Software Testing Imp

Q1. Short Answer type questions.

1. **What is Software Testing?**
   Software Testing is the process of evaluating a system or its components to verify whether it meets the specified requirements and to identify defects.
2. **What is Static Testing?**
   Static Testing is a type of testing that involves reviewing documents, code, or design without executing the program to find defects early in the development process.
3. **State the advantages of manual testing.**
   - Allows for human observation and intuition to identify usability issues.
   - Cost-effective for small projects or when automation is not feasible.
4. **What are the formulae for calculating Cyclomatic Complexity?**
   Cyclomatic Complexity (V(G)) is calculated as:
   - V(G) = E - N + 2, where E = number of edges, N = number of nodes.
   - Alternatively, V(G) = P + 1, where P = number of decision points.
5. **What is Gray-box Testing?**
   Gray-box Testing is a combination of black-box and white-box testing where the tester has partial knowledge of the internal structure of the application.
6. **Define Validation Testing.**
   Validation Testing ensures that the software meets the user's needs and requirements and that it performs its intended functions correctly.

7. **What is Debugging?**
   Debugging is the process of identifying, analyzing, and fixing defects or issues in the software code.

8. **Explain the terms: Error, Fault, and Failure.**
   - **Error:** A mistake made by a developer in the code.
   - **Fault:** A defect in the software caused by an error.
   - **Failure:** The inability of the system to perform its intended function due to a fault.

9. **Define Regression Testing.**
   Regression Testing is performed to ensure that recent changes or updates in the software have not adversely affected existing functionality.

10. **What is a Software Metric?**
    A Software Metric is a quantitative measure used to assess the quality, performance, or progress of a software project.

11. **Explain the term Performance Testing.**
    Performance Testing evaluates the speed, responsiveness, stability, and scalability of a software application under a given workload.

12. **Define the Big Bang Approach (integration testing).**
    The Big Bang Approach is an integration testing method where all modules are combined and tested as a single unit after individual module testing.

13. **Define Failure and Defect.**
    - **Failure:** The deviation of the software from its expected behavior.
    - **Defect:** A flaw or imperfection in the software that causes it to fail.

14. **Define Verification Testing.**
Verification Testing ensures that the software is built according to the specified requirements and design.

15. **What is the Sandwich Approach in integration testing?**
The Sandwich Approach combines top-down and bottom-up integration testing methods to test the system in layers.

16. **What is Structural Testing?**
Structural Testing, also known as white-box testing, focuses on testing the internal structure, logic, and code of the software.

17. **How is Cyclomatic Complexity calculated?**
Cyclomatic Complexity is calculated using the formula:
$V(G) = E - N + 2$, where E = number of edges, N = number of nodes.

18. **Explain types of Acceptance Testing.**
    - **User Acceptance Testing (UAT):** Ensures the software meets user requirements.
    - **Business Acceptance Testing (BAT):** Validates the software against business needs.
    - **Alpha/Beta Testing:** Conducted in a controlled or real-world environment.

19. **What is User Documentation Testing?**
User Documentation Testing verifies that the user manuals, guides, and help documents are accurate, complete, and easy to understand.

20. **Define Software Quality Assurance (SQA).**
SQA is a set of activities designed to ensure that the software development process adheres to quality standards and produces a high-quality product.

21. **Define Winrunner (a testing tool).**
Winrunner is an automated functional testing tool used to test applications by recording and replaying user interactions.

22. **What is a Test Case Design?**
Test Case Design is the process of creating detailed test scenarios, including inputs, execution steps, and expected outcomes, to validate software functionality.

23. **What is Dynamic Testing?**
Dynamic Testing involves executing the software to evaluate its behavior and performance under various conditions.

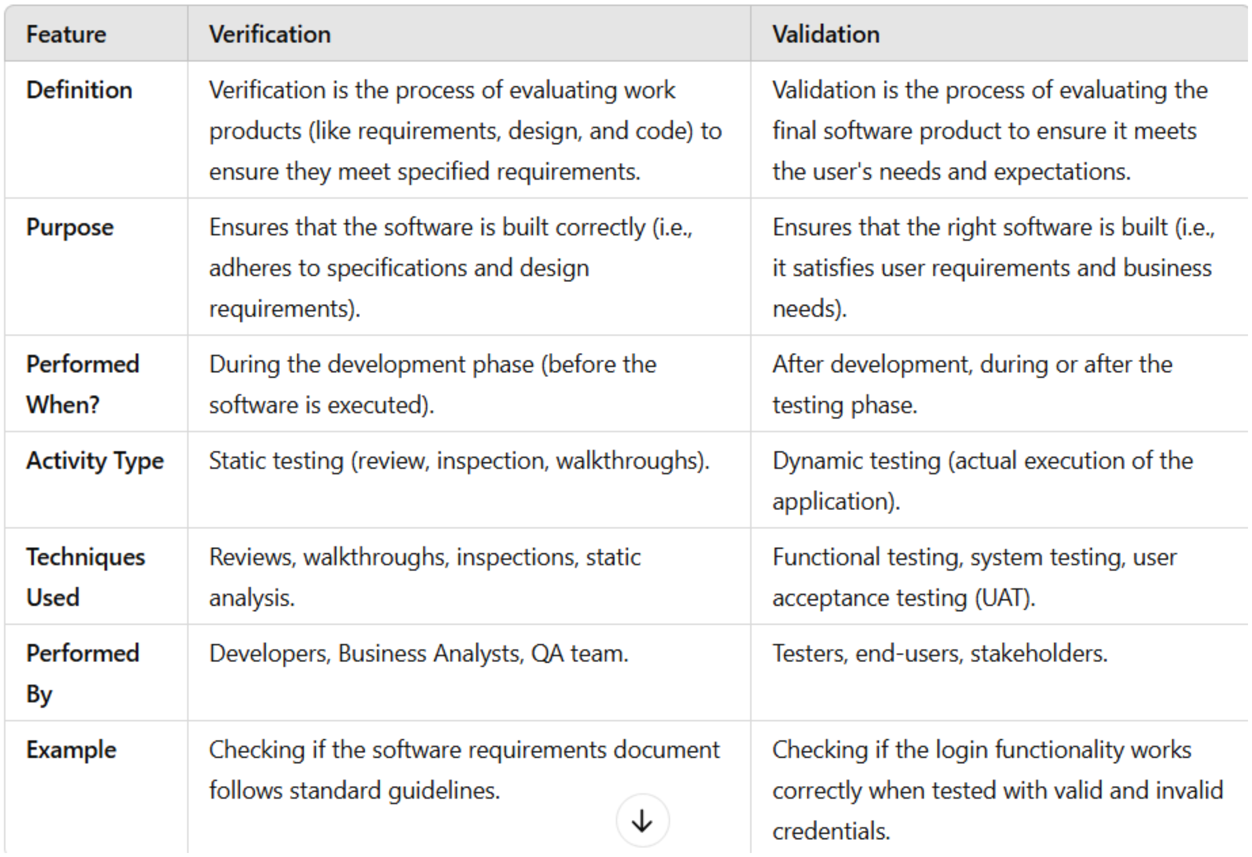
24. **Define Software Review.**
A Software Review is a formal or informal evaluation process where software artifacts (e.g., code, design) are examined to identify defects or improvements.

25. **What is Load Testing?**
Load Testing evaluates the software's performance under expected or peak load conditions to ensure it can handle the required workload.

26. **Define Stub & Driver (in integration testing).**
   - **Stub:** A temporary module used to simulate the behavior of a lower-level module that is not yet developed.
   - **Driver:** A temporary module used to simulate the behavior of a higher-level module that calls the module under test.

Q2. Answer in long.

1. Write the difference between verification and validation.



Feature	Verification	Validation
Definition	Verification is the process of evaluating work products (like requirements, design, and code) to ensure they meet specified requirements.	Validation is the process of evaluating the final software product to ensure it meets the user's needs and expectations.
Purpose	Ensures that the software is built correctly (i.e., adheres to specifications and design requirements).	Ensures that the right software is built (i.e., it satisfies user requirements and business needs).
Performed When?	During the development phase (before the software is executed).	After development, during or after the testing phase.
Activity Type	Static testing (review, inspection, walkthroughs).	Dynamic testing (actual execution of the application).
Techniques Used	Reviews, walkthroughs, inspections, static analysis.	Functional testing, system testing, user acceptance testing (UAT).
Performed By	Developers, Business Analysts, QA team.	Testers, end-users, stakeholders.
Example	Checking if the software requirements document follows standard guidelines.	Checking if the login functionality works correctly when tested with valid and invalid credentials.

2. Explain the software testing life cycle with a diagram.

The Software Testing Life Cycle (STLC) consists of several phases:

1. **Requirement Analysis**: Understanding and analyzing testing requirements.
2. **Test Planning**: Defining the scope, approach, resources, and schedule for testing activities.
3. **Test Case Design**: Creating detailed test cases based on requirements.
4. **Test Environment Setup**: Preparing the environment where testing will be conducted.

5. **Test Execution**: Running the test cases and logging defects.
6. **Test Closure**: Evaluating cycle completion criteria based on test coverage, quality, cost, time, critical business objectives, etc.
   (A diagram would typically illustrate these phases in a circular or linear flow.)

3. Explain Boundary-Value Analysis in detail.

Boundary-Value Analysis (BVA) is a testing technique that focuses on the values at the boundaries of input ranges rather than the center. It is based on the premise that errors are more likely to occur at the edges of input ranges. For example, if a function accepts values from 1 to 100, test cases should include values like 0, 1, 100, and 101. This method helps identify edge cases that could lead to defects, ensuring robust testing of input validation.

4. Explain Acceptance Testing in detail.

Acceptance Testing is the final phase of testing conducted to determine whether the software meets the acceptance criteria and is ready for deployment. It is typically performed by end-users or clients. There are two main types:

1. **User Acceptance Testing (UAT)**: Conducted by end-users to validate the software against their requirements.
2. **Operational Acceptance Testing (OAT)**: Focuses on operational aspects like backup, recovery, and maintenance. Acceptance Testing ensures that the software is fit for use in the real world.

5. Explain Test Case Design along with an example.

Test Case Design involves creating a set of conditions or variables under which a tester will determine whether a system or software application is working correctly. A test case typically includes:

- **Test Case ID**: Unique identifier.
- **Test Description**: What is being tested.
- **Preconditions**: Conditions that must be met before executing the test.
- **Test Steps**: Detailed steps to execute the test.
- **Expected Result**: The expected outcome of the test.
  Example: For a login process, a test case might be:
- **Test Case ID**: TC001
- **Description**: Verify login with valid credentials.
- **Preconditions**: User must be registered.
- **Test Steps**: Enter username and password, click login.
- **Expected Result**: User is redirected to the dashboard.

6. Explain any four testing principles in detail.

1. **Testing Shows the Presence of Defects**: Testing can demonstrate that defects are present but cannot prove that there are no defects.
2. **Exhaustive Testing is Impossible**: It is impractical to test all possible inputs and scenarios; hence, risk-based testing is essential.
3. **Early Testing**: Testing should start as early as possible in the software development lifecycle to identify defects early and reduce costs.
4. **Defect Clustering**: A small number of modules usually contain most of the defects; focusing on these can yield better results in testing.

7. Explain white-box testing and its techniques.

White-box testing, also known as clear-box or glass-box testing, involves testing the internal structures or workings of an application. Testers have knowledge of the code and design. Techniques include:

1. **Statement Coverage**: Ensures every statement in the code is executed at least once.
2. **Branch Coverage**: Ensures every branch (true/false) in control structures is tested.
3. **Path Coverage**: Ensures all possible paths through the code are executed.
4. **Condition Coverage**: Tests all boolean expressions to ensure each condition is evaluated to both true and false.

8. Explain the Sandwich and Big-Bang approach of Integration Testing.

- **Sandwich Approach**: This is a hybrid approach that combines both Top-Down and Bottom-Up integration testing. In this method, the high-level modules are tested first, followed by the lower-level modules. It allows for early testing of critical functionalities while also ensuring that lower-level components are integrated and tested as they become available.
- **Big-Bang Approach**: In this approach, all components or modules are integrated simultaneously after all have been developed. Testing is then performed on the entire system as a whole. This method can lead to challenges in identifying defects since the integration happens all at once, making it difficult to trace issues back to specific modules.

9. Explain Load and Smoke Testing in detail.

- **Load Testing**: This type of testing evaluates the system's performance under expected load conditions. It helps identify how the system behaves under normal and peak load scenarios, ensuring it can handle the anticipated number of users and transactions without performance degradation. Load testing tools simulate multiple users to assess response times, throughput, and resource utilization.
- **Smoke Testing**: Also known as "sanity testing," smoke testing is a preliminary test to check the basic functionality of an application. It is performed after a new build to ensure that the critical features work as expected. If the smoke test fails, the build is rejected, and further testing is halted until the issues are resolved.

10. Write the difference between Static and Dynamic Testing.

- **Static Testing**: This involves reviewing the code, requirements, and design documents without executing the program. Techniques include code reviews, inspections, and static analysis. It helps identify defects early in the development process and is cost-effective.
- **Dynamic Testing**: This involves executing the code and validating the output against expected results. It includes various testing types such as unit, integration, system, and acceptance testing. Dynamic testing helps identify runtime errors and performance issues that static testing cannot uncover.

11. Explain test case design for the login process.

Test case design for the login process involves creating specific scenarios to validate the functionality. A sample test case might include:

- **Test Case ID**: TC_Login_01
- **Description**: Verify login with valid credentials.

- **Preconditions**: User account exists.
- **Test Steps**:
    1. Navigate to the login page.
    2. Enter valid username and password.
    3. Click the "Login" button.
- **Expected Result**: User is redirected to the dashboard.
  Additional test cases would cover invalid credentials, empty fields, and password recovery scenarios.

12. Explain the Stub and Driver concept in Unit Testing.

- **Stub**: A stub is a simulated component that mimics the behavior of a lower-level module that has not yet been developed. It provides predefined responses to calls made during testing, allowing the higher-level module to be tested in isolation.
- **Driver**: A driver is a temporary module that invokes a lower-level module for testing. It simulates the higher-level module's behavior, allowing the lower-level module to be tested independently. Both stubs and drivers facilitate unit testing by enabling isolated testing of components.

13. Explain GUI Testing in detail.

GUI Testing focuses on validating the graphical user interface of an application. It ensures that the interface meets design specifications and provides a user-friendly experience. Key aspects include:

- **Layout and Design**: Verifying that elements are correctly positioned and visually appealing.
- **Functionality**: Ensuring that buttons, links, and other interactive elements work as intended.

- **Usability**: Assessing the ease of use and intuitiveness of the interface.
- **Compatibility**: Testing the GUI across different browsers and devices to ensure consistent behavior. Automated tools can assist in GUI testing, but manual testing is also essential for user experience evaluation.

14. What is the difference between client/server testing and web-based testing?

- **Client/Server Testing**: This involves testing applications that operate in a client-server architecture, where the client requests services from the server. Testing focuses on network communication, data transfer, and server response times. It often requires testing the application in a controlled environment that mimics the client-server setup.
- **Web-Based Testing**: This pertains to testing applications that run on web browsers. It involves validating functionality, performance, and security across various browsers and devices. Web-based testing also includes testing for responsiveness, load handling, and compatibility with different operating systems.

15. How to calculate the cyclomatic complexity of a code? Explain with an example.

Cyclomatic complexity is a software metric used to measure the complexity of a program. It is calculated using the formula:
$M = E - N + 2P$
Where:

- **M** = Cyclomatic complexity
- **E** = Number of edges in the control flow graph
- **N** = Number of nodes in the control flow graph
- **P** = Number of connected components (usually 1 for a single program)

Example: Consider a simple function with the following control flow:

```
 if (condition1) {
   // Block A
} else {
   // Block B
}
if (condition2) {
  // Block C
}
```

In this example:

- **Nodes (N)**: 4 (entry point, Block A, Block B, Block C)
- **Edges (E):** 5 (entry to Block A, entry to Block B, Block A to Block C, Block B to Block C, and Block C to exit)
- **Connected components (P)**: 1

Using the formula:
$M = E - N + 2P = 5 - 4 + 2(1) = 3$

Thus, the cyclomatic complexity is 3, indicating the number of linearly independent paths through the code.

16. Explain the V-Model of testing in detail.

The V-Model, or Verification and Validation Model, is a software development process that emphasizes the relationship between development and testing phases. It is represented in a V shape, where the left side represents the development stages and the right side represents corresponding testing stages. Key components include:

1. **Requirements Analysis**: Define user requirements.
2. **System Design**: Create system architecture.
3. **Architecture Design**: Define system components.
4. **Module Design**: Design individual modules.
5. **Coding**: Implement the design in code.
6. **Unit Testing**: Validate individual modules against design specifications.
7. **Integration Testing**: Test interactions between integrated modules.
8. **System Testing**: Validate the complete system against requirements.
9. **Acceptance Testing**: Ensure the system meets user needs.
   The V-Model emphasizes early testing and validation, ensuring that each development phase has a corresponding testing phase.

17. Differentiate between alpha and beta testing.

- **Alpha Testing**: Conducted in-house by the development team before the software is released to external users. It focuses on identifying bugs and issues in a controlled environment. Alpha testing is typically performed in a lab setting and involves both white-box and black-box testing techniques.
- **Beta Testing**: Conducted by a limited number of external users in a real-world environment after alpha testing is complete. The goal is to gather feedback on the software's performance, usability, and any remaining issues. Beta testing helps identify defects that may not have been discovered during alpha testing and provides insights into user experience.

18. Explain the Capability Maturity Model (CMM) in detail.

The Capability Maturity Model (CMM) is a framework for assessing and improving software development processes. It consists of five maturity levels:

1. **Initial**: Processes are unpredictable and poorly controlled.
2. **Managed**: Basic project management processes are established, and projects are planned and executed according to policy.
3. **Defined**: Processes are documented, standardized, and integrated into a coherent process for the organization.
4. **Quantitatively Managed**: Processes are controlled using statistical and other quantitative techniques.
5. **Optimizing**: Focus on continuous process improvement through incremental and innovative technological improvements.
   CMM helps organizations improve their processes, reduce risks, and enhance product quality.

19. What is debugging? Explain with its phases.

Debugging is the process of identifying, isolating, and fixing defects or bugs in software. It involves several phases:

1. **Error Detection**: Identifying that a bug exists through testing or user reports.
2. **Error Localization**: Isolating the part of the code where the bug is likely located. This may involve reviewing logs, using debugging tools, or analyzing code flow.
3. **Error Correction**: Modifying the code to fix the identified bug. This may involve rewriting code, changing logic, or adjusting configurations.
4. **Verification**: Testing the modified code to ensure that the bug is fixed and that no new issues have been introduced.

5. **Documentation**: Recording the bug, its cause, and the solution for future reference and to improve the debugging process.

20. What is Black-Box Testing? Explain with its techniques.

Black-Box Testing is a testing technique that evaluates the functionality of an application without peering into its internal structures or workings. Testers focus on input and output rather than the code itself. Techniques include:

1. **Equivalence Partitioning**: Dividing input data into valid and invalid partitions to reduce the number of test cases.
2. **Boundary Value Analysis**: Testing at the boundaries of input ranges to identify edge cases.
3. **Decision Table Testing**: Using a table to represent combinations of inputs and their corresponding outputs, ensuring all scenarios are tested.
4. **State Transition Testing**: Testing the application's behavior based on state changes, ensuring it responds correctly to different inputs in various states.
   Black-Box Testing is effective for functional testing and user acceptance testing, as it focuses on user requirements and system behavior.

21. Explain Top-Down and Bottom-Up Integration Testing in detail.

- **Top-Down Integration Testing**: This approach starts testing from the top-level modules and progressively integrates and tests lower-level modules. Stubs are used to simulate the behavior of lower-level modules that are not yet integrated. This method allows for early testing of high-level functionalities and helps identify design flaws early in the development process.

- **Bottom-Up Integration Testing**: In this approach, testing begins with the lower-level modules, which are integrated and tested first. Drivers are used to simulate the behavior of higher-level modules. This method allows for thorough testing of individual components before integrating them into the larger system, ensuring that lower-level functionalities are robust.

22. Explain the term Unit Testing.

Unit Testing is a software testing technique where individual components or modules of a software application are tested in isolation. The primary goal is to validate that each unit of the software performs as expected. Unit tests are typically automated and written by developers during the coding phase. Key aspects include:

- **Isolation**: Each unit is tested independently to ensure that it functions correctly without dependencies on other units.
- **Automation**: Unit tests are often automated using testing frameworks, allowing for quick execution and frequent testing.
- **Early Detection**: By testing units early in the development process, defects can be identified and fixed before they propagate to higher levels of testing.

23. Explain the software testing process.

The software testing process involves a series of steps to ensure that software meets specified requirements and is free of defects. Key steps include:

1. **Test Planning**: Defining the scope, objectives, resources, and schedule for testing activities.
2. **Test Design**: Creating test cases and test scripts based on requirements and design specifications.

3. **Test Environment Setup**: Preparing the hardware and software environment where testing will be conducted.
4. **Test Execution**: Running the test cases and documenting the results, including any defects found.
5. **Defect Reporting**: Logging defects in a tracking system for resolution by the development team.
6. **Test Closure**: Evaluating the testing process, documenting lessons learned, and preparing test summary reports.

24. Write the difference between Quality Assurance (QA) and Quality Control (QC).

- **Quality Assurance (QA)**: QA is a proactive process that focuses on preventing defects by establishing and improving processes and standards. It involves activities such as process definition, training, and audits to ensure that the development process is followed correctly.
- **Quality Control (QC)**: QC is a reactive process that focuses on identifying defects in the final product. It involves activities such as testing, inspections, and reviews to ensure that the software meets quality standards and requirements. While QA aims to improve processes, QC aims to verify the quality of the product.

25. Explain JUnit in detail.

JUnit is a widely used testing framework for Java applications that supports the development of unit tests. Key features include:

- **Annotations**: JUnit uses annotations (e.g., @Test, @Before, @After) to define test methods and setup/teardown procedures.
- **Assertions**: It provides a set of assertion methods (e.g., assertEquals, assertTrue) to validate expected outcomes against actual results.

- **Test Suites**: JUnit allows grouping multiple test cases into test suites for organized execution.
- **Integration with Build Tools**: JUnit integrates seamlessly with build tools like Maven and Gradle, enabling automated testing as part of the build process.
  JUnit promotes test-driven development (TDD) by encouraging developers to write tests before implementing functionality.

26. Explain Performance Testing in detail.

Performance Testing is a type of testing that evaluates the speed, scalability, and stability of a software application under various conditions. Key objectives include:

- **Load Testing**: Assessing how the application performs under expected user loads to ensure it can handle the anticipated traffic.
- **Stress Testing**: Testing the application beyond its normal operational capacity to identify breaking points and ensure it can recover gracefully from failures.
- **Endurance Testing**: Evaluating the application's performance over an extended period to identify memory leaks and performance degradation.
- **Scalability Testing**: Determining how well the application can scale up or down in response to increased or decreased loads.
  Performance testing helps ensure that applications meet performance requirements and provide a satisfactory user experience.

Q3. Short Notes.

1. Software Review

A software review is a systematic examination of software artifacts, such as requirements, design documents, and code, to identify defects and improve quality. Reviews can be categorized into several types:

- **Peer Reviews**: Colleagues review each other's work to provide feedback and identify issues.
- **Formal Inspections**: A structured process involving a team that reviews documents and code against predefined criteria.
- **Walkthroughs**: A less formal process where the author presents the work to stakeholders for feedback.
  The primary goals of software reviews are to catch defects early, improve team communication, and enhance the overall quality of the software product.

2. Testing Documentation

Testing documentation refers to the various documents created during the software testing process to ensure thoroughness and traceability. Key types of testing documentation include:

- **Test Plan**: A document outlining the scope, approach, resources, and schedule for testing activities.
- **Test Cases**: Detailed descriptions of specific conditions under which a test will be conducted, including inputs, execution steps, and expected results.
- **Test Scripts**: Automated scripts that execute test cases in testing tools.
- **Test Summary Report**: A document summarizing the testing activities, results, and any defects found.
  Effective testing documentation helps ensure that testing is systematic, repeatable, and provides a clear record of testing activities.

3. Goals of Software Testing

The primary goals of software testing include:

1. **Defect Detection**: Identify and fix defects before the software is released to ensure quality.
2. **Validation**: Ensure that the software meets the specified requirements and fulfills user needs.
3. **Verification**: Confirm that the software is built according to design specifications and standards.
4. **Risk Mitigation**: Reduce the risk of software failures in production by identifying issues early.
5. **Quality Assurance**: Provide confidence to stakeholders that the software is reliable, efficient, and meets quality standards.

4. Testing for Real-Time Systems

Testing for real-time systems involves evaluating software that must respond to inputs within strict time constraints. Key considerations include:

- **Timing Constraints**: Ensuring that the system meets deadlines for processing inputs and producing outputs.
- **Concurrency**: Testing how the system handles multiple simultaneous operations.
- **Resource Management**: Evaluating how the system manages memory and processing resources under load.
- **Fault Tolerance**: Assessing the system's ability to continue functioning correctly in the presence of faults.
  Real-time testing often requires specialized tools and techniques to simulate real-world conditions and validate performance.

5. Stub and Driver Concept in Unit Testing

In unit testing, stubs and drivers are used to facilitate the testing of individual components in isolation:

- **Stub**: A stub is a simulated component that replaces a lower-level module that has not yet been developed. It provides predefined responses to calls made during testing, allowing the higher-level module to be tested without needing the complete system.
- **Driver**: A driver is a temporary module that invokes a lower-level module for testing. It simulates the higher-level module's behavior, allowing the lower-level module to be tested independently. Both stubs and drivers help isolate units for effective testing.

6. Load Runner

LoadRunner is a performance testing tool developed by Micro Focus (formerly by HP) that allows users to simulate virtual users and measure system performance under load. Key features include:

- **Load Generation**: Simulates thousands of users to test how the application performs under various load conditions.
- **Performance Monitoring**: Monitors system performance metrics such as response time, throughput, and resource utilization during tests.
- **Analysis and Reporting**: Provides detailed reports and graphs to analyze performance results and identify bottlenecks.
  LoadRunner is widely used for load testing, stress testing, and performance testing of applications across different environments.

7. Rational Robot

Rational Robot is an automated functional testing tool that was part of the IBM Rational suite. It allows testers to create, execute, and manage automated tests for applications. Key features include:

- **Record and Playback**: Enables users to record user interactions with the application and replay them for testing.

- **Scripting**: Supports scripting in various programming languages for more complex test scenarios.
- **Integration**: Integrates with other Rational tools for comprehensive test management and reporting.
  Rational Robot is used to improve testing efficiency and ensure consistent test execution.

8. System Testing

System Testing is a level of testing that validates the complete and integrated software system against specified requirements. It is performed after integration testing and focuses on the overall behavior of the application. Key aspects include:

- **End-to-End Testing**: Tests the complete application flow from start to finish to ensure all components work together as expected - **Functional Testing**: Verifies that the system functions according to the requirements, including all features and functionalities.
- **Non-Functional Testing**: Assesses aspects such as performance, usability, reliability, and security of the system.
- **Acceptance Testing**: Often involves end-users to validate that the system meets their needs and is ready for deployment.
  System testing is crucial for identifying defects and ensuring that the software is ready for production.

9. Statement Coverage Criteria of White-Box Testing

Statement coverage is a white-box testing metric that measures the percentage of executable statements in the code that have been executed during testing. The key points include:

- **Objective**: To ensure that every statement in the code is tested at least once, helping to identify untested paths.

- **Calculation**: It is calculated by dividing the number of executed statements by the total number of executable statements and multiplying by 100.
- **Limitations**: While it helps in identifying parts of the code that are not executed, it does not guarantee that all logical paths or conditions are tested.
  Statement coverage is often used in conjunction with other coverage criteria, such as branch coverage, to provide a more comprehensive assessment of test effectiveness.

10. Goal-Question-Metric (GQM) Model

The Goal-Question-Metric (GQM) model is a framework used for defining and interpreting software measurement. It consists of three components:

- **Goals**: High-level objectives that the organization wants to achieve, such as improving software quality or reducing defects.
- **Questions**: Specific questions that help to evaluate whether the goals are being met, guiding the measurement process.
- **Metrics**: Quantitative measures that provide data to answer the questions, allowing for objective assessment of progress towards the goals.
  The GQM model promotes a structured approach to measurement, ensuring that metrics are aligned with organizational objectives and providing valuable insights for decision-making.