**Explain Static Black Box Testing**

**Static Black Box Testing** is a method of testing that focuses on reviewing and analyzing software artifacts without executing the code. It evaluates the system from the user's perspective to ensure compliance with requirements and specifications.

**Characteristics of Static Black Box Testing:**

* Performed early in the development cycle.
* Does not require actual code execution.
* Focuses on finding issues in requirements, design documents, and test cases.

**Techniques Used:**

1. **Requirement Review**: Verifying that all requirements are clear, complete, and testable.
2. **Design Review**: Checking if the design adheres to specified requirements.
3. **Test Case Review**: Ensuring test cases cover all functional requirements.

**Benefits:**

* Identifies defects early, reducing costs.
* Improves the quality of requirements and designs.
* Enhances communication between stakeholders.

### **. What is Software Test Automation?**

**Software Test Automation** is the process of using automated tools or scripts to execute test cases, validate software functionality, and compare the actual results with expected outcomes without human intervention. It improves testing efficiency by eliminating repetitive manual testing tasks, enhancing test coverage, and accelerating the testing process.

**Key Benefits of Test Automation:**

1. **Efficiency**: Faster execution of repetitive test cases.
2. **Consistency**: Avoids human errors during test execution.
3. **Reusability**: Test scripts can be reused across multiple versions of the software.
4. **Cost-Effectiveness**: Reduces long-term costs by saving time on manual testing.
5. **Continuous Testing**: Enables integration into CI/CD pipelines for early defect detection.

### **Types of Test Automation Tools**

#### ****1. Functional Testing Tools****

* Validate that the application meets functional requirements.
* **Examples**: Selenium, UFT (Unified Functional Testing), TestComplete.

### **Explain Static Black Box Testing**

**Static Black Box Testing** is a testing technique that evaluates software artifacts such as requirements, design, and test cases without executing the code. It focuses on detecting defects in documents and specifications before the development or execution phase.

#### ****Characteristics of Static Black Box Testing****:

* It is performed early in the software development life cycle (SDLC).
* The system’s functionality is reviewed from an external perspective, without considering the internal code structure.
* Ensures that requirements and design are complete, correct, and testable.

#### ****Techniques Used in Static Black Box Testing****:

1. **Requirement Review**:
   * Ensures all requirements are well-defined and testable.
   * Identifies gaps or ambiguities in the requirements document.
2. **Design Review**:
   * Checks if the design aligns with specified requirements.
   * Validates the completeness and correctness of the system design.
3. **Test Case Review**:
   * Verifies that the test cases align with the functional requirements.
   * Ensures coverage of all scenarios, including edge and boundary cases.

#### ****Advantages of Static Black Box Testing****:

1. **Early Defect Detection**: Identifies defects in documentation, reducing downstream costs.
2. **Improved Quality**: Enhances the quality of requirements and design.
3. **Time Efficiency**: Saves time by addressing issues before development.
4. **Collaboration**: Facilitates discussions between stakeholders, leading to better understanding and refinement of requirements.

### **b. What is Unit Testing? What are its Requirements?**

#### ****Unit Testing****

Unit testing is the process of testing individual components or modules of a software application in isolation to ensure that they function as expected. It is typically performed by developers during the development phase.

**Key Objectives of Unit Testing:**

1. Validate the correctness of a single unit of code.
2. Detect and fix bugs early in the development process.
3. Ensure that each module behaves as intended in isolation.

#### ****Requirements for Unit Testing:****

1. **Test Data**: Inputs needed to validate the functionality of the unit.
2. **Test Cases**: Clear and concise test cases for each module or function.
3. **Test Environment**: A setup to execute unit tests (e.g., frameworks, compilers).
4. **Test Frameworks**: Tools like JUnit, NUnit, or PyTest to automate unit testing.
5. **Mock Objects**: Simulated objects to test units that interact with external dependencies (e.g., databases, APIs).

### **c. Different Fields in a Test Case**

A test case is a document that outlines the conditions and steps to validate whether a software feature functions as intended. It typically includes the following fields:

1. **Test Case ID**:
   * A unique identifier for the test case (e.g., TC001).
2. **Test Case Title**:
   * A brief description of the test case objective.
3. **Preconditions**:
   * Any conditions that must be met before executing the test (e.g., user logged in).
4. **Input Data**:
   * The data required for the test (e.g., username and password).
5. **Steps to Execute**:
   * Detailed, step-by-step instructions to perform the test.
6. **Expected Result**:
   * The anticipated outcome if the test case is successful.
7. **Actual Result**:
   * The actual result observed during execution.
8. **Pass/Fail Status**:
   * Indicates whether the test case passed or failed.
9. **Priority**:
   * The importance level of the test case (e.g., high, medium, low).
10. **Test Environment**:

* Details about the environment where the test was executed (e.g., OS, browser).

1. **Comments/Notes**:

* Additional remarks, including observations or issues encountered

#### ****Test Management****

Test management involves planning, controlling, and monitoring the testing process to ensure that the software meets quality standards. It encompasses organizing resources, tracking testing progress, and ensuring that all testing objectives are met.

**Key Activities in Test Management:**

1. **Test Execution Monitoring**: Track the progress of test case execution.
2. **Defect Management**: Log, prioritize, and track defects until resolution.
3. **Test Environment Setup**: Ensure proper configuration for testing activities.
4. **Reporting**: Generate reports on testing status, coverage, and metrics.
5. **Resource Management**: Allocate and monitor resources efficiently.

### **a. Test Planning and Test Management**

#### ****Test Planning****

Test planning is the process of defining the approach, scope, resources, and schedule of testing activities for a software project. It ensures that all necessary steps are outlined to achieve testing objectives and deliver quality software.

### **a. What do you mean by Static and Dynamic Testing?**

#### ****Static Testing****

Static testing is a type of software testing that evaluates code, requirements, or design without executing the program. It focuses on preventing defects early in the development life cycle.

**Key Characteristics:**

* Performed during the early stages of development.
* Involves reviewing documents like requirements, design, or code.
* Uses techniques like **reviews**, **walkthroughs**, and **inspections**.
* Helps detect errors like missing requirements, design flaws, or coding mistakes.

**Examples of Static Testing:**

* Code review.
* Requirement review.
* Test case review.

**Advantages:**

* Identifies defects early, reducing costs.
* Improves the quality of documents and designs.

#### ****Dynamic Testing****

Dynamic testing is a type of testing that involves executing the code or application to identify defects. It ensures that the software works as expected under different conditions.

**Key Characteristics:**

* Performed during or after implementation.
* Focuses on the system's functional and non-functional behavior.
* Includes various testing types like **unit testing**, **integration testing**, **system testing**, and **acceptance testing**.

**Examples of Dynamic Testing:**

* Running test cases for login functionality.
* Stress testing to evaluate performance under heavy loads.

**Advantages:**

* Validates the software's behavior in real-world scenarios.
* Ensures the application meets functional requirements.

### **c. Define Software and Software Testing**

#### ****Software****

Software is a collection of instructions, programs, or data that perform specific tasks on a computer system. It includes:

1. **System Software**: Manages hardware (e.g., operating systems).
2. **Application Software**: Performs user-specific tasks (e.g., word processors, web browsers).
3. **Middleware**: Facilitates communication between different software.

#### ****Characteristics of Software:****

* Intangible and adaptable.
* Needs continuous maintenance and upgrades.
* Can be customized for specific user needs.

#### ****Software Testing****

Software testing is the process of evaluating a software application to identify defects, ensure functionality, and verify that it meets specified requirements.

**Types of Testing:**

1. **Manual Testing**: Performed by testers without automation tools.
2. **Automated Testing**: Uses scripts or tools to execute tests.

**Objectives of Software Testing:**

* Identify and resolve defects.
* Ensure that the software meets user requirements.
* Improve software reliability and performance.

### **Goals of Test Planning**

Test planning involves outlining the approach, resources, schedule, and scope for testing activities. The goals of test planning are:

1. **Define Testing Objectives:**
   * Establish what the testing aims to achieve, such as defect detection or performance validation.
2. **Identify Scope and Exclusions:**
   * Determine the functionalities to be tested and specify areas outside the scope.
3. **Allocate Resources:**
   * Assign roles, responsibilities, tools, and infrastructure needed for testing.
4. **Develop Test Strategies:**
   * Define the testing approaches, methodologies, and techniques to be used.
5. **Schedule Testing Activities:**
   * Outline timelines and milestones for each phase of testing.
6. **Risk Identification and Mitigation:**
   * Identify potential risks (e.g., lack of resources, tight schedules) and plan measures to address them.
7. **Establish Metrics for Success:**
   * Define criteria for test completion, such as defect thresholds and coverage goals.

### **. Entry and Exit Criteria for Testing**

#### ****Entry Criteria****

Entry criteria are the conditions that must be met before starting a specific phase of testing. These ensure the readiness of the testing process and reduce the chances of inefficiencies.

**1. Availability of Test Environment**

* The testing environment must be set up and validated to ensure it mimics the production environment.
* Example: Servers, databases, and configurations are correctly set up for functional testing.

**2. Approval of Test Cases**

* All test cases must be written, reviewed, and approved by stakeholders before execution.
* Example: Test cases for a login module have been reviewed and signed off by the QA team lead.

#### ****Exit Criteria****

Exit criteria are the conditions that must be met to declare a testing phase complete. These ensure the quality and comprehensiveness of the testing process.

**1. Test Case Execution Completion**

* All planned test cases have been executed, with results documented.
* Example: Out of 500 test cases, 450 passed, and 50 were identified as known issues with documented workarounds.

**2. Defect Resolution and Closure**

* All critical and high-severity defects must be resolved and closed.
* Example: Bugs impacting login functionality or data security have been resolved and retested.

### **b. What Do You Mean by Test-to-Pass and Test-to-Fail?**

#### ****Test-to-Pass****

* Testing with the intention of ensuring the software works as expected under normal and valid conditions.
* Focuses on verifying that the application meets functional requirements and delivers desired outputs for valid inputs.
* Example: Testing a login system by entering valid credentials to confirm successful login.

**Advantages:**

* Verifies compliance with specifications.
* Ensures that core functionality works correctly.

**Limitations:**

* May miss potential defects because it focuses only on valid scenarios.

#### ****Test-to-Fail****

* Testing with the intention of identifying weaknesses or defects by providing invalid, incorrect, or unexpected inputs.
* Focuses on ensuring the application can handle negative scenarios gracefully without crashing or malfunctioning.
* Example: Testing a login system by entering invalid credentials, such as an empty username or incorrect password, to check error handling.

**Advantages:**

* Identifies robustness and error-handling issues.
* Ensures that the application behaves as expected in edge cases.

**Limitations:**

* Requires careful planning to cover all potential failure scenarios.

### **Advantages of Using Test Automation Tools**

Test automation tools offer several key benefits that enhance the efficiency and effectiveness of the software testing process. Below are four prominent advantages of using test automation tools:

### **1. Faster Execution and Repeatability**

**Advantage**: Automated tests can be executed significantly faster than manual tests. Once a test is written, it can be run repeatedly in a short time frame, making it possible to run large test suites in parallel and speed up the testing process.

**How it Helps**:

* Automated tests can run overnight or during off-hours, allowing the team to get quick feedback without waiting for manual testing.
* Re-running tests across different versions of the application is effortless and helps ensure that new code changes haven't introduced regressions.

**Example**:

* Running a suite of regression tests (e.g., 500 test cases) on different platforms (Windows, macOS, Linux) using automation tools like Selenium can take a few minutes instead of days, compared to manually testing each one.

### **2. Consistency and Accuracy**

**Advantage**: Automation tools execute tests in exactly the same way every time, eliminating the possibility of human error. This ensures consistency in test execution and results.

**How it Helps**:

* Automated tests will execute all steps in the same order and with the same parameters, ensuring that results are reliable and not influenced by tester fatigue or inconsistency.
* Accuracy is improved by eliminating human errors such as incorrect data entry, missed steps, or misunderstanding of the test requirements.

**Example**:

* In a manual test, a tester might forget to check a checkbox or skip a step. In an automated test, these steps will be executed precisely as designed, ensuring no part of the functionality is overlooked.

### **3. Cost-Effectiveness in the Long Run**

**Advantage**: While the initial investment in test automation tools and script development may be high, automated testing can save costs over the long term by reducing the time and effort needed for repetitive testing tasks.

**How it Helps**:

* Once the automated tests are created, they can be used for multiple releases without significant additional costs.
* Automated tests can run faster and more frequently, increasing the testing throughput and reducing the time-to-market for new features.
* Automated testing reduces the need for repetitive manual work, freeing testers to focus on more complex testing tasks.

**Example**:

* In continuous integration (CI) environments, automated tests run each time new code is pushed, allowing issues to be detected early, preventing costly fixes later in the development cycle.

### **4. Better Test Coverage**

**Advantage**: Test automation allows for running a broader range of test cases, including those that would be time-consuming or difficult to execute manually. This increases the overall test coverage and helps ensure that more aspects of the software are tested.

**How it Helps**:

* Automation tools can quickly execute test cases that involve large datasets, complex business logic, or multiple combinations of inputs, which would be hard or impractical to test manually.
* Automation enables tests to cover edge cases, boundary conditions, and performance tests that would be tedious to perform manually.

**Example**:

* Automated load testing with tools like JMeter can simulate thousands of concurrent users accessing a website, something that would be infeasible with manual testing. This ensures that the application performs well under various conditions.

### **Use of State Testing in Dynamic Testing**

**State Testing** is a type of **dynamic testing** that focuses on evaluating the behavior of a system in response to different states or transitions between states. It is used to verify that the software behaves as expected when it changes from one state to another. State testing is particularly useful in systems with complex state machines, where the output of the system depends on its current state and inputs.

### **Load Testing**

**Definition**:  
Load testing is a type of performance testing that is conducted to assess how a system behaves under a specific expected load, or the number of concurrent users or transactions. The goal of load testing is to determine if the system can handle the required volume of traffic or data without experiencing performance degradation.

**Purpose**:  
The main purpose of load testing is to identify the system's performance under normal and peak load conditions and ensure it meets the desired criteria for scalability and stability.

**Key Characteristics**:

* **Normal Load**: The expected number of concurrent users or transactions that the system is designed to handle under typical conditions.
* **Performance Monitoring**: Measures response time, throughput, and resource utilization (e.g., CPU, memory).
* **Scalability Testing**: Helps identify how the system scales as load increases.

**Example**:  
For an e-commerce website, load testing might involve simulating 1,000 users simultaneously browsing, adding items to the cart, and checking out to assess whether the website can handle the load without slowing down.

**Tools for Load Testing**:

* **Apache JMeter**
* **LoadRunner**
* **Gatling**
* **BlazeMeter**

### **Stress Testing**

**Definition**:  
Stress testing is a type of performance testing where the system is subjected to extreme loads, beyond its intended capacity, to see how it behaves under stressful conditions. The primary objective is to determine the system’s breaking point — the load at which it fails or experiences severe performance issues.

**Purpose**:  
Stress testing aims to identify the system’s robustness and its ability to recover from failure. It ensures that the application can handle unexpected spikes in traffic and continue to function, or fail gracefully if necessary.

**Key Characteristics**:

* **Beyond Normal Load**: Stress testing pushes the system to its limits or beyond, often simulating scenarios such as a sudden surge in traffic or a spike in transactions.
* **Failure Detection**: Helps detect how the system behaves when pushed past its maximum capacity, including where bottlenecks or failures occur.
* **Recovery Testing**: Evaluates how quickly and effectively the system recovers after stress is applied or after it crashes.

### **Driver and Stub in Software Testing**

**Drivers** and **Stubs** are used in **integration testing** to simulate the behavior of components or modules that are either not yet developed or are too complex to be integrated into the system during testing. They are tools used to isolate the components being tested, ensuring that tests can proceed without waiting for all parts of the system to be fully developed.

### **Driver**

**Definition**:  
A **driver** is a temporary component or program used in **top-down integration testing** to simulate the calling or controlling module of the software. It is used to call and test lower-level modules that are dependent on the higher-level ones. The driver provides the necessary input and controls the flow of execution of the module being tested.

**When is it used?**

* **In top-down integration testing**: When the main or controlling module (higher-level module) is not yet available or is incomplete, a driver is used to simulate it and pass control to the module under test.
* **To simulate missing or incomplete components**: The driver interacts with the module being tested, ensuring that lower-level modules can be tested without waiting for the complete system.

**Example**:  
In a banking application, if the **"account balance" module** (lower-level module) is being tested but depends on the **"user interface" module** (higher-level module), a driver would simulate the user interface, passing test inputs to the balance module.

### **Stub**

**Definition**:  
A **stub** is a temporary placeholder used in **bottom-up integration testing** to simulate lower-level modules or components that the module under test depends on. Stubs provide predefined responses or behavior, allowing the integration testing of higher-level modules without needing the actual lower-level modules to be developed.

**When is it used?**

* **In bottom-up integration testing**: When the lower-level modules are ready, but the higher-level modules that call these lower modules are not yet available, a stub is used to simulate the higher-level functionality.
* **To simulate complex or incomplete functionality**: The stub is used to simulate the behavior of the higher-level modules, returning appropriate responses to allow testing of lower-level modules.

### **Limitations of Manual Testing**

Manual testing is an essential part of software testing, especially for tasks like exploratory testing or usability testing. However, it has several limitations compared to automated testing. Below are **four key limitations of manual testing**:

### **1. Time-Consuming**

**Explanation**:  
Manual testing requires testers to execute test cases one by one, which can be time-consuming, especially when dealing with large applications with numerous features or test scenarios. Repetitive tasks such as regression testing or performance testing are particularly slow when done manually.

**Impact**:

* It increases the overall testing time, potentially delaying product releases.
* If the same tests need to be run repeatedly across different versions or builds, the time required can significantly increase.

**Example**:  
A web application that undergoes frequent updates (e.g., every few days) will require manual testing of all features with every release, leading to delays in the feedback cycle.

### **2. Human Error and Inconsistency**

**Explanation**:  
Manual testers are prone to human errors due to factors like fatigue, lack of attention, or misunderstanding of test cases. These errors can lead to missed defects or incorrect results, compromising the quality of testing.

**Impact**:

* Errors or inconsistencies in executing test cases can cause false positives or false negatives, meaning a bug might not be detected or a non-issue could be reported as a bug.
* Tester biases or incorrect assumptions may also affect the outcome.

**Example**:  
A tester might skip a step during a test due to fatigue or oversight, leading to an incomplete validation of a feature.

### **3. Limited Reusability**

**Explanation**:  
In manual testing, once a test case is executed, it can't be reused in the same form for future testing unless re-executed manually each time. This is inefficient when the application undergoes frequent changes, and repetitive testing is required (e.g., regression testing).

**Impact**:

* Manual tests are not as easily repeatable as automated tests.
* Creating and maintaining a large set of reusable test cases in manual testing is time-intensive and may require significant effort for each release cycle.

**Example**:  
A regression test suite for a large e-commerce site may need to be manually rerun with every new release to ensure existing features still function properly, which becomes impractical over time.

### **4. Lack of Scalability**

**Explanation**:  
Manual testing becomes less scalable as the complexity of the application grows. As the application adds more features or the number of users increases, the number of test cases also increases. Handling this manually requires a proportional increase in the number of testers or resources.

**Impact**:

* As the application size increases, manual testing efforts need to scale, which may not be feasible with limited resources.
* Testing a large system with many features manually can become overwhelming and may lead to missed or insufficient test coverage.

**Example**:  
In large-scale applications (e.g., an online banking platform), manually testing hundreds of features, security vulnerabilities, or edge cases may become too resource-intensive and difficult to manage.

### **Equivalence Partitioning in Dynamic Black Box Testing**

**Definition**:  
Equivalence Partitioning (EP) is a **black-box testing technique** used to divide the input domain of a program into classes or groups of data, called **equivalence classes**, that are treated the same way by the system. The idea is to test one value from each partition because all values in the same partition are expected to be processed in the same manner.

This technique helps to minimize the number of test cases while ensuring good coverage.

### **Key Features of Equivalence Partitioning**:

1. **Partitioning Input Data**:  
   Inputs are divided into valid and invalid equivalence classes. Each class represents a range or set of values that the program handles in a similar way.
2. **Reduced Test Cases**:  
   Instead of testing every possible input, one test case is chosen from each equivalence class, reducing redundancy.
3. **Dynamic Nature**:  
   It involves executing the program with test cases derived from the equivalence classes to validate system behavior.
4. **Error Detection**:  
   Helps in identifying defects related to input handling and processing.

### **Boundary Value Analysis (BVA) in Black Box Testing**

**Definition**:  
Boundary Value Analysis (BVA) is a black-box testing technique used to identify defects at the boundaries of input domains rather than focusing on the values in the middle of the input range. It is based on the principle that errors are more likely to occur at the boundaries of the input range than within the middle.

This technique complements **Equivalence Partitioning** by focusing on testing the edge cases of each equivalence class.

### **Key Features of BVA**:

1. **Boundary Focus**: Tests at the edges of input ranges (minimum, maximum, and just inside/outside boundaries).
2. **Dynamic Nature**: Involves execution of the application with boundary values as inputs.
3. **High Error Discovery**: Many defects are found when input values are at or near boundaries.

What is the job performed by software tester?

###### ChatGPT said:

ChatGPT

### **Job Responsibilities of a Software Tester**

A software tester plays a vital role in ensuring the quality and reliability of software applications. Their job involves planning, executing, and analyzing tests to identify and report bugs, ensure compliance with requirements, and improve the overall software performance and user experience. Below are the primary responsibilities of a software tester:

### **1. Understand Requirements and Specifications**

**Responsibilities**:

* Review and analyze software requirements, user stories, or technical documentation.
* Collaborate with business analysts, developers, and stakeholders to clarify unclear requirements.

**Purpose**:  
Ensure a clear understanding of the application functionality and objectives to create accurate test plans.

### **2. Develop Test Plans and Test Cases**

**Responsibilities**:

* Create detailed, comprehensive, and well-structured test plans and test cases.
* Define test strategies and scope (e.g., functional, regression, performance testing).

**Purpose**:  
Prepare a roadmap for effective testing and ensure test coverage for all application features.

### **3. Execute Tests**

**Responsibilities**:

* Perform manual and/or automated testing based on the test plan.
* Execute different types of tests such as functional, usability, compatibility, performance, and security testing.

**Purpose**:  
Validate the application’s functionality and performance under various scenarios.

### **4. Identify and Report Defects**

**Responsibilities**:

* Detect, document, and report bugs or issues in the application.
* Use defect tracking tools (e.g., JIRA, Bugzilla) to log and track issues.

**Purpose**:  
Ensure developers are informed about software issues to address them promptly.

### **5. Perform Regression Testing**

**Responsibilities**:

* Retest previously tested functionalities after code changes.
* Verify that the new code does not introduce new defects into the existing functionality.

**Purpose**:  
Ensure application stability after updates or enhancements.