1. **What is Exploratory Testing ?**

* Exploratory Testing is an approach to software testing that emphasizes the personal freedom and responsibility of the individual tester to continuously optimize the quality of their work by treating test-related learning, test design, test execution, and test result interpretation as mutually supportive activities that run in parallel throughout the project..

1. **What is traceability matrix ?**

* Traceability Matrix is a document that helps match requirements with test cases. It shows which requirements have test cases written for them and helps make sure that nothing is missed during testing. It is used to track the progress of testing and to check if all the features are tested properly.

1. **What is Boundary value testing ?**

* **Boundary Value Testing** is a type of software testing where you focus on testing the values at the "boundaries" of input ranges. Instead of testing only normal values, you check the minimum, maximum, just below, and just above the boundaries to see if the system handles edge cases correctly. This helps find errors that might happen at the extreme ends of input values.

1. **What is Equivalence partitioning testing ?**

* **Equivalence Partitioning Testing** is a technique where you divide all possible input values into groups (called "partitions") that should behave the same way. Instead of testing every possible input, you test just one value from each group, assuming that all values in that group will work similarly. This helps save time while ensuring that key scenarios are tested.

1. **What is Integration testing ?**

* Integration Testing is a type of software testing where different parts or modules of a system are tested together to make sure they work well when combined. After testing individual parts separately (called unit testing), integration testing checks if the modules communicate and function correctly when they are put together.

1. **What determines the level of risk ?**

* **There are two types of risk are involve in this determines**

**(1) Project Risk:** One significant project risk is the abrupt risk is the abrupt departure of a senior team member, which can disrupt project continuity and knowledge flow. To effectively manage this risk, it is essential to assess both the likelihood of occurrence and the potential impact on the project, typically rated on a scale , the overall risk score would be from 1 to 10. For instance, if the likelihood is rated as 7 and the impact as 8, the overall risk score would be 56, indicating a high level of concern. Mitigation strategies should be implemented to address this risk, such as conducting knowledge transfer sessions to ensure that critical information is shared is shared among team members. Additionally, having a buffer tester in place can help maintain project momentum and quality in the event of a team member’s sudden departure.

**(2) Product Risk:** An example of a Product risk is the failure of a flight reservation system to install correctly in the test environment. This risk cand lead to delays in testing and ultimately affect the project timeline. Similar to project risks, the likelihood and impact of this product risk should be evaluated on a scale of 1 to 10. For instance, if the likelihood is assessed at 6 and the impact at 9, the overall risk score would be 54, indicating a significant risk that needs to be addressed. To mitigate this risk, conducting smoke or sanity testing can be an effective strategy. This approach ensures that the essential functionalities of the system are operational before proceeding with more extensive testing. Consequently, adjustments to the project scope may be necessary to include these testing procedures, thereby enhancing the overall reliability of the product.

1. **What is Alpha testing ?**

* Alpha testing is a crucial phase in the software development process, primarily conducted by developers at the software development site. This type of testing is typically performed within the organization and is not open to the market or the public. While it is mainly carried out by the development team, it can also involve an independent Testing team. Alpha testing focuses on the software application and project, ensuring that it meets the required standards before moving to the next phase.
* This testing is conducted in a virtual environment, allowing for a controlled setting to identify any issues. It is considered a form of Acceptance testing, as it assesses whether the software meets the acceptance criteria set by stakeholders. Additionally Alpha testing encompasses both white box testing and black box testing methodologies, providing a comprehensive evaluation of the software’s functionality and performance. Overall, Alpha testing is an essential step in ensuring quality and reliability of software before it is released to users.

1. **What is beta testing ?**

* Beta testing is a crucial phase of software development where real customers test the product in their own environment, using their own data. This type of testing is always performed outside the developing organization and is not done by the internal testing team. Instead, it’s typically open to a wider market or public, especially for new software products, allowing for feedback from a diverse user base. It is carried out in a real – time environment, meaning users interact with the software as they would in their daily tasks, without the presence of the development team. Beta testing is a from of Acceptance Testing**,** specifically **User Acceptance Testing (UTA),** as it confirms whether the software meets user needs and expectations before its official release. It’s often referred to as “pre-release” testing and is exclusively a form of Black Box Testing**,** where testers only interact with the software’s interface and functionality, without knowledge of its internal code or structure.

1. **What is component testing ?**

* Component Integration Testing is a vital phase in software development that focuses on ensuring different parts (components) of a software system work together correctly. It's performed *after* individual components have been tested in isolation (component testing/unit testing) and *before* the entire system is tested as a whole (system testing).
* The main goal of this type of testing is to expose defects in the interfaces and interactions between these integrated components. Often, different teams or individuals develop separate components, and Component Integration Testing ensures that these pieces, despite being specified at different times, can seamlessly communicate and function together.
* During this testing, it's crucial to cover negative cases alongside positive ones, because components might make incorrect assumptions about the data they receive from or send to other components.
* Component Integration Testing is typically formal, meaning that detailed records of test design and execution are maintained. Appropriate techniques for this phase include Functional Testing, using Black Box Testing methods to verify that the interfaces meet their requirements. Additionally, Non-functional Testing may be applied where relevant, for instance, to assess the performance or reliability of component interfaces.

1. **What is functional system testing ?**

* Functional System Testing focuses on verifying that the entire system or its components perform as specified by the functional requirements. These requirements, which outline specific functions a system must execute, can be documented as text or models. There are generally two main test approaches to Functional System Testing: Requirement-Based Functional Testing, which directly links test cases to defined requirements, and Process-Based Testing, which evaluates the system's adherence to specified business or operational processes. Functional System Testing focuses on verifying that a software system or its components perform their intended functions as specified. These functional requirements, which describe "what" the system must do, can be documented in various forms, such as text documents or models.

1. **What is Non-Functional Testing ?**

* Non-Functional Testing This type of testing is versatile and can be performed at all test levels, not just during specific Non-Functional System Testing phases. A key element of non-functional testing is its ability to measure characteristics of the system or software that can be quantified on a varying scale. For instance, it's used to assess aspects like response times for performance testing. To elaborate, non-functional testing encompasses a range of specific test types, including performance testing (which aims to check and fine-tune system response times to an acceptable level), load testing (specifically carried out to check system performance under varying numbers of users accessing it), stress testing, usability testing, maintainability testing, reliability testing, and portability testing, among others. Ultimately, it ensures the software is robust, efficient, and user-friendly in real-world conditions.

1. **What is GUI Testing ?**

* Graphical User Interface (GUI) testing is simply the process of checking and validating the visual and interactive parts of a software system. When you perform GUI testing, you're essentially looking at everything a user sees and clicks on – this includes examining the screens, ensuring all controls like menus, buttons, and icons work correctly, and verifying the appearance and functionality of various bars (like toolbars and menu bars), as well as dialog boxes and windows. It's all about making sure the user interface looks right and behaves as expected when someone interacts with the software.

1. **What is Adhoc testing ?**

* Ad-hoc testing is an informal and unstructured approach to software testing that primarily aims to break the system and uncover defects quickly. Unlike formal testing, it does not follow any pre-defined test cases or structured design techniques; in fact, testers often don't create any test cases at all. This method is particularly effective when performed by testers with extensive knowledge and experience of the system under test, as they randomly explore the application without relying on specific business requirement documents. The main goal is to find defects through this random exploration of any part of the application..
* A key technique often used to achieve ad-hoc testing's objectives is Error Guessing. This technique leverages the intuition and vast experience of skilled testers, encouraging them to "guess" the most likely sources of errors or think of situations where the software might fail to cope. Testers who are naturally good at finding weaknesses, or those with significant experience with a particular system, are adept at postulating potential errors. This is why Error Guessing can be highly effective when used to complement more formal test design techniques, as it often uncovers defects that might otherwise be missed. It also provides a significant time-saving benefit by allowing experienced testers to quickly pinpoint likely problem areas, maximizing defect discovery through their informed assumptions and guesses.

1. **What is load testing ?**

* Load testing is a specialized form of performance testing designed to assess a system's behavior under significant stress or heavy user demand. Its primary objective is to simulate real-life usage conditions by subjecting an application, such as a website, to a range of increasing loads. This process helps determine the crucial point at which the system's response time begins to degrade or, in severe cases, fails entirely.

1. **What is stress Testing ?**

* Stress testing is a vital type of software testing that pushes a system far beyond its normal operating limits and specifications to determine how and when it fails. The primary goal is to assess the system's stability, reliability, robustness, and error handling capabilities under extremely heavy load conditions. It's often performed by subjecting the system to an excessive volume of users or data, such as putting large numbers of items beyond storage capacity, executing complex database queries, or providing continuous input to the system.

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

* White Box Testing, also known as structure-based testing, glass-box testing, or open-box testing, is a testing methodology that involves a detailed investigation of the internal structure, logic, and working of a software component or system. Unlike black box testing which focuses on external behavior, white box testing concentrates on "how the software does it," requiring testers to possess a deep understanding of the code's implementation and internal mechanisms.
* **The types of white box testing.**
* **Code Analysis/Review:** This involves directly analyzing the logic by reading the code. This is fundamental to understanding the internal structure and flow.
* **Code Optimization Review:** Evaluating the code to suggest improvements or more efficient alternatives. This requires deep insight into the code's

implementation.

* **Debugging:** This is explicitly mentioned for "Desktop Based Testing" and "Game Based Testing" ("When we debug the code when we writing" and "When some debug the code and play at that time game"). Debugging involves stepping through code to find and fix defects, which is a core White Box activity.
* **Internal Logic Verification/Structural Testing:** While not explicitly named as a separate "type," the actions described (e.g., "Analyze the logic by reading the code," "check in code" for remote device connections in game testing) inherently involve verifying the internal logic and structure of the application across different platforms.

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

* Black-box testing, also known as specification-based testing or input/output driven testing, is a fundamental approach where software is tested without any knowledge of its internal structure, code, or implementation. Testers view the software as a "black box," focusing purely on "what the software does" rather than "how it does it."
* **Black box testing techniques**
* **Equivalence Partitioning:** Equivalence Partitioning **(EP)** is a powerful black box test design technique that aims to reduce the number of test cases by treating groups of inputs as equivalent. The core idea is to select just one representativeinput from each partition (or "class") of data, assuming that if one value in that partition works, all others will too. This technique can be effectively applied at alllevels of testing.
* **Boundary Value Analysis :** Boundary Value Analysis (BVA) is a focused test case design methodology that refines and is typically used in conjunction with Equivalence Partitioning. While Equivalence Partitioning selects one representative from a group, BVA concentrates testing efforts specifically on the extreme ends or limits of valid and invalid ranges within those equivalence classes..
* **State transition testing :** State Transition Testing is a powerful black box test design technique used when a component or system's behavior depends on its current condition or "state." It's based on the concept of a 'finite state machine,' where the system can exist in a limited number of defined states, and transitions between these states are triggered by specific events according to predetermined rules. A key indicator that a system is a good candidate for this technique is when you receive a different output for the same input, depending on what has occurred previously
* **Syntax or Pattern Testing :** Syntax or Pattern Testing is a focused technique used to verify that a system correctly handles data inputs that are expected to conform to a predefined structure or format. Without needing to look at the internal code, testers focus on providing inputs that either perfectly match (valid syntax/pattern) or deliberately violate (invalid syntax/pattern) these rules.

The primary goal is to ensure the software intelligently **accepts valid data** (e.g., a correctly formatted email address, a date in "DD-MM-YYYY" format, or a phone number with the right number of digits) and **gracefully rejects or provides appropriate feedback for invalid data** (e.g., an email missing the "@" symbol, a date with a non-existent month, or an alphanumeric character in a numeric-only field). This type of testing is crucial for data integrity and user experience, as it confirms the system's ability to process structured information according to its design specifications.

**19. Mention what are the categories of defects ?**

* **High Priority & High Severity:** An error which occurs on the basic functionality of the application

and will not allow the user to use the system. (E.g. A site maintaining the student details, on saving

record if it, doesn’t allow to save the record then this is high priority and high severity bug.)

* **High Priority & Low Severity:** The spelling mistakes that happens on the cover page or heading or title of an application.
* **High Severity & Low Priority:** An error which occurs on the functionality of the application (for

which there is no workaround) and will not allow the user to use the system but on click of

link which is rarely used by the end user.

* **Low Priority and Low Severity:** Any cosmetic or spelling issues which is within a paragraph

or in the report (Not on cover page, heading, title).

**20. Mention what big bang testing is ?**

* Big Bang integration testing is a testing approach where all components or modules of a system are integrated simultaneously, and then the entire system is tested as a complete unit. The main advantage of this method is that all individual development work is completed before the integration testing phase even begins. However, its significant disadvantage lies in the fact that because everything is integrated at once and testing happens so late in the process, it can become time-consuming and exceptionally difficult to isolate and trace the root cause of any failures that arise.

1. **What is the purpose of exit criteria ?**

* The purpose of exit criteria is to define the specific conditions that must be met before testing activities can be concluded. These criteria help ensure that the software has achieved an acceptable level of quality, risk is minimized, and the project is ready to move to the next phase or go live. Exit criteria are used to guide decision making and measure whether key goals- such as requirement coverage, defect density, cost limits, and schedule constraints- have been satisfied. They help

Prevent premature release and ensure that stakeholders are informed about any residual risks.

**22. When should "Regression Testing" be performed ?**

* **Regression testing should be performed:**

1. ** After Modifications:** Whenever the system undergoes changes like bug fixes, new feature additions, or code modifications, to ensure these changes do not introduce defects elsewhere in the software.
2. **After Environment Changes :** When there are updates or modifications in the software environment that could impact the system.
3. **During Bug-Fix Releases:** Especially in the maintenance phase when fixing defects, regression testing is necessary to ensure that the fixes haven’t affected other parts of the application.
4. **When System Stabilizes**: Once the system is stable, regression testing ensures that no new defects have been introduced in areas that weren’t directly changed.
5. **To Ensure Unintended Side-Effects Are Not Introduced**: After any modifications, you want to ensure the changes haven’t caused any unintended issues in other parts of the software.
6. **At All Testing Levels**: Regression testing should be carried out at various levels of testing (unit, integration, system, and acceptance.
7. **When the Test Suite Evolves**: Over time, the regression test suite evolves, and as it grows, it becomes more suitable for automation to save time and effort in frequent testing.

**23. What is 7 key principles? Explain in detail ?**

**1. Testing shows presence of Defects**

**2. Exhaustive Testing is Impossible!**

**3. Early Testing**

**4. Defect Clustering**

**5. The Pesticide Paradox**

**6. Testing is Context Dependent**

**7. Absence of Errors Fallacy**

**1. Testing shows presence of Defects :** This principle states that testing can only show the presence of defects, not their absence. Even if a software system passes all tests, it doesn't guarantee that the software is defect-free. Testing can reveal defects, but it cannot prove that the software is entirely defect-free. In other words, no matter how much testing is done, there may still be defects lurking that have not been discovered.

* **Why is this important**? This principle emphasizes the importance of understanding that testing is not a silver bullet. Just because software passes tests doesn't mean it's perfect.
* **Example**: A test suite may catch some bugs, but there could be other bugs that were missed, or new defects introduced later that weren't caught in the tests.

**2. Exhaustive Testing is Impossible !**

* This principle states that it is impossible to test every possible scenario or combination in a complex system, especially when dealing with large, dynamic applications.
* **Why is this important ? = ** Exhaustive testing would mean testing all possible inputs, configurations, and conditions for the software, which is not feasible due to time, cost, and resource limitations.

**Example :** In a system that takes two numbers as input, there are infinite possibilities of values for those numbers. Trying to test all combinations would be impractical. Instead, you prioritize testing based on risk, critical paths, and areas that are most likely to contain defects.

**3. Early Testing :**  Testing activities should start as early as possible in the software or system development life cycle, and should be focused on defined objectives. Testing activities should start as early as possible in the development life cycle These activities should be focused on defined objectives – outlined in the Test Strategy. Remember from our Definition of Testing, that Testing doesn’t start once the code has been written !

* **Why is this important ? :** Early testing helps identify defects as soon as they are introduced, reducing the cost and effort to fix them later. It's easier and cheaper to fix a problem earlier in the development process than after the software is finished.
* **Example**: In Agile development, testing starts with the first user story and continues throughout the development cycle. This helps catch issues early and avoid “defect snowballing” later.

**4**.  **Defect Clustering :** A small number of modules contain most of the defects discovered during pre-release testing or are responsible for the most operational failures. Defects are not evenly spread in a system They are ‘clustered’ In other words, most defects found during testing are usually confined to a small number of

Modules. Similarly, most operational failures of a system are usually confined to a small number of modules. An important consideration in test prioritization!

* **Why is this important? :** understanding that defects tend to concentrate helps testers prioritize testing efforts. If defects are concentrated in a specific module or functionality, you can focus more testing on that area.
* **Example**: In a banking application, the payment processing module may contain more defects than the user authentication module. Focus your testing resources more on payment processing.

**5. The Pesticide Paradox** : If the same tests are repeated over and over again, eventually the same set of test cases will no longer find any new defects. To overcome this “pesticide paradox”, the test cases need to be regularly reviewed and revised ,and new and different tests need to be written to exercise different parts of the software or system to potentially. find more defects. Testing identifies bugs, and programmers respond to fix them. As bugs are eliminated by the programmers, the software improves As software improves the effectiveness of previous tests erodes.

* **Why is this important? :** You need to continuously update and expand your test cases to account for new types of defects or scenarios that were not previously covered. This encourages constant test suite evolution.
* **Example :** If your test suite only checks for login functionality, eventually, you’ll stop finding defects in that area. You would need to expand your tests to cover more complex scenarios, like concurrent logins or session expiration.

**6. Testing is Context Dependent. :** Testing is inherently context-dependent, meaning that the results and interpretations of tests can vary significantly based on the specific circumstances in which they are conducted. This includes factors such as the environment, the population being tested, the purpose of the test, and the conditions under which the test is administered.

**Why is this important ? :** The context in which a test is performed can influence its validity and reliability. For example, a test that is effective in one demographic may not yield the same results in another due to cultural, social, or economic differences. The context in which a test is performed can influence its validity and reliability. For example, a test that is effective in one demographic may not yield the same results in another due to cultural, social, or economic differences.

* 1. **Absence of Errors Fallacy. :** The Absence of Errors Fallacy refers to the mistaken belief that if no errors are detected in a test or process, it must be functioning correctly or producing valid results. This fallacy can lead to overconfidence in the reliability of a system or process, ignoring the possibility of undetected errors or flaws.

**Why is this important** : Believing that the absence of errors equates to correctness can lead to complacency. Organizations may neglect necessary reviews or improvements, assuming that everything is functioning as intended.  In fields such as software development, medicine, and engineering, the absence of errors does not guarantee that a product or service is free from defects. Continuous testing and validation are essential to ensure quality.

**24.Difference between QA v/s QC v/s Tester.**

|  |  |  |
| --- | --- | --- |
| **S.N.** | **Quality Assurance** | **Quality Control** |
| **1** | Activities which ensure the  implementation of processes,  procedures and standards in  context to verification of  developed software and intended  requirements**.** | Activities which ensure the  verification of developed  software with respect to  documented (or not in some  cases) requirements. |
| **2** | Focuses on processes and  procedures rather than conducting  actual testing on the system. | Focuses on actual testing by  executing Software with intend  to identify bug/defect through  implementation of procedures  and process. |
| **3** | Process oriented activities. | Product oriented activities. |
| **4** | Preventive activities. | It is a corrective process. |
| **5** | It is a subset of Software Test Life  Cycle (STLC). | QC can be considered as the  subset of Quality Assurance. |
| **6** | Focuses on processes and procedures rather than conducting actual testing on the system. | Focuses on actual testing by executing software with the intent to identify bugs/defects through the implementation of procedures and processes. |
| **7** | Preventive activities aimed at ensuring quality in the development process. | Corrective process aimed at identifying and fixing defects in the final product. |
| **8** | Testing is not the primary focus; it is more about ensuring quality processes. | Testing is the primary focus, aimed at identifying defects in the product. |

**25. Difference between Smoke and Sanity**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Smoke Testing** | **Sanity Testing** |
| **1** | Smoke Testing is performed to ascertain that the critical functionalities of the program is working fine | Sanity Testing is done to check the new functionality / bugs have been fixed |
| **2** | To verify the "stability" of the system in order to proceed with more rigorous testing. | To verify the "rationality" of the system in order to proceed with more rigorous testing. |
| **3** | This testing is performed by developers or testers. | Sanity testing is usually performed by testers. |
| **4** | Smoke testing is usually documented or scripted. | Sanity testing is usually not documented and is unscripted. |
| **5** | Smoke testing is a subset of Regression testing. | Sanity testing is a subset of Acceptance testing. |
| **6** | Smoke testing exercises the entire system from end to end. | Sanity testing exercises only the particular component of the entire system. |
| **7** | Smoke testing is like a General Health Check-Up, assessing the overall health of the system. | Sanity Testing is like a specialized health check-up, focusing on specific areas of concern. |

1. **Difference between verification and Validation.**

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Verification** | **Validation** |
| **Definition** | The process of evaluating  work-products (not the actual  final product) of a development  phase to determine whether  they meet the specified  requirements for that phase**.** | The process of evaluating software during or at the end of the development  process to determine whether it  satisfies specified business  requirements. |
| **Objective** | To ensure that the product is  being built according to  the requirements and  design specifications. In other  words, to ensure that work  products meet their  specified requirements. | To ensure that the product is  being built according to  the requirements and  design specifications. In other  words, to ensure that work  products meet their  specified requirements. |
| **Question** | Are we building the product right? | Are we building the right product? |
| **Evaluation Items** | Plans, Requirement Specs, Design  Specs, Code, Test Cases | The actual product/software. |
| **Activities** | ∙ Reviews  ∙ Walkthroughs  ∙ Inspections | ∙ Testing |

**27.** **Explain types of Performance testing.**

* Performance Testing is a software testing process used for testing the speed, response time, stability ,reliability, scalability, and resource usage of a software application under a particular workload. The main purpose of performance testing is to identify and eliminate the performance bottlenecks in the software application.
* **Types of Performance Testing**

**Load testing –** checks the application’s ability to perform under anticipated user loads. The objective is to identify performance bottlenecks before the software application goes live.

**Stress testing -** involves testing an application under extreme workloads to see how it handles high traffic or data processing. The objective is to identify the breaking point of an application.

**Endurance testing -** is done to make sure the software can handle the expected load over a long period of time.

**Spike testing** - tests the software’s reaction to sudden large spikes in the load generated by users.

**Volume testing -** Under Volume Testing large no. of. Data is populated in a database, and the overall software system’s behavior is monitored. The objective is to check software application’s performance under varying database volumes.

**Scalability testing** - The objective of scalability testing is to determine the software application’s effectiveness in “scaling up” to support an increase in user load. It helps plan capacity addition to your software system.

**28. What is Error, Defect, Bug and failure ?**

“A mistake in coding is called error ,error found by tester is called defect, defect accepted by development team then it is called bug, build does not meet the requirements then it is failure”

**Error**: A discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition. This can be a misunderstanding of the internal state of the software, an oversight in terms of memory management, confusion about the proper way to calculate a value, etc.

**Defect:** Commonly refers to several troubles with the software products, with its external behavior or with its internal features.

**Bug**: A fault in a program which causes the program to perform in an unintended or unanticipated manner. See: anomaly, defect, error, exception, and fault. Bug is terminology of Tester.

**Failure**: The inability of a system or component to perform its required functions within specified performance requirements. See: bug, crash, exception, and fault.

**29. Difference between Priority and Severity.**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | **Aspect** | **Severity** | **Priority** | | --- | --- | --- | | **Definition** | Severity is absolute and customer-focused. It defines the extent to which the defect affects the system. | Priority is relative and business-focused. It defines the order in which a defect should be resolved. | | **Focus Area** | Focuses on the impact of the defect on the software/system. | Focuses on the urgency to fix the defect based on business needs. | | **Set By** | Usually set by Testers based on technical judgment. | Usually set by TestLeadsorManagers based on customer or business needs. | | **Example** | Application crashes when a remote link is clicked – HighSeverity, LowPriority. | Company name misspelled on homepage – Low Severity, High Priority. | | **Types** | - Critical- Major- Minor- Trivial | - Critical- High- Medium- Low | | **Impact** | Deals with how badly the defect breaks the functionality. | Deals with how **soon** the defect needs to be fixed. | | **Fix Timing** | Fix depends on damagelevel to system, not urgency. | Fix depends on urgencyandimportance to business. | | **Dependency** | Independent of project deadlines and customer delivery. | Dependent on projecttimelines, delivery schedules, and client priorities. | |  |  |

1. **What is Bug Life Cycle?**

* The **Bug Life Cycle**, also known as the **Defect Life Cycle**, is the process that a software bug goes through from the time it is identified until it is fixed and closed. It starts when a tester finds a defect and logs it in the system. The bug is first given a status like "New" and then assigned to a developer. The developer reviews the bug, and if valid, changes the status to "Open" and starts working on it. Once the issue is fixed, the status is updated to "Fixed" and sent back to the tester for verification. If the tester confirms the fix, the bug is marked as "Closed." If the issue still exists, it may be marked as "Reopened." Sometimes, a bug may be marked as "Rejected" (if not valid), "Deferred" (to be fixed later), or "Duplicate" (already reported). This cycle helps track and manage bugs systematically to ensure software quality.

**31. Explain the difference between Functional testing and Non Functional testing.**

| * Aspect | * Functional Testing | * Non-Functional Testing |
| --- | --- | --- |
| * Definition | * Testing based on the specifications of what the system should do. | * Testing the attributes of the system that relate to how the system performs. |
| * Purpose | * To verify that each function of the application behaves as expected. | * To verify system performance, reliability, usability, and other non-functional attributes. |
| * Focus | * Focuses on functions and features. | * Focuses on performance, load, stress, scalability, usability, etc. |
| * Testing Type | * Black-box testing approach (based on external behavior). | * Can involve black-box testing but focuses on quantitative measures and system characteristics. |
| * Concerned With | * Checks what the system does, not how it does it. | * Checks how the system works under specific conditions. |
| * Test Levels | * Can be applied at all test levels (e.g., Unit, Integration, System Testing). | * Also applied at all test levels, especially in system testing. |
| * Examples | * - Verifying login functionality- Checking form submissions- Validating APIs | * - Performance Testing- Load Testing- Stress Testing- Usability and Portability Testing |
| * Test Basis | * Based on requirement specification, use cases, and functional documents. | * Based on system behavior metrics, SLAs, performance benchmarks, etc. |
| * Involves | * Inputs are given, outputs are verified, and results are compared with expected outcomes. | * System is tested under different loads or usage patterns to check behavior. |
| * Tools | * Can be performed manually or with automation tools like Selenium. | * Often requires specialized tools like JMeter, LoadRunner, etc. |

1. **To create HLR & Test Case of Instagram first page**

* [**https://docs.google.com/spreadsheets/d/1BmZan-0mxv9x1aUgxf91stlPZhqAT\_tvMRT3fsQj0gM/edit?usp=sharing**](https://docs.google.com/spreadsheets/d/1BmZan-0mxv9x1aUgxf91stlPZhqAT_tvMRT3fsQj0gM/edit?usp=sharing)

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

| Aspect | SDLC (Software Development Life Cycle) | STLC (Software Testing Life Cycle) |
| --- | --- | --- |
| * Focus | Entire software development process | Only on software testing process |
| * Main Objective | To develop a functional software application | To ensure the software meets quality and functional requirements |
| * Phases Included | Planning, design, coding, testing, deployment, maintenance | Requirement analysis, test planning, test case design, execution, closure |
| * Involvement | Involves developers, project managers, designers, and testers | Primarily involves QA testers and test engineers |
| * Start Time | Starts with business idea and requirement gathering | Starts after software requirements are finalized |
| * Outcome | A fully developed and deployed software system | Verified and validated software ready for release |
| * Relation | Broader process that includes STLC as a phase | A part of SDLC focused only on testing |
|  | | |

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

|  |  |  |  |
| --- | --- | --- | --- |
| Aspect | Test Scenario | Test Case | Test Script |
| Definition | High-level description of a feature or functionality to be tested | A detailed set of steps with inputs, expected results, and conditions for testing a scenario | A step-by-step executable set of instructions (manual or automated) to run a test case |
| Purpose | To identify what needs to be tested | To describe how the scenario should be tested | To execute the test (either manually or through automation) |
| Level of Detail | Low – only a general idea of what to test | Medium to high – includes test steps, input data, expected output | High – includes actual code or detailed manual steps for running tests |
| Format | Sentence or brief statement | Documented in test management tools (like Excel, TestRail, etc.) | Script code (in tools like Selenium, UFT) or written manual instructions |
| Who Creates It | Test Analyst / QA Team | Testers / QA Engineers | Automation Testers (for automated) or Manual Testers (for manual) |
| Automation | Not applicable | Can be manual or automated | Generally used in automation testing |
| Dependencies | Based on business requirements or user stories | Based on test scenarios | Based on test cases |
| Example | "Verify user can log in with valid credentials." | Test Case ID: TC001 Steps: Enter username and password → Click login Expected: Dashboard loads | Selenium code to open browser, enter login details, and verify dashboard |
| Reusability | Can be reused across multiple test cases | Specific to the scenario; reusable with some modification | Often reusable in automated test suites with parameterization |

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

* **What is a Test Plan?**

A Test Plan is a formal document that outlines the scope, approach, resources, schedule, and activities intended for testing a software product. It acts as a blueprint or roadmap for the testing process, guiding the testing team on *what to test*, *how to test*, *when to test*, and *who will do the testing*. A well-prepared test plan ensures that the testing process is organized, systematic, and traceable.

The purpose of the test plan is to ensure that all aspects of testing are thoroughly considered and communicated to stakeholders. It helps in identifying risks, managing responsibilities, and achieving quality goals within the defined timeline.

| **Section** | **Description** |
| --- | --- |
| **1. Test Plan ID** | A unique identifier for the test plan document |
| **2. Introduction** | Overview of the application under test and the purpose of the test plan |
| **3. Scope of Testing** | Defines what features and functionalities are in-scope and out-of-scope for testing |
| **4. Objectives** | The goals and what the testing aims to achieve |
| **5. Test Strategy** | High-level approach to testing (e.g., types of testing, techniques to be used) |
| **6. Resources and Responsibilities** | Names of team members involved and their roles in the testing process |
| **7. Test Environment** | Description of hardware, software, and tools required for testing |
| **8. Test Deliverables** | List of documents and reports to be produced (e.g., test cases, test data, defect reports) |
| **9. Test Schedule** | Timeline or milestone-based schedule for test activities |
| **10. Entry and Exit Criteria** | Conditions to begin testing (entry) and conditions to stop testing (exit) |

**36. What is priority ?**

* In software testing priority refers to the importance or urgency of fixing a defect or executing a test case. It helps the development and testing teams decide which issues should be addressed first, especially when there are many bugs reported.

Priority is usually set by project managers, developers, or business analysts, based on how critical the issue is to the business needs or user experience**.**

**37. What is severity?**

* In software testing, severity refers to the impact or seriousness of a defect on the functionality or performance of the application. It indicates how badly a bug affects the system, regardless of how soon it needs to be fixed.

Severity is usually assigned by testers or QA engineers, based on the technical effect of the defect on the software.

**38. Bug categories are…**

* Bug categories In software testing is **bugs (defects)** can be classified into different **categories** based on their nature, impact, and cause. Categorizing bugs helps testers and developers to understand, prioritize, and resolve issues more effectively.

**Here are the most common bug categories:**

* Functional Bugs
* UI/UX Bugs (User Interface/User Experience)
* Logical Bugs
* Performance bugs
* Security Bugs
* Compatibility Bugs
* Integration Bugs
* Regression Bugs
* Boundary/Validation Bugs

**39. Advantage of Bugzilla .**

* Bugzilla is an open-source, web-based bug tracking system developed by Mozilla. It helps teams track and manage software bugs and issues efficiently.

| **Feature** | **Advantage** |
| --- | --- |
| Cost | Free and open source |
| Accessibility | Web-based, cross-platform |
| Bug Management | Advanced filtering, searching, and tracking |
| Notifications | Real-time email alerts |
| Security | Role-based access control |
| Customization | Custom fields, workflows, and user roles |
| Integration | Works with Jenkins, Git, Test Link, etc. |
| Community | Active support and documentation |

**40. Difference between priority and severity.**

|  |  |  |
| --- | --- | --- |
| Aspect | Severity | Priority |
| Definition | Measures the technical impact of a bug on the system functionality | Defines the urgency of fixing the bug based on business needs |
| Focus | Focuses on the effect of the defect | Focuses on the time or order of fixing |
| Set By | Usually set by testers or QA engineers | Usually set by project managers or product owners |
| Concerned With | The quality and stability of the application | The business or delivery goals |
| Type | Can be Critical, High, Medium, Low | Can be High, Medium, Low |
| Example (High Severity, Low Priority) | Application crashes when you click a rare feature that customers rarely use | Crashing issue but in a feature not used frequently |
| Example (Low Severity, High Priority) | Minor spelling mistake in the company name on homepage | Doesn’t affect function, but must be fixed immediately for brand value |
| Dependency | Based on functionality | Based on business or user impact |
| Fixing Order | Fixed based on how critical it is to system operation | Fixed based on deadlines, client needs, or release planning |

**41. What are the different Methodologies in Agile Development Model ?**

* Agile development consists of several methodologies that follow the principles of the Agile Manifesto. These methodologies emphasize flexibility, collaboration, and customer-centric iterative development. Below is a comparison of different Agile methodologies

|  |  |  |
| --- | --- | --- |
| Methodology | Description | Key Features |
| Scrum | A popular Agile methodology that organizes work into time-boxed sprints. | Sprints (2–4 weeks), Daily Stand-ups, Product Backlog, Sprint Review and Retrospective |
| Kanban | A visual approach to managing work as it moves through a process. | Continuous delivery, Visual workflow (Kanban board), WIP (Work In Progress) limits |
| Extreme Programming (XP) | Focuses on improving software quality and responsiveness to changing requirements. | Pair Programming, Test-Driven Development (TDD), Continuous Integration |
| Lean | Based on Lean manufacturing principles to increase efficiency and reduce waste. | Eliminate waste, Amplify learning, Deliver fast, Respect people |
| Crystal | A family of Agile methodologies tailored to different team sizes and project priorities. | People-centric, Lightweight documentation, Adaptable to team and project size |
| Feature-Driven Development (FDD) | An iterative and incremental software development methodology. | Model-driven, Develop by feature, Frequent, tangible deliverables |
| Dynamic Systems Development Method (DSDM) | A comprehensive Agile project delivery framework. | Active user involvement, Iterative development, Integrated testing |
| Agile Unified Process (AUP) | A simplified version of the Rational Unified Process (RUP) using Agile practices. | Agile modeling, Test-first development, Small releases |

**42. Explain the difference between Authorization and Authentication in Web testing. What are the common problems faced in Web testing ?**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Authentication** | **Authorization** |
| **Definition** | The process of verifying the identity of a user. | The process of verifying what actions or resources a user is permitted to access. |
| **Purpose** | To ensure the user is who they claim to be. | To ensure the user has permission to access the resource. |
| **Process** | Occurs before authorization. | Occurs after authentication. |
| **Data Verified** | Username and password or other credentials | Access level, roles, or privileges. |
| **Example** | Entering login credentials to access an account. | Admin user being allowed to delete records, while regular user is not. |
| **Visibility** | Visible to the use | Not always visible to the user |
| **Failure Result** | If failed, the user cannot log in. | If failed, the user can log in but cannot access certain features or pages. |
| **Focus Area** | User identity verification. | Access rights validation. |
| **Technology Used** | Login forms, multi-factor authentication, biometrics. | Access control lists (ACL), role-based access control (RBAC), policies, and tokens. |

**43. Create Testcases on WhatsApp Group Chat.**

* [**All test case scenario - Google Sheets**](https://docs.google.com/spreadsheets/d/10RhZ0NlJUKXcC9_-5_oEAegaZr43R_QqtRRLc8FFmD0/edit?gid=0#gid=0)

**44. Write a scenario of only WhatsApp chat messages.**

* [**All test case scenario - Google Sheets**](https://docs.google.com/spreadsheets/d/10RhZ0NlJUKXcC9_-5_oEAegaZr43R_QqtRRLc8FFmD0/edit?gid=865295740#gid=865295740)

**45. Write a Scenario of Pen.**

* [**All test case scenario - Google Sheets**](https://docs.google.com/spreadsheets/d/10RhZ0NlJUKXcC9_-5_oEAegaZr43R_QqtRRLc8FFmD0/edit?gid=1302860705#gid=1302860705)

**46. Write a Scenario of Pen Stand.**

* [**All test case scenario - Google Sheets**](https://docs.google.com/spreadsheets/d/10RhZ0NlJUKXcC9_-5_oEAegaZr43R_QqtRRLc8FFmD0/edit?gid=996134999#gid=996134999)

**47. Write a Scenario of Door.**

* [**All test case scenario - Google Sheets**](https://docs.google.com/spreadsheets/d/10RhZ0NlJUKXcC9_-5_oEAegaZr43R_QqtRRLc8FFmD0/edit?gid=1352953138#gid=1352953138)

**48. Write a Scenario of ATM.**

* [**All test case scenario - Google Sheets**](https://docs.google.com/spreadsheets/d/10RhZ0NlJUKXcC9_-5_oEAegaZr43R_QqtRRLc8FFmD0/edit?gid=1563847963#gid=1563847963)

**49. When to used Usability Testing ?**

* Usability Testing is used to evaluate how easy, user-friendly, and intuitive a software application or website is for the end users. It is conducted to ensure that the product meets user expectations in terms of navigation, design, functionality, and overall experience. This type of testing is especially important in identifying design flaws, confusing workflows, or unclear instructions that may frustrate or confuse users. Usability testing is not just about checking if the product works, but whether users can use it effectively and with satisfaction.
* Before **launching the final product:** To ensure that real users can easily navigate and use the application as intended.
* During **the early stages of UI/UX design:** To catch usability problems early when it’s cheaper and easier to fix them.
* When **a new feature or module is added:** To test how well users understand and interact with the new addition.
* To **compare different design versions:** To choose the most user-friendly version between multiple UI/UX options.
* When **feedback indicates poor user experience:** To validate and understand user complaints and fix the issues effectively.
* During **redesign of an existing application:** To ensure the new design actually improves the user experience rather than making it worse.

**50. What is the procedure for GUI Testing ?**

* GUI (Graphical User Interface) Testing is the process of testing a software application's user interface to ensure it meets design specifications and provides a seamless, user-friendly experience. This type of testing focuses on verifying the visual elements of the application—such as buttons, menus, icons, checkboxes, text fields, colors, fonts, and layout—to ensure they appear correctly and function as expected across different devices and screen resolutions.

**51. Write a scenario of Microwave Owen.**

* [**All test case scenario - Google Sheets**](https://docs.google.com/spreadsheets/d/10RhZ0NlJUKXcC9_-5_oEAegaZr43R_QqtRRLc8FFmD0/edit?gid=624332024#gid=624332024)

**52. Write a scenario of Coffee vending Machine**

* [**All test case scenario - Google Sheets**](https://docs.google.com/spreadsheets/d/10RhZ0NlJUKXcC9_-5_oEAegaZr43R_QqtRRLc8FFmD0/edit?gid=2080822684#gid=2080822684)

**53. Write a scenario of chair.**

* [**All test case scenario - Google Sheets**](https://docs.google.com/spreadsheets/d/10RhZ0NlJUKXcC9_-5_oEAegaZr43R_QqtRRLc8FFmD0/edit?gid=869010192#gid=869010192)

**54. Write a Scenario of Wrist Watch**

* [**All test case scenario - Google Sheets**](https://docs.google.com/spreadsheets/d/10RhZ0NlJUKXcC9_-5_oEAegaZr43R_QqtRRLc8FFmD0/edit?gid=614400179#gid=614400179)

**55. Write a Scenario of Lift(Elevator)**

* [**All test case scenario - Google Sheets**](https://docs.google.com/spreadsheets/d/10RhZ0NlJUKXcC9_-5_oEAegaZr43R_QqtRRLc8FFmD0/edit?gid=1742883886#gid=1742883886)

**56. Write a Scenario of WhatsApp payment.**

* [**All test case scenario - Google Sheets**](https://docs.google.com/spreadsheets/d/10RhZ0NlJUKXcC9_-5_oEAegaZr43R_QqtRRLc8FFmD0/edit?gid=384148770#gid=384148770)

**57. Create Test Cases on Compose Mail Functionality.**

* [**https://docs.google.com/spreadsheets/d/1kpyIW1afmuWmJhjPORz8YDqY3B6F4B-Md5zwnXZKFQc/edit?usp=sharing**](https://docs.google.com/spreadsheets/d/1kpyIW1afmuWmJhjPORz8YDqY3B6F4B-Md5zwnXZKFQc/edit?usp=sharing)

**58. Create test cases on Online shopping to buy product (Flipkart)**

* [**https://docs.google.com/spreadsheets/d/1JY7OLSIM4HFzAzfgrUySmmnfNWlbbVPeI6M1hPXTIwo/edit?usp=sharing**](https://docs.google.com/spreadsheets/d/1JY7OLSIM4HFzAzfgrUySmmnfNWlbbVPeI6M1hPXTIwo/edit?usp=sharing)

**59. To create HLR & Test Case of Facebook Login Page.**

* [**https://docs.google.com/spreadsheets/d/1X8X1PB2i5KQBC4u22jQLCL\_dn5Kbj\_ooCkMLvIHNO2k/edit?usp=sharing**](https://docs.google.com/spreadsheets/d/1X8X1PB2i5KQBC4u22jQLCL_dn5Kbj_ooCkMLvIHNO2k/edit?usp=sharing)