1) what is testing in ur words?

Testing, in my own words, refers to the process of evaluating a system or a component to ensure that it behaves as expected and meets specified requirements. It involves executing the software with the intention of finding defects or verifying that it functions correctly according to its design and user requirements.

Testing can involve various activities such as:

Verification: Ensuring that the software meets its specified requirements.

Validation: Confirming that the software meets the needs of the end-users and stakeholders.

Error Detection: Identifying defects or discrepancies between expected and actual behavior.

Quality Assurance: Guaranteeing the quality of the software product through systematic testing processes.

Performance Evaluation: Assessing the speed, responsiveness, and scalability of the software under different conditions.

Overall, testing is crucial for ensuring the reliability, functionality, and usability of software products before they are deployed to users. It helps identify and rectify issues early in the development process, thereby reducing risks and enhancing the overall quality of the software.

2) what is manual testing?

Manual testing is a type of software testing where testers manually execute test cases without the use of any automation tools. In manual testing, human testers interact directly with the software application to evaluate its functionality, usability, performance, and other aspects.

Here are some key characteristics of manual testing:

Test Case Execution: Testers execute predefined test cases, step by step, to verify the behavior of the software under test. Test cases may cover various aspects such as functional requirements, user interface, compatibility, and error handling.

Exploratory Testing: Testers may also conduct exploratory testing, where they explore the software freely to discover defects, usability issues, or areas that were not covered by predefined test cases.

Ad Hoc Testing: Testers may perform ad hoc testing, which involves testing the software without following a predefined test plan. Ad hoc testing can be useful for uncovering unexpected issues or scenarios.

Usability Testing: Manual testing often includes usability testing, where testers evaluate the software's user interface, ease of use, and overall user experience.

Regression Testing: Testers may perform manual regression testing to ensure that new changes or updates to the software do not introduce new defects or regressions into existing functionality.

Manual testing offers several advantages, including:

Flexibility to adapt test cases based on real-time feedback and observations.

Ability to simulate real-user interactions and scenarios effectively.

Early detection of usability issues and user experience problems.

Cost-effectiveness for small-scale or short-term projects.

However, manual testing also has limitations, such as:

Time-consuming and labor-intensive, especially for repetitive or large-scale testing.

Prone to human error and subjectivity in test execution and evaluation.

Limited scalability, as it may not be feasible for testing complex or rapidly changing software systems.

Despite the rise of automated testing tools and practices, manual testing remains an integral part of the software testing process, especially for tasks that require human judgment, creativity, and adaptability.

3) what is test case?

A test case is a set of conditions or variables under which a tester determines whether a system or software application works correctly. It serves as a detailed description of the steps to be taken, along with the expected results, to verify that a particular feature or functionality of the software meets specified requirements.

4) what stlc?

STLC stands for Software Testing Life Cycle. It is a structured approach used by software development teams to plan, execute, and manage testing activities throughout the software development process. STLC consists of several phases, each with its specific objectives, tasks, and deliverables. These phases ensure that testing is conducted systematically and efficiently, leading to high-quality software products. Here are the typical phases of the Software Testing Life Cycle:

Requirement Analysis: In this phase, testers analyze the software requirements to understand what needs to be tested. They review the requirements documents, identify testable features, and create a test plan outlining the testing approach, scope, resources, and schedule.

Test Planning: Test planning involves creating a detailed test strategy and test plan based on the information gathered during requirement analysis. Testers define test objectives, test scenarios, test cases, test data, and test environments. They also allocate resources and establish timelines for testing activities.

Test Case Development: In this phase, testers develop test cases based on the test scenarios identified during test planning. Test cases describe the step-by-step procedures to be followed during testing, along with the expected results. Testers may also create test scripts for automated testing if applicable.

Test Environment Setup: Test environment setup involves configuring the necessary hardware, software, and network infrastructure for testing. Testers ensure that the test environment closely resembles the production environment to accurately simulate real-world conditions.

Test Execution: Test execution is the phase where testers execute the test cases developed earlier. They perform functional, regression, integration, performance, and other types of testing as per the test plan. Testers record test results, log defects, and communicate with developers to resolve issues.

Defect Tracking and Management: During test execution, testers identify defects or bugs in the software. They log these defects in a defect tracking system, providing detailed information such as the steps to reproduce the issue, severity, priority, and status. Testers collaborate with developers to fix the defects and verify the fixes.

Test Reporting: Test reporting involves generating test reports to communicate the testing progress, results, and metrics to stakeholders. Test reports provide insights into the quality of the software, identify risks, and help in decision-making regarding software release.

Test Closure: The final phase of STLC is test closure, where testers evaluate the completion criteria defined in the test plan to determine whether testing objectives have been met. They compile lessons learned, document best practices, and archive test artifacts for future reference.

5) what is requirement analysis?

Requirement Analysis: In this phase, testers analyze the software requirements to understand what needs to be tested. They review the requirements documents, identify testable features, and create a test plan outlining the testing approach, scope, resources, and schedule.

6) what is functional testing?

Functional testing is a type of software testing that focuses on verifying that the software functions correctly according to its specified requirements. The primary goal of functional testing is to ensure that the software performs its intended functions accurately and reliably, without any unexpected behavior or deviations from the expected functionality. Functional testing involves testing the various features, functionalities, and user interactions of the software application to validate its behavior against the functional requirements.

Key characteristics of functional testing include:

Testing Based on Requirements: Functional testing is driven by the functional requirements of the software. Test cases are designed to validate that the software performs the functions described in the requirements documents, user stories, or other specifications.

Black Box Testing Approach: Functional testing typically follows a black-box testing approach, where testers examine the software from an external perspective without knowledge of its internal implementation details. Testers focus on inputs, outputs, and user interactions to validate the software's behavior.

Test Scenarios and Test Cases: Test scenarios and test cases are developed based on functional requirements to cover various use cases and workflows of the software application. Testers execute these test cases to verify that the software functions as expected under different conditions.

Functional Coverage: Functional testing aims to achieve comprehensive coverage of the software's functional capabilities. Test cases are designed to cover all features, functionalities, and user interactions to ensure thorough testing.

Regression Testing: Functional testing often includes regression testing, where previously tested functionalities are retested to ensure that new changes or updates to the software do not introduce regressions or unintended side effects.

User Interface Testing: User interface (UI) testing is an integral part of functional testing, focusing on validating the appearance, layout, and usability of the software's graphical user interface (GUI). UI testing ensures that the software meets usability requirements and provides a satisfactory user experience.

Examples of functional testing techniques include:

Unit Testing: Testing individual units or components of the software in isolation.

Integration Testing: Testing the interactions and interfaces between integrated components or modules.

System Testing: Testing the entire system as a whole to verify end-to-end functionality.

Acceptance Testing: Testing the software from the perspective of end-users to ensure that it meets their acceptance criteria and business requirements.

Overall, functional testing plays a critical role in validating the correctness and reliability of software applications, helping to ensure that they meet user expectations and deliver value to stakeholders.

7) what is non functional testing?

Non-functional testing, also known as quality attribute testing or performance testing, focuses on evaluating the non-functional aspects of a software application such as its performance, reliability, usability, scalability, security, and compatibility. Unlike functional testing, which verifies specific functionalities of the software, non-functional testing assesses how well the software meets the quality attributes or characteristics that are critical for its overall performance and user experience.

Here are some key aspects of non-functional testing:

Performance Testing: Performance testing evaluates the responsiveness, speed, and scalability of the software under different conditions, such as varying loads or concurrent users. It includes subtypes such as load testing, stress testing, and endurance testing.

Reliability Testing: Reliability testing assesses the software's ability to perform consistently and reliably over time. It verifies whether the software can handle prolonged usage without failures or unexpected behavior.

Usability Testing: Usability testing focuses on the ease of use and user experience of the software. It evaluates factors such as navigation, intuitiveness, accessibility, and user satisfaction.

Scalability Testing: Scalability testing assesses how well the software can handle increases in workload, data volume, or user traffic while maintaining performance and responsiveness. It helps determine the software's ability to scale up or down as needed.

Security Testing: Security testing identifies vulnerabilities and weaknesses in the software's security mechanisms and ensures that sensitive data is protected against unauthorized access, breaches, or attacks.

Compatibility Testing: Compatibility testing verifies that the software functions correctly across different platforms, operating systems, devices, browsers, and network environments. It ensures that the software is compatible with a wide range of configurations and setups.

Interoperability Testing: Interoperability testing evaluates the software's ability to interact and integrate with other systems, applications, or components. It ensures seamless communication and data exchange between different systems.

Maintainability Testing: Maintainability testing assesses how easily the software can be maintained, modified, or updated over time. It evaluates factors such as code readability, modularity, and documentation.

Non-functional testing is essential for ensuring that software meets the overall quality standards and requirements beyond just its functional behavior. By conducting comprehensive non-functional testing, organizations can identify and address potential issues related to performance, reliability, usability, security, and other critical aspects, thereby enhancing the overall quality and user satisfaction with the software application.

8) write the test cases on (exm:- one login page is u need to login and u place the order

Test Case: Verify successful login with valid credentials

Preconditions: User navigates to the login page of the website

Test Steps:

Enter valid username and password

Click on the login button

Expected Result: User should be successfully logged in and directed to the homepage.

Test Case: Verify error message for login with invalid username

Preconditions: User navigates to the login page of the website

Test Steps:

Enter an invalid username

Enter a valid password

Click on the login button

Expected Result: User should see an error message indicating that the username is invalid.

Test Case: Verify error message for login with invalid password

Preconditions: User navigates to the login page of the website

Test Steps:

Enter a valid username

Enter an invalid password

Click on the login button

Expected Result: User should see an error message indicating that the password is invalid.

Test Case: Verify error message for login with both invalid username and password

Preconditions: User navigates to the login page of the website

Test Steps:

Enter an invalid username

Enter an invalid password

Click on the login button

Expected Result: User should see an error message indicating that both username and password are invalid.

Test Case: Verify successful placement of an order after logging in

Preconditions: User is logged in and on the homepage

Test Steps:

Navigate to the product page

Select a product and specify quantity

Click on the "Add to Cart" button

Go to the shopping cart

Click on the "Proceed to Checkout" button

Enter shipping details and payment information

Click on the "Place Order" button

Expected Result: User should receive a confirmation message indicating that the order has been successfully placed.

These are just a few examples of test cases for the given scenario. Depending on the specific requirements and functionalities of the login page and order placement process, additional test cases may be needed to ensure thorough testing coverage.

10) what is difference btw smoke and sanity testing

Smoke testing and sanity testing are both types of software testing performed early in the software development life cycle to quickly assess the stability and basic functionality of the application. However, they serve different purposes and focus on different aspects of testing:

Smoke Testing:

Purpose: Smoke testing, also known as build verification testing (BVT) or build acceptance testing, is performed to verify that the critical functionalities of the software are working correctly and to assess the readiness of the software for further testing.

Scope: Smoke testing covers broad areas of functionality, typically focusing on major features or functionalities that are essential for the basic operation of the software.

Depth: Smoke testing is shallow and does not involve in-depth testing of individual features or components. Its primary goal is to identify showstopper defects or critical issues that would prevent further testing.

Timing: Smoke testing is performed on each new build or release of the software, usually immediately after it is deployed or integrated, to ensure that the build is stable and suitable for further testing.

Execution: Smoke tests are often automated to allow for quick execution and verification of essential functionalities. They may include a predefined set of test cases covering high-level scenarios.

Sanity Testing:

Purpose: Sanity testing, also known as sanity checking or health check testing, is performed to quickly verify that specific areas or functionalities of the software have been fixed or improved after changes or modifications.

Scope: Sanity testing focuses on specific areas or functionalities of the software that have undergone recent changes, enhancements, or bug fixes. It aims to ensure that these changes have not introduced new defects or regressions.

Depth: While smoke testing is shallow and verifies basic functionality, sanity testing can be more in-depth and may involve targeted testing of specific features or components affected by recent changes.

Timing: Sanity testing is typically performed after a specific set of changes or modifications has been made to the software, such as bug fixes, enhancements, or updates. It is used to validate that the changes have been successfully implemented without negatively impacting other areas of the software.

Execution: Sanity tests may be manual or automated, depending on the complexity of the changes and the availability of testing resources. They are often executed with a focus on the areas or functionalities affected by the recent changes.

In summary, while both smoke testing and sanity testing are performed early in the testing process to assess the stability and functionality of the software, smoke testing focuses on overall build stability and critical functionalities, while sanity testing focuses on specific areas or functionalities affected by recent changes or modifications.

11) what is UI testing ?

UI testing, also known as user interface testing or GUI testing, is a type of software testing that focuses on verifying the graphical user interface (GUI) of a software application. The primary goal of UI testing is to ensure that the user interface functions correctly, looks visually appealing, and provides a positive user experience.

Here are some key aspects of UI testing:

Functional Testing: UI testing verifies that the user interface elements such as buttons, menus, forms, links, and controls behave as expected and perform their intended functions. It involves testing user interactions and workflows to ensure that users can navigate through the application and perform tasks effectively.

Layout and Design: UI testing evaluates the layout, design, and visual elements of the user interface to ensure consistency, alignment, and readability. It verifies that the graphical elements are positioned correctly, styled appropriately, and adhere to design guidelines or branding requirements.

Navigation and Usability: UI testing assesses the ease of navigation and usability of the user interface. It checks whether users can intuitively navigate through different screens, access features easily, and understand the functionality of various elements without confusion.

Compatibility and Responsiveness: UI testing verifies that the user interface displays correctly and functions properly across different devices, screen sizes, resolutions, and web browsers. It ensures that the application is responsive and adapts to various viewing environments without layout distortions or functional issues.

Error Handling and Validation: UI testing validates error messages, notifications, and validation checks displayed to users in response to invalid inputs or erroneous actions. It ensures that error messages are clear, informative, and guide users to take appropriate corrective actions.

Localization and Internationalization: UI testing checks the localization and internationalization aspects of the user interface, ensuring support for multiple languages, cultural conventions, and regional preferences. It verifies that text translations, date formats, currency symbols, and other locale-specific elements are displayed correctly.

Accessibility: UI testing evaluates the accessibility of the user interface to ensure that it is usable by individuals with disabilities. It verifies compliance with accessibility standards such as Web Content Accessibility Guidelines (WCAG) and checks for features such as keyboard navigation, screen reader compatibility, and alternative text for images.

UI testing can be performed manually by testers interacting with the application's graphical interface or automated using UI testing tools that simulate user interactions and validate UI elements programmatically. Both manual and automated UI testing are essential for ensuring the quality, usability, and effectiveness of the software application's user interface.

10) what is android testing on phone?

Android testing on a phone refers to the process of testing Android applications directly on physical Android devices, such as smartphones or tablets, to ensure that they function correctly and provide a satisfactory user experience on real-world devices.

Here are some key aspects of Android testing on a phone:

Device Compatibility Testing: Android applications need to be tested on a variety of Android devices with different screen sizes, resolutions, hardware configurations, and Android versions to ensure compatibility and consistency across devices. Testing on real devices allows developers to identify device-specific issues and ensure that the app works well on popular devices used by the target audience.

User Interface Testing: Testing the user interface (UI) and user experience (UX) of Android applications on real devices helps verify that the app's layout, design, navigation, and interactions function as intended. It allows testers to assess the app's responsiveness, touch gestures, animations, and visual elements in a real-world context.

Performance Testing: Performance testing on Android devices involves evaluating the app's speed, responsiveness, memory usage, CPU utilization, battery consumption, and network performance under various conditions. Testing on physical devices enables testers to measure real-world performance metrics and identify performance bottlenecks or issues that may affect the app's performance on different devices.

Integration Testing: Integration testing on Android devices involves testing how the app interacts with device features, sensors, peripherals, and external services such as GPS, camera, accelerometer, gyroscope, microphone, and network connectivity. Testing on real devices allows developers to validate integration points and ensure seamless communication between the app and device hardware or external services.

User Acceptance Testing (UAT): User acceptance testing on Android devices involves validating the app's functionality, usability, and overall user experience with real users or stakeholders. Testing on physical devices allows testers to gather feedback from users in real-world usage scenarios and identify usability issues, bugs, or areas for improvement.

Security Testing: Security testing on Android devices involves assessing the app's security features, data protection mechanisms, encryption, authentication, authorization, and compliance with Android security best practices. Testing on real devices helps identify security vulnerabilities or weaknesses that may pose risks to user data or device security.

Localization and Internationalization Testing: Localization and internationalization testing on Android devices involves testing the app's support for different languages, regions, cultures, and locale-specific features. Testing on physical devices allows testers to verify that text translations, date formats, currency symbols, and other localized content are displayed correctly and meet the needs of international users.

Android testing on a phone can be performed manually by testers interacting with the app on physical devices or automated using Android testing frameworks and tools such as Espresso, UI Automator, Appium, and Firebase Test Lab. Both manual and automated testing approaches are essential for ensuring the quality, reliability, and performance of Android applications across a diverse range of devices and usage scenarios.