced due to the changes.

Internationalization Testing

34. What are the various tools available for internalization? [May/Jun 2012]

Name of the tools for Microsoft OS	Name of the tool for Linux OS
MS Localization studio	GNU gettext()
http://BabelFish.Altavista.com	http://BabelFish.Altavista.com
MS regional settings	LANG and set of environmental variables
IME	Unicode IME
Htttp://www.snowcrest.net/Donnelly/piglatin.html	http://www.snowcrest.net/donnelly/piglatin.html

Ad-Hoc Testing

35. What are the drawbacks of Ad-hoc testing?

- ✓ Difficult to track exact steps
- ✓ Large number of defects
- ✓ Lack of data for metrics analysis

36. What are the ways used to plan the Ad-hoc testing?

- ✓ After a certain number of planned test cases are executed
- ✓ Prior to planned testing

Alpha – Beta Tests

37. What is the difference between alpha and beta testing? [Nov/ Dec 2009][IT - Nov / Dec 2014 -8M]

Alpha Testing	Beta Testing
It is always performed by the developers at the software development site.	It is always performed by the customers at their own site.

It comes under the category of both White Box Testing and Black Box Testing.	It is only a kind of Black Box Testing.

Testing OO Systems

38. What are the OO concepts relevant to testing?

- ✓ Classes
- ✓ Objects
- ✓ Constructor
- ✓ Encapsulation
- ✓ Inheritance
- ✓ Polymorphism

39. What are the steps involved in Alpha – Omega Method?

Step1: Test the constructor methods first

Step 2:Test the “get” methods or “access” methods

Step 3: Test the methods that modify the object variables

Step 4: Object has to be destroyed and when the object is destroyed

Usability and Accessibility Testing

40.What are the characteristics of usability testing?

- ✓ Ease of use
- ✓ Speed
- ✓ Pleasantness and aesthetics

41.What are the types of messages?

- ✓ Informal message
- ✓ Warning message
- ✓ Error message

42.What are the phases involved in usability testing?

1. Design Validation
2. Usability Testing done as a part of component and integration testing phases of test cycle

43.Mention the quality factors for usability.

1. Comprehensibility
2. Consistency
3. Navigation
4. Responsiveness

44. What are the roles involved in usability testing?

- ✓ Usability architect

- ✓ Usability expert
- ✓ Human factors specialist
- ✓ Graphic designer
- ✓ Usability Manager/ lead
- ✓ Usability Test Engineer

Configuration Testing

45. What is configuration testing?

Configuration testing is the process of checking the operation of the software we are testing with all these various types of hardware. This is usually a dynamic white-box testing and programmer-debugging effort.

46. What are the processes to be done before planning configuration testing?

- ✓ Decide the Types of Hardware You'll Need
- ✓ Decide What Hardware Brands, Models, and Device Drivers Are Available
- ✓ Decide Which Hardware Features, Modes, and Options Are Possible
- ✓ Pare Down the Identified Hardware Configurations to a Manageable Set
- ✓ Identify Your Software's Unique Features That Work with the Hardware Configurations
- ✓ Design the Test Cases to Run on Each Configuration
- ✓ Execute the Tests on Each Configuration
- ✓ Rerun the Tests Until the Results Satisfy Your Team

47. What approach is used for testing the large configuration?

The best approach for testing the system with more combinations of configuration testing is equivalence partition.

Compatibility testing

48. What is compatibility testing?

Software compatibility testing means checking that your software interacts with and shares information correctly with other software. This interaction could occur between two programs simultaneously running on the same computer or even on different computers connected through the Internet thousands of miles apart.

Testing the documentation

49. What are the factors that a good software document consists of?

Good software documentation contributes to the product's overall quality in three ways:

- ✓ **It improves usability:** "Usability Testing," all the issues related to a product's usability? Much of that usability is related to the software documentation.
- ✓ **It improves reliability:** Reliability is how stable and consistent the software is.
- ✓ **It lowers support costs.**

Website Testing

50. What are the approaches used for website testing?

- ✓ White box approach
- ✓ Black box approach
- ✓ Gray box approach (mixture of both)

51. What are the types of tests that Web performance testing should include?

Web performance testing should include

- ✓ Web Load Testing
- ✓ Web Stress Testing

PART B

1. **State and explain different levels of testing [May / Jun 2012- 8M]** *[Refer Pg.no:65]*

   (or)

   Briefly explain the levels of testing with an example. [Nov 2014- 8M]

   (or)

   Describe the activities or tasks and responsibilities for developer or tester in support of multilevel testing? [Nov / Dec 2012- 8M]
2. **List the tasks that must be performed by the developer or tester during the preparation for unit testing. [Nov / Dec 2012- 8M]** *[Refer Pg.no:67]*
3. **Write a short note on Test Harness. [6M]** *[Refer Pg.no:68]*
4. **Explain how to run the unit test and record the results of it. [6M]** *[Refer Pg.no:69]*
5. **Explain the integration test and its design and planning.[Nov 2009 -16M] [IT - Nov / Dec 2016 -10M] [Nov 2017]** *[Refer Pg.no:70]*

   (or)

   Write a detailed note on integration testing. Explain with an example. [IT - Nov / Dec 2014 -8M]

   (or)

   State the need for integration testing in procedural code. [Nov / Dec 2012 – 8M]
6. **Write short notes on Scenario testing. (May/ Jun 2012 -8 M)** *[Refer Pg.no:73]*
7. **Write a short note on Defect Bash Elimination. [6M]** *[Refer Pg.no:75]*
8. **Write the importance of security testing and what are the consequences of security breaches, also write the various areas which has to be focused on during security testing. [Nov / Dec 2012 – 8M]** *[Refer Pg.no:76]* **(or)**

   Explain in detail system testing and its types.

9. **Write a short note on acceptance testing. [8 M]** *[Refer Pg.no:80]*
10. **Write short notes on Performance testing (May/ Jun 2012 -8M)** *[Refer Pg.no:81]*
11. **Write short note on Internationalization testing. [8M]** *[Refer Pg.no:90]*
12. **Compare and contrast regression and AD-Hoc testing (May / Jun 2012 - 8M)** *[Refer Pg.no:88 & 91]*

    (Or)

    What is regression testing? Give its types. When is it necessary to perform regression testing and how is it done? [Nov / Dec 2009 – 16M] *[Refer Pg.no:88]*

    (or)

    Discuss the importance of regression testing when developing a new software release. What items from previous release would be useful to the regression tester? [IT - Nov / Dec 2014 -8M]
13. **Explain in detail Ad-hoc testing with example. [16M]** *[Refer Pg.no:90]*

    (or)

    Compare and contrast regression and AD-Hoc testing (May / Jun 2012 - 8M) *[Refer Pg.no:88 & 91]*
14. **Write a short note on alpha and beta tests. [6M][IT-Nov/Dec 2016] [Nov 2017]** *[Refer Pg.no:94]*
15. **Explain how to test OO system in detail. [Nov 2014 – 6M]** *[Refer Pg.no:95]*
16. **Explain in detail about usability and accessibility testing. [16M]** *[Refer Pg.no:98]*
17. **Explain configuration testing with suitable example. [16M]** *[Refer Pg.no:103]*
18. **Explain in detail about compatibility testing with suitable examples. [16M][April/May 2017-4M]** *[Refer Pg.no:105]*
19. **Explain in detail about testing the documentation. [16M] [April/May 2017-4M]** *[Refer Pg.no:108]*
20. **Explain in detail about website testing. [16M]** *[Refer Pg.no:111]*
21. **Explain types of testing in detail with suitable example. [Nov / Dec 2014-16M]** *[Refer Pg.no:76]*