**Submitted by: kunal joshi**

Module–2(Manual Testing)

1. What is Exploratory Testing?

Exploratory testing is an approach to software testing that emphasizes simultaneous learning, test design, and test execution. Unlike traditional scripted testing, where test cases are predefined, exploratory testing involves testers actively exploring the application or system under test, learning its behavior, and designing and executing tests on the fly.

1. What is traceability matrix?

A traceability matrix, also known as a requirements traceability matrix (RTM), is a document used in software development and testing to establish and track the relationship between different project artifacts. It provides a way to ensure that all requirements, design elements, test cases, and other project deliverables are aligned and properly linked together.

1. What is Boundary value testing?

Boundary value testing is a software testing technique that focuses on testing the boundaries or extreme values of input and output ranges. It aims to identify defects or errors that are likely to occur at the edges or boundaries of valid input or output ranges.

The principle behind boundary value testing is based on the observation that defects often tend to cluster around the boundaries of input and output domains. By testing the values at the boundaries, it is more likely to uncover errors related to boundary conditions, off-by-one errors, and other boundary-related issues that might not be evident through normal testing.

1. What is Equivalence partitioning testing?

Equivalence partitioning testing is a software testing technique that involves dividing the input data of a system into groups or partitions, where each partition is expected to exhibit similar behavior. The purpose of this technique is to reduce redundant test cases while ensuring adequate coverage of the system's functionality.

The fundamental principle behind equivalence partitioning is that if a particular test case uncovers a defect in one partition, it is likely to affect other values within the same partition as well. Conversely, if a test case passes within a partition, it is expected to pass for any other value within that partition.

1. What is Integration testing?

Integration testing is a software testing technique that focuses on verifying the correct interaction and communication between different components or modules of a system when they are integrated together. The purpose of integration testing is to detect defects or issues that may arise due to the integration of individual components, ensuring that they work harmoniously as a cohesive unit.

1. What determines the level of risk?

Probability: The probability or likelihood of a specific event or risk occurring is a significant determinant of risk level. It assesses the chances of an undesirable event happening, such as the likelihood of a software defect occurring, a security breach happening, or a project milestone being missed.

Impact: The impact or consequence of a risk event occurring is another crucial factor in determining risk level. It evaluates the severity of the potential outcome and its potential effects on various aspects, such as financial loss, reputation damage, safety concerns, legal implications, or project failure.

Exposure: The level of exposure or vulnerability to a risk is important in assessing its potential impact. Factors such as the criticality of a system or process, the sensitivity of data, the complexity of the project, or the reliance on external dependencies can contribute to the level of exposure to risks.

Mitigation measures: The effectiveness of mitigation measures or controls in place to manage or reduce risks also influences the overall risk level. Robust risk management practices, proactive monitoring, security measures, redundancy, and contingency plans can help lower the level of risk by reducing the likelihood or impact of potential adverse events.

Context and industry standards: The context in which the risk is assessed and the industry standards or regulations applicable to the specific domain can also impact risk levels. Different industries and contexts may have different tolerance levels for risks, and compliance with industry-specific standards and regulations can help determine risk acceptability.

Risk appetite: The risk appetite or tolerance level of individuals, organizations, or stakeholders involved also plays a role in determining risk levels. Some entities may be more risk-averse and cautious, while others may be more willing to take on higher levels of risk to achieve certain objectives.

1. What is Alpha testing?

Alpha testing is a type of software testing that is performed by the developers or a select group of users in a controlled environment. It is conducted towards the end of the software development life cycle (SDLC) and prior to the release of the software to a wider audience.

1. What is beta testing?

Beta testing is a type of software testing that involves releasing a software product to a select group of external users or customers before its official launch. It is conducted in a real-world, non-laboratory environment and aims to gather user feedback, identify bugs or issues, and assess the software's performance and usability.

1. What is component testing?

Component testing, also known as module testing, is a level of software testing that focuses on testing individual components or modules of a system in isolation. It is performed after unit testing (testing individual functions or units of code) and before integration testing (testing the interaction between components).

1. What is functional system testing?

Functional system testing is a type of software testing that verifies the functional requirements and behavior of a complete software system or application. It involves testing the system as a whole, focusing on its intended functionality and how well it performs its intended tasks or operations.

1. What is Non-Functional Testing?

on-functional testing refers to the testing of software applications or systems against non-functional requirements. Unlike functional testing, which focuses on the specific functionalities of a system, non-functional testing aims to evaluate the characteristics or qualities of the system that are not directly related to its functionality.

1. What is GUI Testing?

GUI (Graphical User Interface) testing is a type of software testing that focuses on verifying the functionality, usability, and visual aspects of a software application's graphical user interface. The primary goal of GUI testing is to ensure that the application's interface functions correctly, is intuitive for users, and adheres to the specified design standards.

1. What is Adhoc testing?

Ad hoc testing is an informal and unplanned software testing technique where testers perform testing activities without any predefined test cases or test scripts. In ad hoc testing, testers explore the application or system in an unstructured manner to identify defects, issues, or unexpected behaviors.

1. What is load testing?

Load testing is a type of performance testing that assesses the behavior and performance of a software application or system under specific expected or peak workloads. The purpose of load testing is to evaluate the system's ability to handle the anticipated user load and to identify performance bottlenecks or limitations.

1. What is stress Testing?

Stress testing is a type of software testing that evaluates the behavior and stability of a system or application under extreme or unfavorable conditions. The purpose of stress testing is to determine the system's robustness, reliability, and performance under excessive loads, resource limitations, or other stressful scenarios.

1. What is white box testing and list the types of white box testing?

White box testing is a software testing technique that focuses on examining the internal structure, design, and implementation of a software system. It is also known as clear box testing, glass box testing, or structural testing. In white box testing, the tester has access to the internal workings of the system and has knowledge of its internal code, architecture, and design.

Unit Testing: This type of testing involves testing individual components or units of code in isolation to ensure their functionality is correct. It is typically done by developers using frameworks like JUnit or NUnit.

Integration Testing: Integration testing focuses on testing the interaction between different modules or components of a system to ensure that they work together correctly. It verifies the interfaces, data flow, and communication between various parts of the system.

Statement Coverage Testing: Statement coverage testing aims to ensure that each statement in the code is executed at least once during testing. The goal is to achieve 100% coverage of all executable statements.

Branch Coverage Testing: Branch coverage testing aims to test all possible branches (decision points) in the code. It ensures that both true and false outcomes of conditional statements are tested.

Path Coverage Testing: Path coverage testing aims to test all possible paths through the code. It ensures that every possible sequence of statements and branches is executed.

Condition Coverage Testing: Condition coverage testing ensures that each Boolean condition in the code is tested with both true and false values. It focuses on evaluating the outcomes of individual conditions.

Loop Coverage Testing: Loop coverage testing focuses on testing the loops in the code. It aims to test the loop boundaries, including the minimum number of iterations, maximum number of iterations, and handling of empty loops.

Function Coverage Testing: Function coverage testing aims to test all the functions or methods defined in the code. It ensures that each function is called and tested.

Mutation Testing: Mutation testing involves modifying certain parts of the code to create mutants, which are slightly altered versions of the original code. The objective is to check if the test cases can detect and identify these mutated versions, thereby ensuring the effectiveness of the tests.

1. What is black box testing? What are the different black box testing techniques?

Black box testing is a software testing approach that focuses on examining the functionality of a software system without having knowledge of its internal implementation. Testers treat the system as a "black box" and only interact with the inputs and observe the outputs, without knowing how the system processes the inputs to produce the outputs. The main objective of black box testing is to validate the software against the specified requirements and ensure that it behaves as expected from the end-user's perspective.

Functional Testing: Functional testing involves testing the functional requirements of the software system. Testers design test cases to verify that the system performs the intended functions correctly, according to the specified requirements.

Boundary Value Analysis: Boundary value analysis focuses on testing the behavior of the software system at the boundaries of input ranges. Testers select test cases that include both minimum and maximum valid inputs, as well as inputs just beyond these boundaries. This technique helps identify issues related to boundary conditions and off-by-one errors.

Equivalence Partitioning: Equivalence partitioning is a technique where testers divide the input data into different partitions or groups based on the system's behavior. Test cases are designed to cover each partition, aiming to test representative values from each group. This technique helps reduce the number of test cases while ensuring good coverage.

Decision Table Testing: Decision table testing is a technique used for testing systems with complex business rules or logic. Testers create a decision table that maps different combinations of inputs to the expected outputs based on the system's rules. This technique ensures that all possible combinations of inputs and corresponding outcomes are tested.

State Transition Testing: State transition testing focuses on testing systems with states or modes that can change based on specific conditions or events. Testers design test cases to exercise different state transitions and validate the system's behavior at each state. This technique helps uncover defects related to state changes, transitions, and state-dependent functionality.

Error Guessing: Error guessing is an informal technique where testers use their experience, intuition, and domain knowledge to identify potential error-prone areas of the software system. Test cases are designed based on these assumptions to target specific areas where defects are likely to occur.

Exploratory Testing: Exploratory testing is a dynamic and ad hoc testing approach where testers explore the software system with minimal predefined test cases. Testers actively learn, experiment, and adapt their testing as they navigate through the system. It is useful for discovering defects, usability issues, and understanding the system's behavior.

Regression Testing: Regression testing is performed to ensure that recent changes or fixes in the software system have not introduced new defects or caused unintended side effects. Test cases from previous test cycles are executed to validate the system's stability and compatibility.

1. Mention what are the categories of defects?

Functional Defects: Functional defects refer to issues or bugs that affect the expected functionality of the software. These defects can result in incorrect outputs, missing features, or deviations from the specified requirements.

Performance Defects: Performance defects are defects that impact the performance characteristics of the software system. They can include issues related to slow response times, high resource utilization, memory leaks, or bottlenecks that affect the system's efficiency.

Usability Defects: Usability defects are defects that affect the user experience of the software system. They can include issues related to poor user interface design, confusing navigation, unclear instructions, or difficulties in using the software effectively.

Compatibility Defects: Compatibility defects are defects that arise when the software system fails to function properly in different environments or configurations. These defects can include issues related to interoperability with specific hardware, operating systems, browsers, or other software components.

Security Defects: Security defects are defects that expose vulnerabilities or weaknesses in the software system, making it prone to unauthorized access, data breaches, or malicious attacks. These defects can include issues such as inadequate data encryption, improper access controls, or injection vulnerabilities.

Documentation Defects: Documentation defects refer to errors, omissions, or inaccuracies in the software system's documentation. These defects can include missing or outdated information, unclear instructions, or inconsistencies between the documentation and the actual software behavior.

Compatibility Defects: Compatibility defects occur when the software system fails to work correctly with other related systems or components, such as third-party libraries, databases, APIs, or network protocols. These defects can lead to integration failures, data corruption, or incorrect behavior.

Configuration Defects: Configuration defects are defects that occur when the software system is not properly configured or customized to meet the specific requirements of the environment or user. These defects can include issues related to incorrect settings, missing dependencies, or improper parameterization.

Data Defects: Data defects are defects that involve problems with data handling, storage, or processing within the software system. These defects can include issues such as data corruption, data loss, data inconsistency, or incorrect data transformations.

Localization and Internationalization Defects: Localization defects occur when the software system fails to adapt or translate properly to different languages, cultures, or regions. Internationalization defects occur when the software system is not designed to support different languages or regional requirements from the outset.

1. Mention what big bang testing is?

Big Bang testing is a software testing approach where all the individual components or modules of a software system are integrated together, and the system as a whole is tested. In this approach, testing is performed without any specific order or sequence of integration.

In Big Bang testing, the focus is on testing the interactions and functionality of the integrated system rather than testing individual components in isolation. It is often used when the development of individual components is completed, and there is a need to quickly assess the overall behavior of the system.

1. What is the purpose of exit criteria?

Decision Making: Exit criteria serve as a basis for making informed decisions about whether to continue or halt testing activities. They provide stakeholders, such as project managers, test managers, and development teams, with clear guidelines on when testing can be considered complete and the software can progress to the next phase.

Quality Assessment: Exit criteria help assess the quality of the software or the test phase. By defining specific quality measures or benchmarks, exit criteria enable stakeholders to evaluate whether the software or the test activities have met the desired quality standards. This assessment is crucial for ensuring that the software is sufficiently stable, reliable, and fit for the intended purpose.

Risk Management: Exit criteria contribute to effective risk management by establishing thresholds for critical quality attributes or areas that need to be addressed before proceeding further. By setting criteria related to high-priority defects, performance issues, or specific functional requirements, exit criteria help identify and mitigate potential risks that could impact the software's stability or user experience.

Resource Allocation: Exit criteria aid in resource management by providing guidance on when to reallocate testing resources to other project activities. Once the exit criteria are met, testing teams can focus on activities such as documentation, analysis of test results, test closure, or preparing for the next phase, optimizing resource utilization and project timelines.

Test Completion: Exit criteria define the completion of a specific test phase, such as system testing, acceptance testing, or regression testing. They ensure that the intended scope of testing is covered, required deliverables are produced, and objectives of the test phase are achieved. This helps maintain a structured and systematic approach to testing.

Communication and Transparency: Exit criteria promote effective communication among stakeholders by providing clear guidelines and expectations for test completion. They enable transparent and open discussions about the readiness of the software or the test phase, facilitating collaboration and shared understanding among project members.

1. When should "Regression Testing" be performed?

Regression testing should be performed after making changes to a software system or application. It is typically conducted to ensure that the existing functionalities of the system have not been affected by the recent changes or updates. Regression testing is especially important when:

New features or enhancements are added to the software.

Bug fixes or patches are applied.

Configuration changes are made to the system.

Integration of different software components or systems.

Third-party libraries or modules are updated.

Performance optimizations or code refactoring is performed.

1. What is 7 key principles? Explain in detail?

.1 Testing Shows the Presence of Defects

As stated in this testing principle, “Testing talks about the presence of defects and doesn’t talk about the absence of defects”. In software testing, we look for bugs to be fixed before we deploy systems to live environments – this gives us confidence that our systems will work correctly when goes live to users. Despite this, the testing process does not guarantee that the software is 100% error-free. It is true that testing greatly reduces the number of defects buried in software, however discovering and repairing these problems does not guarantee a bug-free product or system.

2 Exhaustive Testing is Impossible

Exhaustive testing usually tests and verifies all functionality of a software application while using both valid and invalid inputs and pre-conditions. No matter how hard you try, testing EVERYTHING is pretty much impossible. The inputs and outputs alone have an infinite number of combinations, so it is 100% not possible to test an application from every angle.

3 Early Testing

In software development, early testing means incorporating testing as early as possible in the development process. It plays a critical role in the software development lifecycle (SDLC). For instance, testing the requirements before coding begins.

4. Defect Clustering

In software testing, defect clustering refers to a small module or feature that has the most bugs or operation issues. This is because defects are not evenly distributed within a system but are clustered. It could be due to multiple factors, such as the modules might be complicated or the coding related to such modules might be complex. In software testing, defect clustering refers to a small module or feature that has the most bugs or operation issues. This is because defects are not evenly distributed within a system but are clustered. It could be due to multiple factors, such as the modules might be complicated or the coding related to such modules might be complex.

5. Pesticide Paradox

In software testing, the Pesticide Paradox generally refers to the practice of repeating the exact same test cases over and over again. As time passes, these test cases will cease to find new bugs. Developers will create tests which are passing so they can forget about negative or edge cases. This is based on the theory that when you repeatedly spray the same pesticide on crops in order to eradicate insects, the insects eventually develop an immunity, making the pesticide ineffective.

6 Testing is Context-Dependent

Each type of software system is tested differently. According to this principle, testing depends on the context of the software developed, and this is entirely true. The reality is that every application has its own unique set of requirements, so we can’t put testing in a box. Of course, every application goes through a defined testing process, however, the testing approach may vary based on the application type.

7. Absence of Error – Fallacy

The software which we built not only must be 99% bug-free software but also it must fulfill the business, as well as user requirements otherwise it will become unusable software. Even bug-free software may still be unusable if incorrect requirements are incorporated into the software, or if the software fails to meet the business needs.

1. Difference between QA v/s QC v/s Tester

**Quality assurance** is “A system for evaluating performance, as in the delivery of services or the quality of products provided to consumers, customers, or patients.

Quality Assurance(QA) is a standard process set up by an organization to meet quality standards. This process helps to avoid mistakes and defects, delivering products and services to customers.

QA is the foundation pillar for quality management as it focuses on the integrity of the product. It also provides confidence to stakeholders that desired and mentioned requirements are fulfilled.

**Quality Control** is “setting standards and testing to ensure something, like a product or service, is done correctly.”

Quality control aims to check whether the prescribed model was followed. It can be achieved by performing audits and determining whether the team followed the defined model to attain quality.

1. Difference between Smoke and Sanity?

**Smoke testing**

Smoke Testing is carried out post software build in the early stages of SDLC (software development life cycle) to reveal failures, if any, in the pre-released version of a software. The testing ensures that all core functionalities of the program are working smoothly and cohesively. A similar test is performed on hardware devices to ensure they don’t release smoke when induced with a power supply. Thus, the test gets its name ‘smoke test’. It is a subset of acceptance testing and is normally used in tester acceptance testing, system testing, and integration testing.

**Senity testing**

Smoke and Sanity Testing are both rapid in checking core functionalities of the code besides checking eligibility for further tests. However, while smoke testing ensures that acute functionalities of a program are working fine, sanity testing checks that the proposed functionality works as expected.

Most of the time, we get confused between smoke testing and sanity testing. To clear such confusion, let’s explore the differences between the two testing methods in detail.

1. Difference between verification and Validation

Verification

As mentioned, verification is the process of determining if the software in question is designed and developed according to specified requirements. Specifications act as inputs for the software development process. The code for any software application is written based on the specifications document.

Validation

Validation is often conducted after the completion of the entire software development process. It checks if the client gets the product they are expecting. Validation focuses only on the output; it does not concern itself about the internal processes and technical intricacies of the development process.

1. Explain types of Performance testing.

Load Testing: Load testing involves subjecting the system to realistic workloads and determining its behavior and performance under normal and peak load conditions. It assesses the application's ability to handle the expected number of users, transactions, or data volumes. The objective is to identify performance issues such as response time degradation, throughput limitations, and resource utilization problems.

Stress Testing: Stress testing involves testing the system beyond its normal operational capacity to determine how it handles excessive loads or unfavorable conditions. It aims to identify the breaking point or the system's ability to recover from failure. Stress testing often involves pushing the system to its limits by increasing the load, simulating extreme user activity, or reducing available resources.

Soak Testing: Soak testing, also known as endurance testing, involves subjecting the system to a sustained load for an extended period to assess its stability and performance over time. It helps identify issues such as memory leaks, resource depletion, or performance degradation that may occur after prolonged use. Soak testing aims to ensure the application can handle sustained activity without any adverse effects.

Spike Testing: Spike testing involves rapidly and significantly increasing the number of users or load on the system to evaluate its response and performance during sudden traffic spikes. It helps assess how the application scales and handles sudden increases in user activity. The objective is to determine if the system can handle sudden bursts in traffic without significant degradation in performance.

Scalability Testing: Scalability testing focuses on evaluating the application's ability to handle increased workloads by adding resources, such as hardware, servers, or network capacity. It aims to determine the application's performance as the workload is increased and ensures that it can scale effectively to meet growing demands.

Volume Testing: Volume testing involves testing the application's performance when subjected to a large volume of data. It aims to assess how the system handles and performs with a significant amount of data, such as large databases, files, or concurrent user sessions. The objective is to identify any performance issues related to data handling, processing, or storage.

Stability Testing: Stability testing focuses on assessing the application's performance and behavior over an extended period under normal conditions. It aims to identify any gradual performance degradation, memory leaks, or other issues that may arise after prolonged use.

1. What is Error, Defect, Bug and failure?

Error: An error, also known as a mistake or fault, refers to a human action that produces an incorrect or unintended result. It is a human action that deviates from the expected behavior or specification. Errors are introduced during the software development process and can include mistakes in design, coding, or documentation.

Defect: A defect, also referred to as a fault or flaw, is an imperfection or flaw in the software or its components that can cause the software to behave incorrectly, produce incorrect results, or fail to perform its intended function. Defects are introduced during the software development process and can occur due to errors in design, coding, or other implementation aspects.

Bug: The term "bug" is often used interchangeably with the term "defect." It refers to an error or flaw in the software that causes it to behave in an unintended or erroneous manner. The term "bug" originated from the early days of computing when actual insects were known to cause malfunctions in mechanical systems. Today, it is a commonly used term to describe any software defect.

Failure: A failure occurs when the software or system deviates from its expected behavior and does not perform its intended function. It is the manifestation of a defect or bug during the execution of the software. Failures can lead to incorrect outputs, crashes, system errors, or other undesirable consequences. Failures are observed when users encounter issues while using the software.

1. Difference between Priority and Severity

Priority: Priority is a measure of the importance or urgency assigned to a specific bug or issue. It helps determine the order in which bugs should be addressed and fixed. The priority of a bug is typically set by the project manager, product owner, or a designated person responsible for prioritizing tasks. The priority level indicates how quickly the issue needs to be resolved, considering factors such as business impact, user needs, project timelines, and other relevant considerations. Priority levels are often defined using labels like high, medium, or low, or numerical values like 1, 2, 3.

Severity: Severity, on the other hand, is a measure of the impact or seriousness of a specific bug or issue on the software's functionality. It indicates the extent to which the bug affects the desired behavior of the system. Severity is usually determined by the tester or quality assurance team based on the observed behavior and the potential consequences of the bug. Severity levels are typically defined using labels such as critical, major, minor, or trivial. The severity level helps developers understand the impact of the bug and prioritize their efforts to fix critical or high-severity issues first.

1. What is Bug Life Cycle?

New/Opened: In this stage, a bug is identified or reported by a tester, user, or any other stakeholder. The bug is logged into a bug tracking system or tool with relevant details such as bug description, steps to reproduce, severity, and other necessary information.

Assigned: Once a bug is logged, it is assigned to the appropriate developer or development team responsible for addressing the bug. The bug is then marked as assigned and taken up for investigation and resolution.

In Progress: In this stage, the assigned developer works on reproducing the bug, analyzing the code or system behavior, and developing a fix. The bug is marked as "in progress" during this phase.

Fixed: Once the developer has successfully identified and implemented a fix for the bug, the bug is marked as "fixed." The developer may also provide additional information or comments about the fix.

Verified: After the bug is fixed, it goes through a verification process to ensure that the fix is effective and the bug no longer exists. A tester or quality assurance team member verifies the bug fix by retesting the system or executing specific test cases related to the bug. If the fix is confirmed, the bug moves to the next stage.

Closed: After the bug fix is verified, the bug is marked as "closed." It means that the bug is resolved, and no further action is required. The bug tracking system usually keeps a record of closed bugs for future reference.

Reopened: In some cases, a closed bug may be reopened if the issue resurfaces or the fix is found to be ineffective. If a bug is reopened, it goes back to the "assigned" or "in progress" stage for further investigation and resolution.

1. Explain the difference between Functional testing and NonFunctional testing

Functional Testing:

Objective: Functional testing primarily evaluates the system's behavior and functionality by testing its individual functions or features.

Focus: It focuses on verifying whether the software meets the specified functional requirements, including user interactions, data processing, and system responses.

Testing Approach: Functional testing involves designing and executing test cases based on functional specifications and user requirements.

Examples: Unit testing, integration testing, system testing, user acceptance testing, and regression testing are types of functional testing.

Test Criteria: Functional testing validates if the system performs as expected, provides accurate outputs, handles inputs correctly, and satisfies the functional requirements defined for the software.

Non-Functional Testing:

Objective: Non-functional testing assesses the software's performance, usability, reliability, and other quality attributes beyond its basic functionality.

Focus: It evaluates how the software performs in terms of speed, scalability, security, usability, compatibility, reliability, and other non-functional aspects.

Testing Approach: Non-functional testing involves designing and executing test cases that measure and evaluate the system's performance against predetermined benchmarks or criteria.

Examples: Performance testing, load testing, stress testing, security testing, usability testing, compatibility testing, and reliability testing are types of non-functional testing.

Test Criteria: Non-functional testing validates if the software meets specific performance targets, adheres to security standards, delivers a satisfactory user experience, and performs reliably under different conditions.

1. What isthe difference between the STLC (Software Testing Life Cycle) and SDLC (Software Development Life Cycle)?

Software Development Life Cycle (SDLC):

Definition: The SDLC is the overall process that defines the stages, activities, and tasks involved in developing a software application. It encompasses the entire software development process, starting from requirements gathering and analysis to system deployment and maintenance.

Focus: The SDLC focuses on the development and delivery of a software product or application that meets the specified requirements. It involves activities such as requirement analysis, design, coding, testing, and maintenance.

Key Stages: The common stages in the SDLC include requirement gathering, system analysis, design, coding, testing, deployment, and maintenance.

Participants: The SDLC involves various stakeholders, including business analysts, project managers, software architects, developers, testers, and end-users.

Software Testing Life Cycle (STLC):

Definition: The STLC is the process that defines the activities, tasks, and phases involved in testing a software application to ensure its quality, functionality, and performance.

Focus: The STLC focuses on verifying and validating the software application against the specified requirements and quality standards. It involves activities such as test planning, test case design, test execution, defect tracking, and test closure.

Key Stages: The typical stages in the STLC include test planning, test case development, test environment setup, test execution, defect reporting and tracking, and test closure.

Participants: The STLC involves testers, quality assurance engineers, test leads, and other individuals responsible for testing and ensuring the software's quality.

1. What is the difference between test scenarios, test cases, and test script?

Test Scenario:

A test scenario is a high-level description or outline of a particular test condition or situation. It defines the objective or goal of the test and provides a context for testing. Test scenarios focus on capturing the expected behavior or outcome of the system under specific conditions. They are usually written in a narrative format and can encompass multiple test cases.

Test Case:

A test case is a detailed set of conditions, inputs, and expected results designed to test a specific functionality or aspect of the software. It provides step-by-step instructions for executing the test, including preconditions, test steps, and expected outcomes. Test cases are derived from test scenarios and are more specific and granular in nature. They are often written in a structured format and can be executed independently.

Test Script:

A test script is a set of instructions or code that is used to automate the execution of test cases. Test scripts are written in programming or scripting languages and provide a sequence of actions to be performed during the test. Test scripts can interact with the software being tested, simulate user inputs, and validate expected outcomes automatically. Test scripts are primarily used in automated testing to increase efficiency and repeatability.

1. Explain what Test Plan is? What isthe information that should be covered.

A Test Plan is a document that outlines the objectives, scope, approach, and schedule of testing activities for a specific software project or release. It serves as a roadmap for the testing process and provides guidance to the testing team on how to proceed with the testing efforts. The information covered in a test plan may vary depending on the project's complexity and specific requirements, but here are the key components typically included:

Introduction:

Overview of the software project and its purpose.

Objectives and goals of testing.

Scope and boundaries of the testing effort.

Assumptions and dependencies.

**Test Strategy:**

Testing approach and methodologies to be followed.

Test levels and types to be performed (e.g., unit testing, integration testing, system testing, etc.).

Test techniques, tools, and environments to be utilized.

Test data management approach.

Risk assessment and mitigation strategies.

**Test Deliverables:**

List of test deliverables, such as test plans, test cases, test scripts, test data, and test reports.

Description of each deliverable and its purpose.

**Test Schedule:**

Timeline and milestones for testing activities.

Dependencies on other project activities.

Resource allocation and availability.

Test execution and reporting cycles.

**Test Environment:**

Description of the test environment, including hardware, software, and network configurations.

Setup and configuration instructions for the test environment.

Dependencies and integration points with other systems or components.

Test Execution:

Test case management approach.

Test case prioritization and selection criteria.

Test execution process, including how defects will be logged and tracked.

Test coverage and exit criteria.

Test Management:

Roles and responsibilities of the testing team members.

Communication and collaboration channels among team members and stakeholders.

Defect management process, including defect tracking and resolution.

Change management process for handling test-related changes.

Risks and Contingencies:

Identified risks and their potential impact on testing.

Mitigation strategies and contingency plans.

Actions to address identified risks during the testing process.

Sign-Off and Approval:

Criteria for test completion and sign-off.

Stakeholder approval process.

1. What is priority?

High Priority: Defects classified as high priority have a significant impact on the system's functionality or the user's ability to use the software effectively. These defects often result in system failures, critical errors, or severe usability issues that prevent users from performing essential tasks. High priority issues require immediate attention and resolution.

Medium Priority: Medium priority defects have a moderate impact on the system's functionality or user experience. While they may not cause critical failures, they can still affect important features or processes. Medium priority issues are important but may not require immediate resolution.

Low Priority: Low priority defects have a minimal impact on the system's functionality or user experience. They represent minor issues, cosmetic flaws, or non-critical enhancements that do not significantly impact the software's core functionality. Low priority issues are typically addressed after higher priority defects have been resolved.

1. What is severity?

Severity, in the context of software testing and bug tracking, refers to the extent of impact a defect or issue has on the system's functionality or the user's ability to use the software effectively. It helps determine the degree of seriousness or criticality of a defect. Severity is typically assessed by the testing team based on the observed impact of the defect.

1. Bug categories are…

Functional Bugs: These bugs are related to the functional requirements of the software. They occur when the system or application does not perform as intended or specified. Functional bugs may include issues such as incorrect calculations, improper data processing, or failure to execute certain functions.

User Interface Bugs: User interface (UI) bugs are related to the graphical user interface of the software. They involve issues with the visual presentation, layout, responsiveness, or usability of the user interface. UI bugs may include problems such as misaligned elements, overlapping text, unresponsive buttons, or inconsistent colors.

Performance Bugs: Performance bugs refer to issues that affect the performance and responsiveness of the software. These bugs may cause slow execution, excessive memory usage, high CPU utilization, or inefficient resource management. Performance bugs can lead to system slowdowns, delays, or crashes.

Compatibility Bugs: Compatibility bugs occur when the software does not function correctly or as expected on different hardware platforms, operating systems, web browsers, or versions of third-party software. These bugs can result in issues such as feature incompatibility, data corruption, or system instability on specific configurations.

Security Bugs: Security bugs are vulnerabilities or weaknesses in the software that can be exploited to compromise the confidentiality, integrity, or availability of the system or its data. Security bugs may include issues such as insufficient authentication, authorization flaws, input validation errors, or code injection vulnerabilities.

Usability Bugs: Usability bugs are related to the user experience and ease of use of the software. These bugs impact the user's ability to navigate, understand, or interact with the system effectively. Usability bugs may involve problems such as confusing user interfaces, unclear error messages, or inconsistent behavior.

Documentation Bugs: Documentation bugs refer to issues or errors in the software documentation, such as user manuals, help guides, or technical specifications. These bugs may include outdated information, incorrect instructions, missing details, or inconsistent documentation.

Localization Bugs: Localization bugs occur when the software fails to adapt or display correctly for different languages, regions, or cultural settings. These bugs may involve issues such as text truncation, incorrect translations, or cultural insensitivity.

1. Advantage of Bugzila .

Bug Tracking and Management: Bugzilla provides a centralized platform for tracking and managing software defects. It allows users to log, categorize, and prioritize bugs efficiently. The system maintains a comprehensive history of each bug, including status changes, comments, attachments, and related information, enabling effective collaboration and bug resolution.

Customization and Flexibility: Bugzilla is highly customizable and flexible, allowing organizations to adapt it to their specific needs. It offers extensive configuration options, including custom fields, workflows, and user roles, enabling organizations to tailor Bugzilla to match their bug tracking and management processes.

Collaboration and Communication: Bugzilla promotes collaboration and communication among team members involved in bug resolution. It allows users to comment on bugs, attach relevant files or screenshots, and engage in discussions. This facilitates effective communication between developers, testers, project managers, and other stakeholders, leading to faster bug resolution.

Search and Reporting Capabilities: Bugzilla offers robust search capabilities that allow users to filter and search for specific bugs based on various criteria, such as status, severity, priority, or keywords. It also provides reporting features, enabling users to generate custom reports and metrics to assess bug trends, project progress, and overall software quality.

Integration with Development Tools: Bugzilla integrates well with other development tools and systems, such as version control systems, project management tools, and continuous integration systems. This integration streamlines the bug tracking and resolution process by automatically linking bugs to relevant code changes, test cases, or project milestones.

Community Support and Extensibility: Bugzilla has a large and active user community, which means extensive community support and resources are available. Users can benefit from community forums, mailing lists, and documentation for guidance and assistance. Additionally, Bugzilla's extensibility allows users to develop and integrate custom plugins or extensions to enhance functionality based on their specific requirements.

Cost-Effective Solution: Bugzilla is an open-source tool, which means it is freely available and can be downloaded and used without licensing costs. This makes it a cost-effective option for organizations, particularly smaller teams or those with budget constraints.

1. Difference between priority and severity

Severity:

Severity refers to the impact or seriousness of a defect on the functionality or usability of the software.

It assesses the degree of impact a defect has on the system's core functionality, security, performance, or user experience.

Severity is typically assigned by testers or quality assurance professionals based on the observed impact and the deviation from expected behavior.

Severity levels may include critical, major, moderate, or minor, indicating the extent of impact on the system.

Priority:

Priority, on the other hand, relates to the urgency or importance of addressing a defect based on business or project requirements.

It determines the order in which defects should be resolved, considering factors such as business impact, project deadlines, customer needs, and stakeholder expectations.

Priority is usually set by project stakeholders, such as product owners, project managers, or business analysts, and can be subject to negotiation and agreement among the stakeholders.

Priority levels may include high, medium, or low, indicating the level of urgency or importance of addressing the defect.

1. What are the different Methodologies in Agile Development Model?

Scrum: Scrum is one of the most widely used Agile methodologies. It emphasizes iterative and incremental development. In Scrum, the work is divided into time-bound iterations called sprints, usually lasting 1-4 weeks. The team collaborates closely, and at the end of each sprint, a potentially shippable product increment is delivered.

Kanban: Kanban is a visual Agile methodology that focuses on managing work as it flows through the development process. It uses a Kanban board to visualize the workflow and limit work in progress (WIP) to optimize the flow of tasks. Kanban provides flexibility by allowing work to be pulled based on capacity rather than being timeboxed.

Extreme Programming (XP): XP is an Agile methodology that emphasizes high-quality software development practices. It promotes frequent releases, continuous integration, test-driven development (TDD), pair programming, and collective code ownership. XP aims to maximize customer satisfaction and adapt to changing requirements.

Lean Development: Lean development applies the principles of Lean manufacturing to software development. It focuses on delivering value to customers with minimum waste. It encourages continuous improvement, eliminating bottlenecks, and maximizing efficiency throughout the development process.

Dynamic Systems Development Method (DSDM): DSDM is an Agile methodology that provides an iterative and incremental framework for project delivery. It emphasizes the active involvement of stakeholders and focuses on delivering business value. DSDM provides a set of principles, roles, and practices to guide the development process.

Feature-Driven Development (FDD): FDD is an Agile methodology that organizes software development around feature teams. It follows a five-step process: domain walkthrough, development of an overall model, construction of a feature list, planning by feature, and iterative development. FDD places a strong emphasis on design and code quality.

Crystal: Crystal is a family of Agile methodologies that tailors the development approach to the project's unique characteristics. Crystal methodologies focus on teamwork, communication, and simplicity. They offer different guidelines and practices based on the project size, criticality, and team size.

1. Write a scenario of only Whatsapp chat messages

• Checking that user can update chat wallpaper

• checking that user can send message to other user

• checking that user can unsend the message for him and for everyone

• checking that user can insert emoji or stickers in chat

• checking by inserting video in chat

• checking by sending location to other person

• checking by keeping it blank and tapping enter

• checking wheater it shows single tick,double tick and blue tick(message seen status).

1. POSITIVE AND NEGATIVE OF GMAIL RECEIVING MAIL

* POSITIVE
* Getting new mails
* New message and unread msg would be highlighted
* After reading message,it would b seen inn normal form
* NEGATIVES
* Not getting new mails
* Not able to open mails
* Message is still highlighted after opening

POSITIVE AND NEGATIVE OF FLIPKART

* POSITIVE
* The cost and name of product is perfectly writtened
* Checking the discount for the product
* Verifying offers
* User is able to write pincode
* Product is able to moved to cart
* Product can be bought directly from buy now
* NEGATIVES
* Not showing any offers
* Not able to write pincode
* Add to cart nott working

Q41 TEST SCENARIO ON LIFT

Lift

1. When pressing on button,The button is not responding.so due to this,the lift is not usable.
2. Lift calling button is not working,so we cannot call lift and makes it unusable.
3. Lift’s light is not working so it is hard to use lift in dark.
4. Lift door is opening and closing as expected
5. There’s always need current floor indicator inside the lift to know the status of current floor
6. Lift material quality is must b as expected
7. must be able to visit all floors which is mentioned inside lift
8. The lift must visit the particular floor from where it get called.
9. When tapped on lift button from multiple floor..it must check the nearest floor to visit
10. Lift must be in working condition
11. PEN SCENARIO
12. Pen’s point is rough so it would not be good to write from that pen
13. While writing..it leaks ink so it can make our hands dirty
14. Pens point is broken so we cannot use it to write anything nd also spread ink everywhere
15. Shape of pen is not proper so it may hurt us while writing
16. Refill of pen is empty so we cannot write anything using it
17. Checking that the pen’s ink is waterproof or not
18. Pens body material quality should be good
19. Pen must not spread while writing
20. Checking by removing old refill and inserting new refill.
21. The content written by pen should be bright and readable by anyone.
22. PEN STAND SCENARIO
23. Checking the size and quality of the pen stand
24. Checking by putting multiple pens in the pen stand
25. Checking the capacity of the pen stand
26. Checking whether the pen stand is unbreakable or not
27. Checking the weight of the pen stand
28. Checking the shape of pen stand to see,it can hold pens or not.
29. Checking the material of the pen stand
30. Checking durability by doing toughness test
31. Checking the space consumption of the pen stand
32. Checking wheather the pen stand is usable or not.
33. SCENARIO OF ATM

* MACHINE MUST BE IN WORKING CONDITION
* TOUCH OF ATM MUST BE RESSPONSIVE
* BUTTON MUST BE IN WORKING CONDITION
* PIN MUST NOT BE VISIBLE WHILE TYPING
* Cash must be coming out from atm
* ATM’s Internal software must not be visible to user

1. SCENARIO OF WHATS APP GENERATE GROUP

* New group option must be clickable
* It must show the list of contacts,which is on whats app
* Contact must be selected by user
* There need a search bar to search the contacts
* Enter button must be in working condition

1. INSTA VC WITH FRIEND SCENARIO

* Vc option must be clickable
* User must be connected to the internet
* There must be option to switch from front cam to main cam
* There’s must need a option to end vc
* There must need timer which track the time of vc

1. WHATS APP PAYMENT SCENARIO

* Payment option must be clickable
* Active network connection required
* It must redirect user to bank portal page
* User must be able to input digit
* There’s must need indication to see the status of money

1. SCENARIO OF WRIST WATCH

* Watch must be in working condition
* Watch strap quality must be upto mark
* Checking ruggedness of the watch
* Checking wheater watch having any ip rating
* Glass must be of high quality

1. SCENARIO ON CHAIR

* Chair must not cause any back pain when sitting for long time.
* Chair must be of high quality
* Checking the height and weight of chair
* Checking that the chair is waterproof or not
* Checking the stability of chair
* Checking wheather the chair is foldable or not

1. SCENARIO OF DOOR

* Checking the color off the door
* Checking the quality of door
* Checking from where the door opens like,open inside,outside or both.
* Checking wheather door having lock or not
* checking the type of lock in door
* checking the durability of door
* checking wheather the door is smart door or normal door
* checking wheather the door is water proof or not
* checking wheather the door makes noice opening and closing

1. SCENARIO OF MICROWAVE OVEN

* checking that the produt powers on and is in working condition
* checking whether the food heats at ideal temperature
* verifying that the microwave support battery or not
* verifying that the display working as expected
* checking that the door of microwave opens and close without making any noice
* checking that timer onn microwave is working properly
* checking the working of all buttons are as expected or not
* checking by inserting different food items
* verifying touch panel wheater it is touchable or not
* checking by ssetting different settings

1. COFFEE VENDING MACHINE

* verifying that the product powers on and is in working condition
* checking the quality of the product
* checking the minimnum power requirements to run the device
* checking buttons of the product
* checking touch paell of the product
* checking wheater it is working as expected
* checking that powe button is working or not
* checking the operating temperature of product
* checking ingrediants which is needed to make coffee