# 19CS408T – Software Testing

## Part - A

1. **C. Test bed**
2. **D. i, ii, iii**
3. **C. developers**
4. **B. 499, 10001**
5. **B. stubs and drivers**
6. **B. Integration Testing**
7. **D. Test summary specifications**
8. **D. Work breakdown structure**
9. **A. high and low**
10. **B. Azendoo**

## PART _ B

**11. Compare verification and validation.**

Verification is the process of evaluating a software system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase

Validation is the process of evaluating a software system or component during, or at the end of, the development cycle in order to determine whether it satisfies specified requirements

| **Verification** | **Validation** |
|---|---|
| Verification addresses the concern: "Are you building it right?" | Validation addresses the concern: "Are you Building the right thing?" |
| Ensures that the software system meets all the functionality. | Ensures that the functionalities meet the intended behaviour |

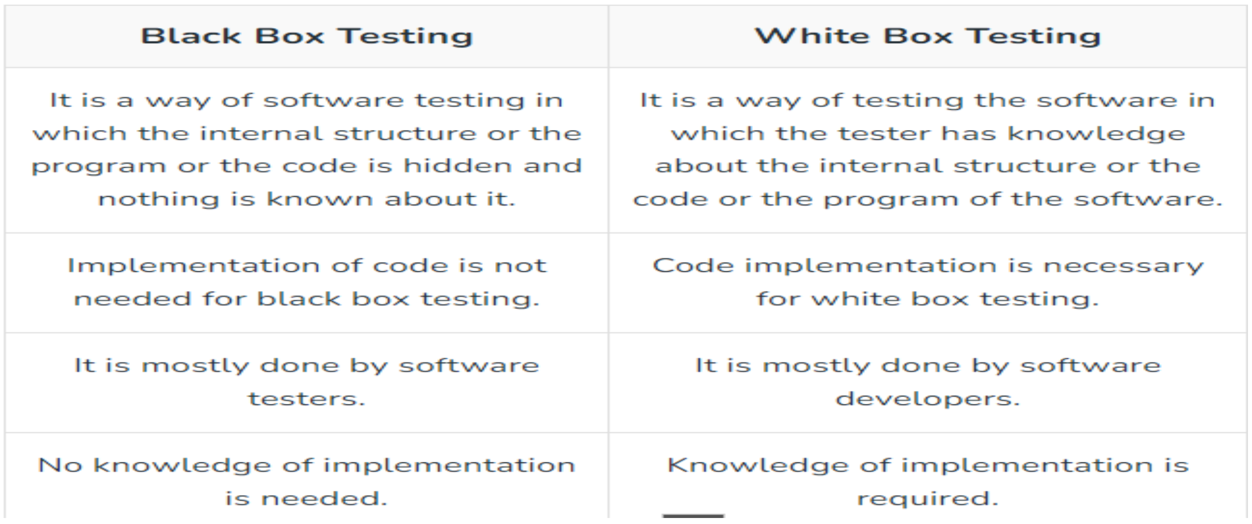
**12. Why it is important to test a software?**

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

A successful test is one which uncovers an undetected error.

The principal goal of testing is to ensure that users will not be negatively affected by any software bug. Those bugs are software errors that cause unexpected or incorrect results. There are many kinds of bugs that we can divide into various categories

**13. Compare White box and black box testing.**



| Black Box Testing | White Box Testing |
|---|---|
| It is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it. | It is a way of testing the software in which the tester has knowledge about the internal structure or the code or the program of the software. |
| Implementation of code is not needed for black box testing. | Code implementation is necessary for white box testing. |
| It is mostly done by software testers. | It is mostly done by software developers. |
| No knowledge of implementation is needed. | Knowledge of implementation is required. |

**14. What is domain testing? Give ex.**

Domain testing is the ability to design and execute test cases that relate to the people who will buy and use the software. It helps in understanding the problem they are trying to solve and the ways in which they are using the software to solve them. Domain testing involves testing the product, not by going through the logic built into the product.

**15. Recall levels of testing.**

There are four levels of software testing,

- Unit Testing . Integration Testing
- System Testing Acceptance Testing

**16. Differentiate alpha and Beta testing**

| **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.** |

**17. Where you focus on values that are significantly far away from these limits. For the given scenario of an amount text field accepting values only from 500 to 1000, let's identify test cases for both Boundary Limit Testing (BLT) and Additional Boundary Testing (ABT).**

**Boundary Limit Testing (BLT):**

**Lower Bound Testing (BLB):**

Test Case 1: Input 499 (One less than the lower bound) - Should reject the input.
Test Case 2: Input 500 (Lower bound) - Should accept the input.
Test Case 3: Input 501 (One more than the lower bound) - Should accept the input.

**Upper Bound Testing (ALB):**

SSSSSAASSSSS
Test Case 4: Input 999 (One less than the upper bound) - Should accept the input.
Test Case 5: Input 1000 (Upper bound) - Should accept the input.
Test Case 6: Input 1001 (One more than the upper bound) - Should reject the input.

**18. Define the term goal of an organization.**

The foremost objective of software testing is to identify and eliminate defects or bugs within the application. These defects can range from minor glitches to critical errors that may lead to system crashes or incorrect data processing.

**19. What is mean by product oriented test automation?**

In product-oriented test automation, the emphasis is on a specific software product line so that it can be used in testing its related testing activities.

20. **Recall the use of function library in keyboard driven automation framework**

Keyword Driven Framework is an active automation testing frame that allocates test cases into four distinct fractions to segregate coding from test cases and test steps for satisfactory automation. The keyword-driven testing framework distributes the test cases as steps of tests, issues of test steps, data for test objects, and efforts on test steps for better understanding.

## PART – C

### 21.a.SOFTWARE TESTING PRINCIPLES

Testing principles are important to test specialists and engineers because they are the foundation for developing testing knowledge and acquiring testing skills. They also provide guidance for defining testing activities.

**Principle 1: Testing is the process of exercising a software component using a selected set of test cases, with the intent of revealing defects, and evaluating quality.**

❖ This principle supports testing as an execution-based activity to detect defects. It also supports the separation of testing from debugging since the intent of debugging is to locate defects and repair the software.

❖ The term ―software component‖ means any unit of software ranging in size and complexity from an individual procedure or method, to an entire software system.

❖ The term ―defects‖ represents any deviations in the software that have a negative impact on its functionality, performance, reliability, security, and/or any other of its specified quality attributes.

**Principle 2: When the test objective is to detect defects, then a good test case is one that has a high probability of revealing yet undetected defects.**

❖ Testers must carry out testing in the same way as scientists carry out experiments.

❖ Testers need to create a hypothesis and work towards proving or disproving it, it means he/she must prove the presence or absence or a particular type of defect.

**Principle 3: Test results should be inspected meticulously.**

❖ Testers need to carefully inspect and interpret test results. Several erroneous and costly scenarios may occur if care is not taken.

❖ A failure may be overlooked, and the test may be granted a ―pass‖ status when in reality the software has failed the test.

❖ Testing may continue based on erroneous test results.

❖ The defect may be revealed at some later stage of testing, but in that case it may be more costly and difficult to locate and repair.

**Principle 4: A test case must contain the expected output or result.**

❖ The test case is of no value unless there is an explicit statement of the expected outputs or results. Expected outputs allow the tester to determine

✓ Whether a defect has been revealed,

✓ Pass/fail status for the test.

❖ It is very important to have a correct statement of the output so that time is not spent due to misconceptions about the outcome of a test.

❖ The specification of test inputs and outputs should be part of test design activities.

**Principle 5: Test cases should be developed for both valid and invalid input conditions.**

❖ A tester must not assume that the software under test will always be provided with valid inputs. Inputs may be incorrect for several reasons.

❖ Software users may have misunderstandings, or lack information about the nature of the inputs
❖ They often make typographical errors even when complete/correct information is available.
❖ Devices may also provide invalid inputs due to erroneous conditions and malfunctions.
**Principle 6: The probability of the existence of additional defects in a software component is proportional to the number of defects already detected in that component.**
❖ The higher the number of defects already detected in a component, the more likely it is to have additional defects when it undergoes further testing.
❖ If there are two components A and B, and testers have found 20 defects in A and 3 defects in B, then the probability of the existence of additional defects in A is higher than B.
**Principle 7: Testing should be carried out by a group that is independent of the development group**.
❖ This principle is true for psychological as well as practical reasons. It is difficult for a developer to admit that software he/she has created and developed can be faulty.
❖ Testers must realize that
✓ Developers have a great pride in their work,
✓ Practically it is difficult for the developer to conceptualize where defects could be found.
Principle 8: Tests must be repeatable and reusable.
❖ The tester needs to record the exact conditions of the test, any special events that occurred, equipment used, and a carefully note the results.
❖ This information is very useful to the developers when the code is returned for debugging so that they can duplicate test conditions.
❖ It is also useful for tests that need to be repeated after defect repair.
**Principle 9: Testing should be planned.**
❖ Test plans should be developed for each level of testing. The objective for each level should be described in the associated plan. The objectives should be stated as quantitatively as possible.
**Principle 10: Testing activities should be integrated into the software life cycle.**
❖ Testing activity should be integrated into the software life cycle starting as early as in the requirements analysis phase, and continue on throughout the software life cycle in parallel with development activities.
**Principle 11: Testing is a creative and challenging task.**
❖ Difficulties and challenges for the tester include the following:
❖ A tester needs to have good knowledge of the software engineering discipline.
❖ A tester needs to have knowledge from both experience and education about software specification, designed, and developed.
❖ A tester needs to be able to manage many details.
❖ A tester needs to have knowledge of fault types and where faults of a certain type might occur in code construction.
❖ A tester needs to reason like a scientist and make hypotheses that relate to presence of specific types of defects.
❖ A tester needs to have a good understanding of the problem domain of the software that he/she is testing. Familiarly with a domain may come from educational, training, and work related experiences. A tester needs to create and document test cases.
A tester needs to plan for testing and allocate proper resources.
❖ A tester needs to execute the tests and is responsible for recording results.
❖ A tester needs to analyse test results and decide on success or failure for a test.
❖ This involves understanding and keeping track of huge amount of detailed information.
❖ A tester needs to learn to use tools and keep updated of the newest test tools.
❖ A tester needs to work and cooperate with requirements engineers, designers, and developers, and often must establish a working relationship with clients and users.
❖ A tester needs to be educated and trained in this specialized area.

**(or)**

**21.b. Explain in detail the processing and monitoring of the defects with defect repository.**
To increase the effectiveness of their testing and debugging processes, software organizations need to initiate the creation of a defect database, or defect repository.
❖ The defect repository supports storage and retrieval of defect data from all projects in a centrally accessible location.
❖ A requirement for repository development should be a part of testing and/or debugging policy statements.
❖ A defect repository can help in implementing several TMM maturity goals including

✓ Controlling and monitoring of test,
✓ Software quality evaluation and control,
✓ Test measurement,
✓ Test process improvement.
❖ Defect monitoring should be done for each on-going project. The distribution of defects will change when changes are made to the process.

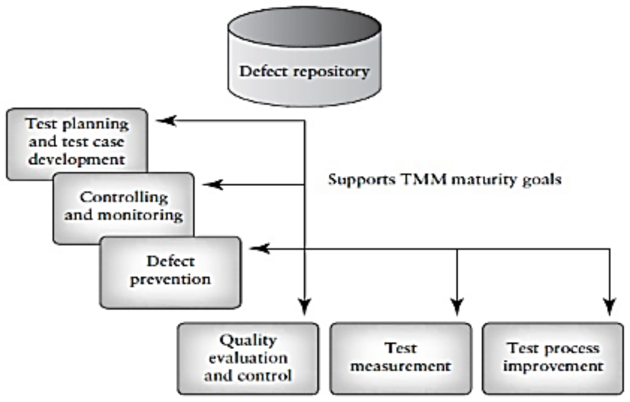

Defect repository
Test planning and test case development
Controlling and monitoring
Defect prevention
Supports TMM maturity goals
Quality evaluation and control
Test measurement
Test process improvement

***Figure 1.7: The Defect Repository and Support for TMM Maturity Goals***

- The defect data is useful for test planning.

❖ It is a TMM level 2 maturity goal.
❖ It helps a tester to select applicable testing techniques, design the test cases, and allocate the amount of resources needed to detect and remove defects.
❖ This allows tester to estimate testing schedules and costs.
❖ The defect data can support debugging activities also.

22.a. suppose a testing module have three separate conditions that apply to the input:
i. it must consist of alphanumeric characters
ii. The range for the total number of characters in between 3 and 15
iii. The first two characters must be letters. Identify the valid and invalid test cases by using equivalence class portioning and boundary value analysis.

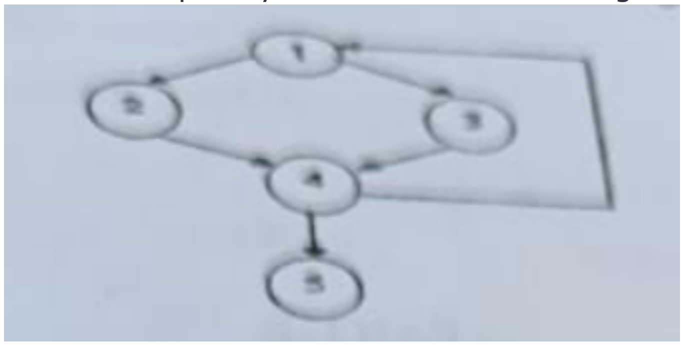
Let's break down the three conditions and use equivalence class partitioning along with boundary value analysis to identify valid and invalid test cases:

Conditions:
i. Input must consist of alphanumeric characters.
ii. The total number of characters should be between 3 and 15.
iii. The first two characters must be letters.

Equivalence Class Partitioning:
Condition 1: Alphanumeric Characters
Valid Equivalence Class: Alphanumeric characters (letters A-Z, a-z, and digits 0-9).
Invalid Equivalence Class: Non-alphanumeric characters (symbols, special characters).
Condition 2: Total Number of Characters
Valid Equivalence Class: 3 to 15 characters.
Invalid Equivalence Class: Less than 3 characters and more than 15 characters.
Condition 3: First Two Characters are Letters
Valid Equivalence Class: First two characters are letters (A-Z or a-z).
Invalid Equivalence Class: First two characters are not letters (digits, symbols, special characters).
Boundary Value Analysis:

Condition 1: Alphanumeric Characters
Valid Boundary Values: Alphanumeric characters like "A1", "abC123"
Invalid Boundary Values: Non-alphanumeric characters like "@$", "*%"
Condition 2: Total Number of Characters
Valid Boundary Values: Strings with 3, 15 characters.
Invalid Boundary Values: Strings with < 3 characters or > 15 characters.
Condition 3: First Two Characters are Letters
Valid Boundary Values: Strings like "ab...", "BC..."
Invalid Boundary Values: Strings starting with digits, symbols, or non-letter characters.
Valid Test Cases:
"Abc123" (Valid for all conditions)
"XY123456789012" (Valid for conditions 1 and 2)
"AAabcdef" (Valid for all conditions)
"A1$" (Valid for condition 1, invalid for conditions 2 and 3)
"ABCDEF" (Valid for conditions 1 and 3)
Invalid Test Cases:
"12@XYZ" (Invalid for all conditions)
"Ab" (Invalid for conditions 2 and 3)
"XY" (Invalid for conditions 1 and 3)
"@@abc123" (Invalid for condition 1)
By using equivalence class partitioning and boundary value analysis, you can identify test cases that cover both the valid and invalid scenarios based on the specified conditions for the input in the testing module.

22.b.ii) write the steps to to convert a program to convert flow graph. Apply the significance of control flow graph graphs to identify the number of nodes, edges, prediction and cyclometric complexity values for the flow diagram



The control flow graph aids in determining various metrics like the number of nodes, edges, and cyclomatic complexity. Here are the steps and the significance of the control flow graph metrics:
Steps to Convert a Program into a Control Flow Graph:
**Understand the Program Structure:**
Gain a thorough understanding of the program's structure, including loops, conditionals, function calls, and control flow constructs.
**Identify Basic Blocks:**
Break down the program into basic blocks, usually delimited by control flow statements like if-else, loops, or function calls.
**Create Nodes for Basic Blocks:**
Represent each basic block as a node in the control flow graph. Nodes represent individual statements or blocks of code.
**Identify Control Flow Edges:**
Determine the flow of control between basic blocks. Connect nodes with directed edges (arrows) to represent the flow of execution.
**Add Conditional Branches and Loops:**
Include decision nodes for conditional statements (if-else) and loop nodes to depict loops in the flow graph. Connect these nodes accordingly.

**Analyze the Control Flow Graph:**
Review and validate the flow graph to ensure all program paths are represented accurately. Validate that each node has appropriate connections and no isolated sections exist.
Significance of Control Flow Graph Metrics:
**Number of Nodes (N):**
Represents the total number of nodes in the control flow graph, indicating the number of statements or blocks in the program.
**Number of Edges (E):**
Indicates the total number of edges in the control flow graph, showing the transitions between nodes. It helps understand the flow of execution.
**Cyclomatic Complexity (V(G)):**
Calculated using various formulas (e.g., V(G) = E - N + 2P, where P is the number of connected components). Cyclomatic complexity measures the number of linearly independent paths through the program and is an indicator of program complexity and testing effort.
**Decision Nodes (P):**
Represents the number of decision points or conditional statements in the control flow graph. Each decision node adds complexity and contributes to the cyclomatic complexity value.
Importance of Metrics:
Understanding Program Structure: N and E help understand the structure and flow of the program.
Complexity Analysis: Cyclomatic complexity (V(G)) assesses code complexity, aiding in testing and maintenance efforts.
So, **cyclomatic complexity M** would be defined as,

**M = E – N + 2P** where E = the number of edges in the control flow graph
N = the number of nodes in the control flow graph
P = the number of connected components
6-5+2=1

**23. a. System testing is a type of black box testing – Interpret the statement and justify the same with example**

The statement "System testing is a type of black box testing" is accurate and represents a common approach in software testing methodologies. Black box testing focuses on examining the functionality of a system without considering its internal structure or implementation details. System testing, as a phase in the software testing lifecycle, often adopts this black box testing approach.
Justification with an Example:
Consider a scenario where a software application, let's say an e-commerce platform, undergoes system testing before its release. The testing team is concerned with assessing the entire system's behavior as a whole, without delving into its internal code structure.

Example Steps in System Testing:
Functionality Testing:

Black Box Approach: Testers interact with the e-commerce platform just like end-users would. They perform various actions (e.g., browsing products, adding items to the cart, making purchases) without examining the source code. They validate whether the system behaves as expected, without knowledge of the internal coding logic.
Usability Testing:

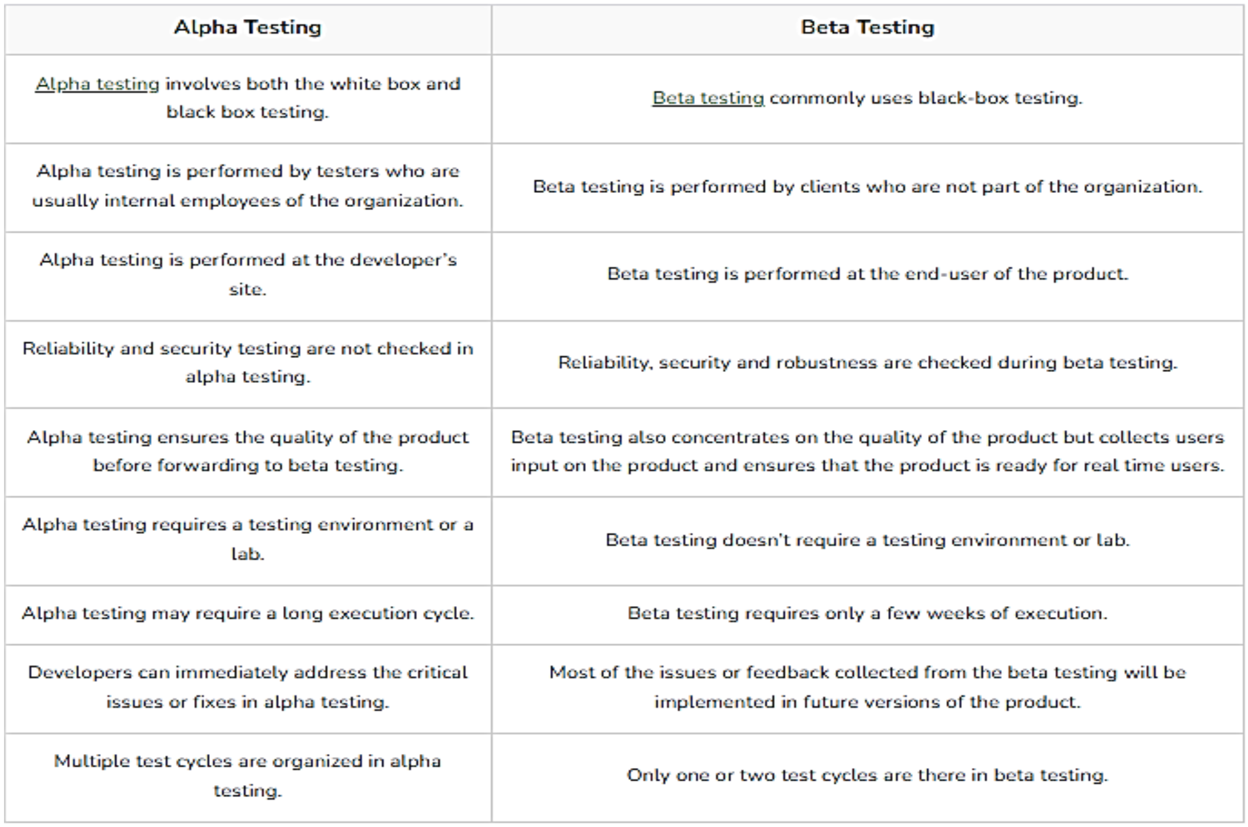
Black Box Approach: Testers evaluate the user interface, navigation, responsiveness, and overall user experience. They assess how easy or intuitive it is for a user to perform tasks within the platform without considering the underlying code structure.
Compatibility Testing:

Black Box Approach: Testers check whether the system functions correctly across different browsers, devices, or operating systems. They validate its compatibility without knowing the intricacies of how the system is implemented.
Performance Testing:

Black Box Approach: Testers analyze the system's response time, resource utilization, and overall performance metrics under various load conditions. They measure the system's performance without needing to understand the internal coding or algorithms.
Justification:
In each of these testing phases within system testing, the testers focus on evaluating the system's functionalities, usability, compatibility, and performance without access to the system's internal structures, code, or algorithms. The testing is solely based on external inputs, system outputs, and observed behaviors, aligning with the principles of black box testing.

Hence, system testing is considered a type of black box testing as it concentrates on verifying the system's behavior and functionality without considering its internal implementation details.

**23. b. Compare Alpha and Beta Testing.**



| Alpha Testing | Beta Testing |
|---|---|
| Alpha testing involves both the white box and black box testing. | Beta testing commonly uses black-box testing. |
| Alpha testing is performed by testers who are usually internal employees of the organization. | Beta testing is performed by clients who are not part of the organization. |
| Alpha testing is performed at the developer's site. | Beta testing is performed at the end-user of the product. |
| Reliability and security testing are not checked in alpha testing. | Reliability, security and robustness are checked during beta testing. |
| Alpha testing ensures the quality of the product before forwarding to beta testing. | Beta testing also concentrates on the quality of the product but collects users input on the product and ensures that the product is ready for real time users. |
| Alpha testing requires a testing environment or a lab. | Beta testing doesn't require a testing environment or lab. |
| Alpha testing may require a long execution cycle. | Beta testing requires only a few weeks of execution. |
| Developers can immediately address the critical issues or fixes in alpha testing. | Most of the issues or feedback collected from the beta testing will be implemented in future versions of the product. |
| Multiple test cycles are organized in alpha testing. | Only one or two test cycles are there in beta testing. |

23.b. i) What is meant by Testing policy? Outline a sample testing policy statement appropriate for a Testing Maturity Model (TMM) Level 2 organizations
**Testing policy** is used to ensure the right direction of the software testing process and its effectiveness. Priority goals also include to verify, validate and estimate the quality of the product under examination so it could meet the expectations of end-users.
It provides direction and sets the standards for testing activities within an organization.
defining processes and standards. The testing policy should reflect this by establishing fundamental guidelines and expectations for testing activities.

Sample Testing Policy Statement for TMM Level 2:
Title: Testing Policy

Purpose:

Scope:

Policy Statements:

Quality Objective:

This testing policy will be reviewed periodically by the Testing Governance Body or designated authority for any necessary updates or modifications.
Effective Date:

Date:
The above sample testing policy statement is designed to guide the organization in maintaining consistent testing practices, aligning with TMM Level 2 objectives of establishing standardized processes. The policy sets the foundation for structured testing methodologies and emphasizes the importance of quality in software development.

**b. ii) Explain the various test plan components described under IEEE std**

❖ Test Plan identifier
❖ Introduction
❖ Items to be tested
❖ Features to be tested
❖ Pass/Fail criteria
❖ Suspension & Resumption criteria
❖ Testing tasks
❖ Test environment
❖ Risks & Contingencies
❖ Testing costs
❖ Approvals

24. a. Illustrate the various components of test plan  with an example.
Test Plan Identifier
❖ Each test plan should have a unique identifier so that it can be associated with a specific project and become a part of the project history. The project history and all project-related items should be stored in a project database or come under the control of a configuration management system.
Introduction
❖ In this section the test planner gives an overall description of the project, the software system being developed or maintained, and the software items and/or features to be tested. It is useful to include a high-level description of testing goals and the testing approaches to be used.
Items to Be Tested
❖ This is a listing of the entities to be tested and should include names, identifiers, and version/revision numbers for each entity.
❖ The items listed could include procedures, classes, modules, libraries, subsystems, and systems. References to the appropriate documents where these items and their behaviors are described such as requirements and design documents, and the user manual should be included in this component of the test plan.
❖ These references support the tester with traceability tasks.
Features to Be Tested
❖ In this component of the test plan the tester gives another view of the entities to be tested by describing them in terms of the features they encompass. Features may be described as distinguishing characteristics of a software component or system.
Approach
❖ This section of the test plan provides broad coverage of the issues to be addressed when testing the target software. Testing activities are described. The level of descriptive detail should be sufficient so that the major testing tasks and task durations can be identified.
Item Pass/Fail Criteria
❖ Given a test item and a test case, the tester must have a set of criteria to decide on whether the test has been passed or failed upon execution.
❖ The master test plan should provide a general description of these criteria. In the test design specification section more specific details are given for each item or group of items under

test with that specification.
Suspension and Resumption Criteria
❖ In this section of the test plan, criteria to suspend and resume testing are described. In the simplest of cases testing is suspended at the end of a working day and resumed the following morning. For some test items this condition may not apply and additional details need to be provided by the test planner.
Test Deliverables
❖ Execution-based testing has a set of deliverables that includes the test plan along with its associated test design specifications, test procedures, and test cases. The latter describe the actual test inputs and expected outputs.
Testing Tasks
❖ In this section the test planner should identify all testing-related tasks and their dependencies. Using a Work Breakdown Structure (WBS) is useful here. A Work Breakdown Structure is a hierarchical or treelike representation of all the tasks that are required to complete a project.
The Testing Environment
❖ Here the test planner describes the software and hardware needs for the testing effort. The planner must also indicate any laboratory space containing the equipment that needs to be reserved.
Responsibilities
❖ The staff who will be responsible for test-related tasks should be identified. This includes personnel who will be:
• Transmitting the software-under-test;
• Developing test design specifications, and test cases;
• Executing the tests and recording results;
• Tracking and monitoring the test efforts;
• Checking results;
• Interacting with developers;
• Managing and providing equipment;
• Developing the test harnesses; and Interacting with the users/customers Staffing and Training Needs
❖ The test planner should describe the staff and the skill levels needed to carry out test-related responsibilities such as those listed in the section above. Any special training required to perform a task should be noted.
Scheduling
❖ Task durations should be established and recorded with the aid of a task networking tool. Test milestones should be established, recorded, and scheduled.
Risks and Contingencies
❖ Every testing effort has risks associated with it. Testing software with a high degree of criticality, complexity, or a tight delivery deadline all impose risks that may have negative impacts on project goals. These risks should be identified, evaluated in terms of their probability of occurrence, prioritized, and contingency plans should be developed that can be activated if the risk occurs.
Testing Costs
❖ The IEEE standard for test plan documentation does not include a separate cost component in its specification of a test plan. This is the usual case for many test plans since very often test costs are allocated in the overall project management plan. Test costs that should included in the plan are:
• Costs of planning and designing the tests;
• Costs of acquiring the hardware and software necessary for the tests;
• Costs to support the test environment;
• Costs of executing the tests;
• Costs of recording and analyzing test results;
• Tear-down costs to restore the environment.

24.b.ii) Discuss in detail the various skills needed for a test specialist.
❖ Because of the nature of technical and managerial responsibilities assigned to the tester, many managerial and personal skills are necessary for success in the area of work.
❖ On the personal and managerial level a test specialist must have:

• Organizational, and planning skills;
• The ability to keep track of, and pay attention to, details;
• The determination to discover and solve problems;
• The ability to work with others and be able to resolve conflicts;
• The ability to mentor and train others;
• The ability to work with users and clients;
• Strong written and oral communication skills;
he ability to work in a variety of environments;
• The ability to think creatively
❖ On the technical level testers need to have:
• An education that includes an understanding of general software engineering principles, practices, and methodologies;
• Strong coding skills and an understanding of code structure and behavior;
• A good understanding of testing principles and practices;
• A good understanding of basic testing strategies, methods, and techniques;
• The ability and experience to plan, design, and execute test cases and test procedures on multiple levels (unit, integration, etc.);
• A knowledge of process issues;
• Knowledge of how networks, databases, and operating systems are organized and how they work;
• A knowledge of configuration management;
• A knowledge of test-related documents and the role each documents plays in the testing process;
• The ability to define, collect, and analyze test-related measurements;
• The ability, training, and motivation to work with testing tools and equipment;
• Knowledge of quality issues.
❖ Testers must have knowledge of both white and black box techniques and methods and the ability to use them to design test cases.
❖ Organizations need to realize that this knowledge is a necessary prerequisite for tool use and test automation.
❖ The list of skills and knowledge requirements needed to be a successful test specialist is long and complex.
❖ Acquiring these skills takes education, training, experience, and motivation. Organizations must be willing to support such staff since they play a valuable role in the organizational structure and have a high impact on the quality of the software delivered

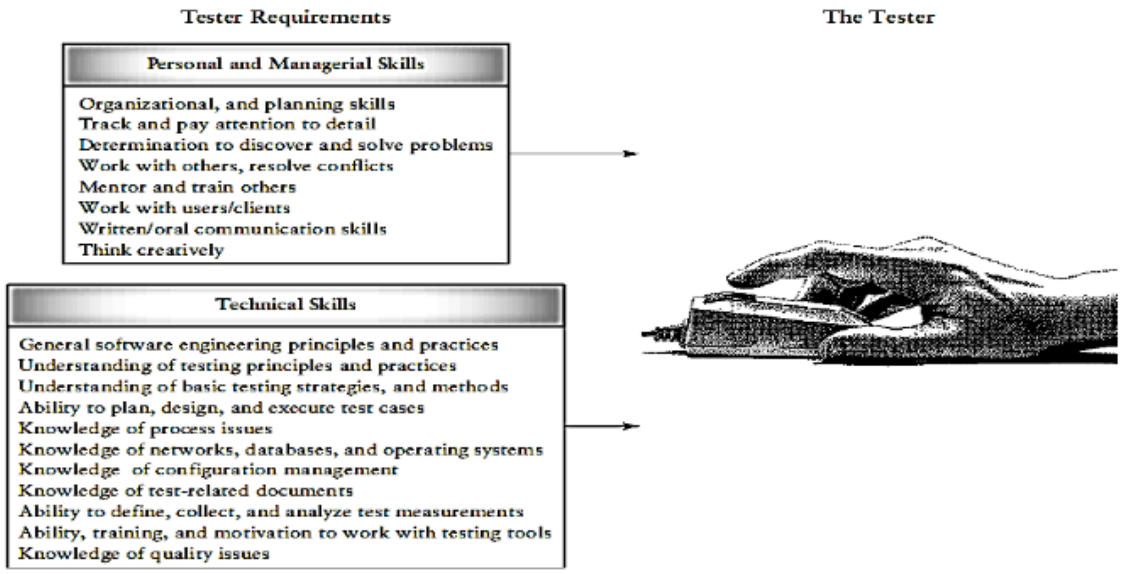
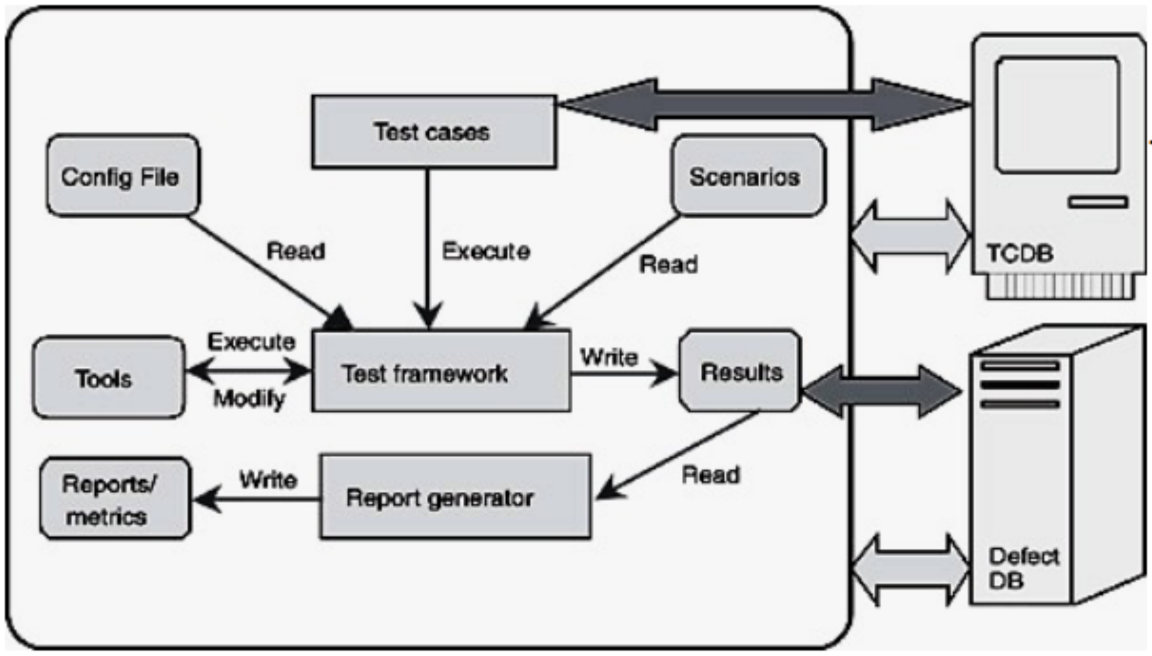

Tester Requirements
The Tester
Personal and Managerial Skills
Organizational, and planning skills
Track and pay attention to detail
Determination to discover and solve problems
Work with others, resolve conflicts
Mentor and train others
Work with users/clients
Written/oral communication skills
Think creatively
Technical Skills
General software engineering principles and practices
Understanding of testing principles and practices
Understanding of basic testing strategies, and methods
Ability to plan, design, and execute test cases
Knowledge of process issues
Knowledge of networks, databases, and operating systems
Knowledge of configuration management
Knowledge of test-related documents
Ability to define, collect, and analyze test measurements
Ability, training, and motivation to work with testing tools
Knowledge of quality issues

***Figure 4.5: Skills needed for Test Specialist***

**25.a. Different types of test automation framework**
The work on automation can go simultaneously with product development and can overlap with

multiple releases of the product.
❖ Like multiple-product releases, automation also has several releases. One specific requirement for automation is that the delivery of the automated tests should be done before the test execution phase so that the deliverables from automation effort can be utilized for the current release of the product.
❖ The requirements for automation span multiple phases for multiple releases, like product requirements.
❖ Test execution may stop soon after releasing the product but automation effort continues after a product release



Selecting a Test Tool:
❖ Having identified the requirements of what to automate, a related question is the choice of an appropriate tool for automation. Even though we have listed these as two sequential steps, oftentimes, they are tightly interlinked.
Automation for Extreme Programming Model:
❖ Unit test cases are developed before coding phase starts;
❖ Code is written for test cases and are written to ensure test cases pass;
❖ All unit tests must run 100% all the time.
❖ Everyone owns the product; they often cross boundaries.
Modules:
❖ External Modules
❖ Scenario and configuration file modules
❖ Test cases and test framework modules
❖ Tools and results modules
❖ Report generator and report / metrics modules
External Modules
❖ There are two modules that are external modules to automation TCDB and defect DB, all the test cases, the steps to execute them and the history of their execution are stored in the TCDB.
❖ The test cases in TCDB can be manual or automated. The interfaces are shown by thich arrows represents the interaction between TCDB and the automation framework onl for automated test cases.
❖ Defect DB on defect database or defect repository contains details of all the defects that are found in various products that are tested in a particular organization.
Scenario and configuration file modules
❖ Scenarios are nothing but information on ―how to execute a particular test case‖.
❖ A configuration file contains a set of variables that are used in automation.
❖ The values of variables in this configuration file can be changed dynamically to achieve different execution, input, output and state conditions.
Test cases and test framework modules

❖ A test case means the automated test cases taken from TCDN and executed by the framework. A test case is an object for execution for other modules in the architecture and does not represent any interaction by itself.
❖ A test framework is a module that combines ―what to execute‘ and ―how they have to be executed‖. It
picks up the specific test cases that are automated from TCDB and picks up the scenarios and executes them.
Tools and results modules
❖ When a test framework performs its operations, there are a set of tools that may be required.
Report Generator and Report / Metrics Modules
❖ The module that takes the necessary inputs and prepares a formatted report is called a report generator.
❖ All the reports and metrics that are generated are stored in the reports/metrics module of automation for future use and analysis.

**25.b.**
Generic Requirements for Test Tool/Framework
❖ While illustrating the requirements, we have used examples in a hypothetical metal language to drive home the concept.
**Requirement 1: No Hard Coding In the Test Suite**
**Requirement 2: Test case / suite expandability**
**Requirement 3: Reuse Of Code For Different Types Of Testing, Test Cases.**
**Requirement 4: Automatic Set Up and Clean Up**
**Requirement 5: Independent Test Cases**
**Requirement 6: Test Case Dependency**
**Requirement 7: Insulating Test Cases during Execution**
**Requirement 8: Coding Standards and Directory Structures.**
**Requirement 11: Parallel Execution of Test Cases**
**Requirement 12: Looping the Test Cases.**
**Requirement 13: Grouping Of Test Scenarios.**
**Requirement 14: Test Cases Execution Based On Previous Results**
**Requirement 15: Remote Execution of Test Cases.**
**Requirement 16: Automatic Archival Of Test Data**
**Requirement 17: Reporting Scheme**
**Requirement 18: Independent of Language**
**Requirement 19: Portability to Different Platforms.**