Que (1) what is Exploratory Testing?

- Exploratory Testing is one of the type of Experience Based Testing
- Exploratory testing is a powerful approach within manual testing where testers actively investigate the software to uncover potential issues, rather than strictly following pre-written test cases
- It's often described as **simultaneous learning, test design, and execution**
- Rather than following a script, testers use their intuition, experience, and creativity to:
- Discover bugs
- Understand the application
- Find usability issues

☑ Here's a breakdown of what that means in the context of manual testing:

- **Learning:** As a manual tester performs exploratory testing, they continuously learn about the software's functionality, features, and potential weaknesses. This understanding guides their subsequent testing.
- **Test Design:** Based on their learning and intuition, the tester designs tests on the fly. There's no formal test case document to refer to; the test ideas emerge during the exploration.
- **Execution:** The tester immediately executes the tests they've just conceived, observing the software's behaviour and looking for deviations from expected outcomes or potential problems.

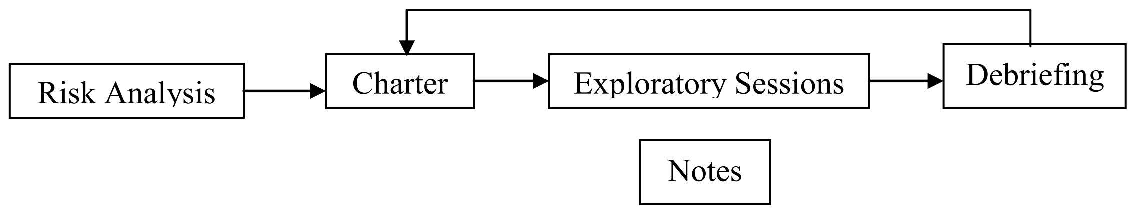
☑ **Key Characteristics of Exploratory Testing in Manual Testing:**

- **Tester as the Primary Tool:** The tester's knowledge, experience, creativity, and critical thinking skills are paramount. They decide what to test and how to test it based on their understanding of the application.
- **Flexibility and Adaptability:** Manual exploratory testing allows for quick adaptation to changes in the software or new information discovered during testing. Testers can easily shift their focus and explore unexpected areas.
- **Emphasis on Discovery:** The primary goal is to discover bugs and usability issues that might be missed by scripted tests, especially in complex or poorly documented areas.
- **Lightweight Documentation:** While not entirely undocumented, the focus is more on the testing activity itself rather than extensive upfront documentation. Testers often document their findings, test ideas, and the flow of their exploration as they go. Techniques like session-based test management (SBTM) can add structure to this documentation.
- **User-Centric Approach:** Exploratory testers often try to think and act like end-users, exploring the application from their perspective to identify potential usability problems.

☑ **Benefits of Exploratory Testing in Manual Testing:**

- **Finds Hidden and Critical Defects:** Exploratory testing can uncover unexpected bugs and edge cases that structured testing might miss.
- **Improves Test Coverage:** By exploring various scenarios and user flows, testers can often achieve broader test coverage.
- **Provides Faster Feedback:** Issues can be identified and reported quickly as testing and analysis happen simultaneously.
- **Enhances Tester Skills and Engagement:** It encourages testers to think critically, learn about the product deeply, and be more engaged in the testing process.
- **Adaptable to Changing Requirements:** It's particularly useful when requirements are unclear, incomplete, or changing rapidly.

- **Complements Scripted Testing:** Exploratory testing can identify areas where more formal, scripted tests might be needed.
- More structured than Error guessing



Que (2) what is traceability matrix?

- To protect against changes you should be able to trace back from every system component to the original requirement that caused its presence.

☑ **Core Purpose for Traceability matrix are following**

- Ensure all requirements are covered by manual test cases
- Verify that manual testing efforts are aligned with the intended functionality.
- Facilitate the tracking of defects found during manual execution back to their originating requirements.
- Help manage the impact of requirement changes on manual test efforts.

☑ **Types of Traceability matrix**

- Forward Traceability – Mapping of Requirements to Test cases
- Backward Traceability – Mapping of Test Cases to Requirements
- Bi-Directional Traceability - A Good Traceability matrix is the References from test cases to basis documentation and vice versa.

☑ **Pros of Traceability Matrix**

- Make obvious to the client that the software is being developed as per the requirements.
- To make sure that all requirements included in the test cases
- To make sure that developers are not creating features that no one has requested
- Easy to identify the missing functionalities
- If there is a change request for a requirement, then we can easily find out which test cases need to update.
- The completed system may have "Extra" functionality that may have not been specified in the design specification, resulting in wastage of manpower, time and effort.

☑ **Cons of Traceability Matrix**

- No traceability or Incomplete Traceability Results into:

✓ Poor or unknown test coverage, more defects found in production
✓ It will lead to miss some bugs in earlier test cycles which may arise in later test cycles. Then a lot of discussions arguments with other teams and managers before release.
✓ Difficult project planning and tracking, misunderstandings between different teams over project dependencies, delays, etc.

Que (3) what is Boundary value testing?

- Boundary Value Analysis is called as B.V.A.
- Boundary value analysis is a methodology for designing test cases that concentrates software testing effort on cases near the limits of valid ranges.

- Boundary Value Testing is a software testing technique focused on testing the edges or boundaries of input domains.
- The idea is that errors often occur at or near the limits of the allowed values for input variables.
- Instead of testing a wide range of typical values, boundary value testing concentrates on these critical points.
- Boundary value analysis is a method which refines equivalence partitioning.
- Boundary value analysis generates test cases that highlight errors better than equivalence partitioning
- The trick is to concentrate software testing efforts at the extreme ends of the equivalence classes
- At those points when input values change from valid to invalid errors are most likely to occur.
- Boundary Value Analysis (BVA) uses the same analysis of partitions as EP and is usually used in conjunction with EP in test case design.

☑ **B.V.A. Work**
- For each input variable, boundary value testing typically considers the following values:
- **Minimum (min):** The lowest allowed value.
- **Maximum (max):** The highest allowed value.
- **Nominal (nom):** A valid value within the range (often the minimum + 1 or some other representative value).
- **Minimum - 1 (min-1):** A value just below the minimum (if applicable and makes sense for the data type). This tests the lower out-of-bounds condition.
- **Maximum + 1 (max+1):** A value just above the maximum (if applicable and makes sense for the data type). This tests the upper out-of-bounds condition.

☑ **Advantages of Boundary Value Testing:**
- Relatively simple to understand and apply.
- Effective in finding errors related to boundary conditions.
- Generates a manageable number of test cases compared to exhaustive testing.
- Provides good test coverage at the edges of input domains.

☑ **Disadvantages of Boundary Value Testing:**
- May not detect errors that occur in the middle of the input range or due to combinations of input values.
- Assumes that each input variable is independent, which may not always be the case.
- The "min-1" and "max+1" values might not be applicable or meaningful for all data types.

Que (4) what is Error, Defect, Bug and failure?

- Error : A **mistake** made by a developer or human during coding, design, or requirement gathering
- Defect: A **flaw or issue** in the software that results from an error. It exists in the code or design.
- Bug: **Informal term** for a defect. When a defect is found during testing or in production, it's typically referred to as a **bug**.

- Failure: When the software **does not perform as expected** during execution because of a defect.
- So we can say tha

$$\begin{tikzcd}\textbf{Error} \arrow[r] & \textbf{Defect} \arrow[r] & \textbf{Bug} \arrow[r] & \textbf{Failure} \\ \textbf{(By Developer)} & \textbf{(In code)} & \textbf{(Find in test)} & \textbf{(By User/Tester)}\end{tikzcd}$$

Que (5) what is 7 key principles? Explain in detail?

- The 7 key principles of manual testing provide a solid framework for effective and insightful testing. Here they are:

1. **Testing shows presence of defects:** This principle emphasizes that testing can reveal defects in the software, but it cannot guarantee that the software is entirely defect-free. Even after extensive testing, some hidden bugs might still exist.
2. **Exhaustive testing is impossible:** Testing every single possible input combination, execution path, and environmental variation is practically infeasible, especially for complex software. The number of possibilities often becomes astronomically large. Therefore, test efforts need to be prioritized and focused on the most critical areas and likely defect-prone functionalities.
3. **Early testing:** To catch defects early in the software development life cycle (SDLC) is significantly more cost-effective than fixing them later. The earlier a defect is found, the less impact it has on other parts of the system and the easier and cheaper it is to rectify.
4. **Defect clustering:** Experience shows that defects tend to cluster in certain areas of the software. A small number of modules or functionalities often contain the majority of the defects. By identifying these high-risk areas based on past experience, complexity, or changes, testing efforts can be focused more effectively.
5. **The pesticide paradox:** If the same set of test cases is run repeatedly over time, eventually, these test cases will no longer find any new defects. Just like pesticides become ineffective against pests that develop resistance, test cases become less effective at uncovering new bugs if they aren't updated or diversified. Regular review and revision of test cases are crucial.
6. **Testing is context dependent:** Testing is not a one-size-fits-all activity. The approach, techniques, and types of testing performed will vary depending on the context of the software being tested. Factors like the application domain (e.g., banking, gaming, and e-commerce), risk level, and regulatory requirements will influence the testing strategy.
7. **Absence-of-errors fallacy:** Finding and fixing a large number of defects doesn't necessarily mean that the software is usable or meets the users' needs. It's possible to build a system that is technically sound but doesn't fulfil the intended business objectives or user expectations. Therefore, testing should also focus on validating requirements and user satisfaction.

- These principles serve as guiding lights for manual testers, helping them plan, execute, and evaluate their testing activities more effectively. They highlight the limitations and realities of testing while emphasizing strategies for maximizing its value.

Que (6) Difference between QA v/s QC v/s Tester

QA (Quality Assurance)	QC (Quality control)	Tester
• Activities which ensure the implementation of processes, procedures and standards in context to verification of developed	• Activities which ensure the verification of developed software with respect to documented (or not in some cases)	• Activities which ensure the identification of bugs/error/defects in the Software

software and intended requirements.	requirements.	
• 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.	• Focuses on actual testing
• Process oriented activities	• Product oriented activities	• Product oriented activities.
• Preventive activities	• It is a corrective process	• It is a preventive process
• It is a subset of Software Test Life Cycle (STLC)	• QC can be considered as the subset of Quality Assurance.	• Testing is the subset of Quality Control.

Que (7) Difference between Smoke and Sanity?

☑ **Smoke Testing (Build Verification Testing)**

- Smoke Testing is performed after software build to ascertain that the critical functionalities of the program is working fine.
- It is executed "before" any detailed functional or regressions tests are executed on the software build means to quickly verify the stability of a new software build.
- The purpose is to reject a badly broken application, so that the QA team does not waste time installing and testing the software application.
- In Smoke Testing, the test cases chosen cover the most important functionality or component of the system.
- The objective is not to perform exhaustive testing, but to verify that the critical functionalities of the system are working fine.

☑ **Sanity Testing (Surface-Level Testing)**

- After receiving a software build, with minor changes in code, or functionality, Sanity testing is performed to ascertain that the bugs have been fixed and no further issues are introduced due to these changes.
- The goal is to determine that the proposed functionality works roughly as expected
- If sanity test fails, the build is rejected to save the time and costs involved in a more rigorous testing
- The objective is "not" to verify thoroughly the new functionality, but to determine that the developer has applied some rationality (sanity) while producing the software.

Smoke Testing	Sanity Testing
• 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

• The objective of this testing is to verify "stability" of the system in order to with more rigorous testing	• The objective of the testing is to verify the "rationality" of the system in order proceed to proceed with more rigorous testing
• Performed by Developer	• Performed by Taster
• Usually Documented or Scripted	• Usually not Documented or Unscripted
• subset of Regression testing	• subset of Acceptance testing
• exercises the entire system from end to end	• exercises only the particular component of the entire system
• is like General Health Check up	• is like Specialized Health Check up

Que (8) Difference between verification and Validation.

- Verification and validation are two distinct processes in quality assurance.
- **Verification focuses on ensuring the product meets its specifications**, often through static analysis and reviews, while **validation focuses on ensuring the product meets the needs and expectations** of the end-user, typically through dynamic testing.

Verification	Validation
• As per Definition of ISTQB, confirmation by examination and through the provision of objective evidence that specified requirements have been fulfilled.	• As per Definition of ISTQB, confirmation by examination and through the provision of objective evidence that the requirements for a specific intended use or application have been fulfilled.
• It includes checking documents, designs, codes, and programs.	• It includes testing and validating the actual product
• Verification is the **Static testing**.	• Validation is **dynamic testing.**
• It does ***not* include the execution of the code**.	• It **includes the execution of the code**.
• Methods used in verification are **reviews, walkthroughs, inspections and desk-checking**.	• Methods used in validation are **Black Box Testing**, **White Box Testing** and **Non-Functional testing**.
• It checks whether the software conforms to specifications or not.	• It checks whether the software meets the requirements and expectations of a customer or not.
• It can find the **Bugs in the early stage of the development**.	• It can only **find the bugs that could not**

	be found by the verification process.
• The goal of verification is application and software architecture and specification.	• The goal of validation is an actual product
• Verification is typically performed by the quality assurance (QA) team, focusing on reviewing documents, designs, and code to ensure compliance with specified requirements.	• Validation is performed by the testing team, which executes the software in real environments to ensure it meets user expectations and requirements
• It comes before validation.	• It comes after verification
• It consists of checking of documents/files and is performed by human	• It consists of execution of program and is performed by computer.
• After a valid and complete specification the verification starts	• Validation begins as soon as project starts.
• Verification is for prevention of errors.	• Validation is for detection of errors.

Que (9) Explain types of Performance testing.

- Performance testing in software testing encompasses various types designed to evaluate how a system behaves under different loads and conditions.
- Common types include load testing, stress testing, scalability testing, endurance testing, and spike testing, each focusing on a specific aspect of performance.
- Types of Performance Testing are following

(1) Spike testing	(4) Load testing
(2) Volume testing	(5) Stress testing
(3) Scalability testing	(6) Endurance testing

(1) Load Testing: This assesses the system's behaviour under expected user loads, monitoring how it handles various levels of traffic and transactions.
(2) Stress Testing: This evaluates the system's performance under extreme conditions, pushing it beyond its normal operating point to identify potential failure points and assess its stability.
(3) Endurance Testing: This verifies the system's reliability and stability over extended periods of use, helping to identify issues like memory leaks or degradation over time.
(4) Spike Testing: This analyzes the system's response to sudden and significant increases in traffic, simulating scenarios like flash sales or high-traffic events.
(5) Volume Testing: This focuses on the system's ability to handle large amounts of data, ensuring it can process and store data efficiently.
(6) Scalability Testing: This examines the system's ability to handle increasing or decreasing user demands, ensuring it can scale up or down effectively as needed.

Que (10) what is stress Testing?

- Stress testing in software testing is a specialized type of performance testing that evaluates how a system behaves under extreme or abnormal conditions, pushing it beyond its normal capacity to identify vulnerabilities and performance bottlenecks.
- The goal is to determine the system's breaking point and ensure it can handle unexpected loads without failing.

☑ **Key aspects of stress testing:**

- Testing Beyond Normal Limits:

Stress testing goes beyond normal usage by simulating heavy loads, high user traffic, or resource constraints to see how the system responds.

- Identifying Vulnerabilities:

By pushing the system to its limits, stress testing helps uncover weaknesses and potential failure points that might not be apparent under regular usage.

- Ensuring Robustness:

Stress testing verifies the system's resilience and stability under extreme conditions, ensuring it can withstand unexpected pressures.

- Measuring System Performance:

Stress tests measure how the system performs under pressure, including response times, resource usage, and error handling.

- Finding the Breaking Point:

The primary objective is to determine the system's breaking point, the point at which it fails or experiences significant performance degradation.

☑ **Types of Stress Testing:**

- Distributed Stress Testing:

Tests the application's ability to handle workloads distributed across multiple systems or locations.

- Burst Testing:

Simulates rapid, unpredictable spikes in usage, like during flash sales or high-demand events.

- Endurance Testing (Soak Testing):

Evaluates the system's ability to handle continuous usage over a long period.

- Systemic Stress Testing:

Tests multiple systems running on the same server, identifying defects that can block other applications.

☑ **Importance of Stress Testing:**

- Ensuring System Stability:

By identifying potential problems under stress, stress testing helps ensure the system remains stable and reliable under various conditions.

- Improving Performance:

Stress tests can reveal bottlenecks and areas for optimization, leading to improved system performance and efficiency.

- Preventing Failures:

By identifying vulnerabilities, stress testing can help prevent system failures or critical issues from occurring in production.

- Enhancing User Experience:

By ensuring the system can handle heavy loads and maintain stability, stress testing contributes to a better user experience.

Que (11) what is load testing?

- Load testing in software testing is a performance test that simulates real-world usage by subjecting a system to expected or peak user loads.
- It helps identify performance bottlenecks, response delays, and stability issues, ensuring the application can handle anticipated traffic without performance degradation.

☑ **Purpose**

- Load testing aims to determine how a system behaves under normal and high-load conditions.
- It helps identify the maximum operating capacity of the application and any potential issues that might arise under stress.

☑ **Simulating Real-World Usage**

- Load testing mimics the way users typically interact with a system, including the number of concurrent users, transaction types, and timing patterns

☑ **Identifying Performance Issues**

- By subjecting the system to simulated loads, testers can uncover problems such as slow response times, crashes, memory leaks, and other issues that might not be apparent under light load conditions.

☑ **Benefits**

- Load testing helps ensure the application can handle expected workloads, maintain stability, and provide a good user experience. It also helps prevent costly downtime and reduces the risk of user frustration.

☑ **Tools**

- Various tools are available to automate load testing, such as JMeter, LoadRunner, and Grafana Cloud k6.

Que (12) what is Adhoc testing?

- Adhoc testing is an informal testing type with an aim to break the system
- This testing is primarily performed if the knowledge of testers in the system under test is very high
- Main aim of this testing is to find defects by random checking.
- Adhoc testing can be achieved with the testing technique called Error Guessing.
- The Error guessing is a technique where the experienced and good testers are encouraged to think of situations in which the software may not be able to cope.
- Ad-hoc testing, also known as random or monkey testing is an informal software testing technique where testers don't follow a pre-defined plan or test cases.
- Instead, they use their intuition, experience, and knowledge of the software to explore different scenarios and uncover potential defects.
- This method is often used to identify issues that might be missed during formal testing.

- It also saves a lot of time because of the assumptions and guessing made by the experienced testers to find out the defects which otherwise won't be able to find.

☑ **Key Characteristics of Ad-hoc Testing**

- Unstructured: It doesn't rely on a structured testing process, test plans, or documented test cases.
- Informal: It's a more relaxed approach compared to formal testing methods.
- Intuitive: Testers use their knowledge and experience to identify potential issues.
- Random/Monkey Testing: The term "monkey testing" highlights the random and unpredictable nature of this testing approach.
- Uncovers hidden bugs: Ad-hoc testing can reveal defects that might be overlooked by formal testing methods.
- Used after formal testing: It's often conducted after formal testing to catch any remaining issues or loopholes.
- No documentation: Ad-hoc testing doesn't require documentation of test cases or procedures.

☑ **Types of Ad-hoc Testing**

(1) Buddy Testing (2) Pair testing (3) Monkey Testing

(1) Buddy Testing

- Two buddies mutually work on identifying defects in the same module. Mostly one buddy will be from development team and another person will be from testing team. Buddy testing helps the testers develop better test cases and development team can also make design changes early. This testing usually happens after unit testing completion

(2) Pair testing

- Two testers are assigned modules, share ideas and work on the same machines to find defects. One person can execute the tests and another person can take notes on the findings. Roles of the persons can be a tester and scriber during testing

(3) Monkey Testing

- Randomly test the product or application without test cases with a goal to break the system.

☑ **Benefits of Ad-hoc Testing**

- Discovering unexpected defects:

It can reveal bugs that wouldn't have been found through structured testing.

- Exploration of different scenarios:

Testers can explore various aspects of the software and identify potential weaknesses.

- Early detection of issues:

It can be used early in the development cycle to identify problems before they become major issues.

- Efficient use of time

When time is limited, ad-hoc testing can provide a quick way to identify defects.

☑ **Drawbacks of Ad-hoc Testing:**

- Difficult to reproduce defects:

Since there are no documented steps, it can be challenging to reproduce and fix defects.

- Requires experienced testers:

It relies heavily on the tester's knowledge and experience, so it's not suitable for inexperienced testers.

- Not suitable for all projects:

Ad-hoc testing is not appropriate for projects with strict requirements or formal testing processes.

In essence, ad-hoc testing is a valuable technique for uncovering hidden defects and exploring software behaviour informally, but it should be used strategically and with the appropriate expertise.

Que (13) what is GUI Testing?

- GUI (Graphical User Interface) testing is a type of software testing that focuses on verifying the visual elements and functionality of a software application's user interface.
- It ensures that the GUI behaves as expected, adheres to design specifications, and provides a positive user experience.

☑ **Approach of GUI Testing**

- MANUAL BASED TESTING
  Under this approach, graphical screens are checked manually by testers in conformance with the requirements stated in business requirements document
- RECORD AND REPLAY
  GUI testing can be done using automation tools. This is done in 2 parts. During Record, test steps are captured into the automation tool. During playback, the recorded test steps are executed on the Application under Test. Example of such tools – QTP
- MODEL BASED TESTING
  A model is a graphical description of system's behaviour. It helps us to understand and predict the system behaviour. Models help in a generation of efficient test cases using the system requirements
- Build the model
- Determine Inputs for the model
- Calculate expected output for the model
- Run the Tests
- Compare the actual output with the expected output
- Decision on further action on the model
- Model Based Testing is an evolving technique for the generating the test cases from the requirements.
- Its main advantage, compared to above two methods, is that it can determine undesirable states that your GUI can attain
- Some of the modelling techniques from which test cases can be derived:

(I) Charts – Depicts the state of a system and checks the state after some input.

(II) Decision Tables – Tables used to determine results for each input applied

Que (14) what is Non-Functional Testing?

- Testing the attributes of a component or system that do not relate to functionality, e.g. reliability, efficiency, usability, interoperability, maintainability and portability.
- Non-functional testing focuses on evaluating how well a software system performs under various conditions, such as its speed, stability, scalability, and security, rather than its specific features.
- It ensures that the system meets quality attributes beyond just its functionality.

- Non-functional testing is a type of software testing that verifies aspects of the software that are not directly related to its functionality.
- It assesses how well the system performs, how it handles load, how secure it is, and how usable it is.
- It's about ensuring the software meets quality standards beyond just its features.

☑ **Key areas covered are following**

- Performance: Testing the system's speed, responsiveness, and ability to handle various loads.
- Scalability: Evaluating how well the system can handle increasing workloads and demands.
- Security: Identifying vulnerabilities and ensuring the system is protected against threats.
- Usability: Assessing how easy it is for users to interact with and use the software.
- Reliability: Determining how consistently and reliably the system functions.

☑ **Non-Functional Testing are important for following reason**

- Non-functional testing is crucial for ensuring the software meets real-world performance and usability requirements.
- It helps identify potential problems early in the development process, which can be more cost-effective than fixing issues later.
- It contributes to a better overall user experience and higher quality software

☑ **Types of non-Functional testing types**

- Performance Testing : Evaluating the application's speed, responsiveness, and scalability under different load scenarios
- Load Testing: Simulating a high volume of users to assess the system's ability to handle peak loads.
- Volume Testing :
- Stress Testing: Pushing the system to its limits to identify potential weaknesses and failure points.
- Security Testing: Assessing the system's vulnerability to security threats and ensuring proper security measures are in place.
- Installation Testing: Upgrade testing and installation testing verify that software will work properly on everyone's machines. So, upgrade testing is done for existing users. And installation testing is done for new users.
- Penetration Testing : Penetration testing aims to identify and exploit vulnerabilities before a real attack can occur, allowing developers and security teams to fix them
- Compatibility Testing: Compatibility testing in software testing is a crucial process that verifies if an application functions correctly across various environments, including different devices, browsers, operating systems, and network configurations.
- Migration Testing: Ensuring the software functions correctly in different languages, regions, and time zones.

Que (15) what is functional system testing?

- Testing based on an analysis of the specification of the functionality of a component or system.
- Functional testing helps identify and fix issues early in the development lifecycle, ensuring the system is reliable and meets user expectations.

- Functional system testing is a type of software testing that verifies whether a system functions correctly according to its specified requirements and meets the intended business needs.
- It's a black-box testing method, meaning the internal code structure is not considered during the testing process. Functional testing focuses on the outputs and results of processing, ensuring each feature and function works as expected.

☑ **Key Aspects of Functional System Testing**

☑ **Requirement-Based:**

Functional testing validates the system against the functional specifications and requirements outlined in the documentation.

☑ **Black-Box Testing:**

It focuses on the external behaviour of the system, testing inputs and outputs without looking at the internal code.

☑ **Focus on Functionality:**

It ensures that the system performs its intended tasks correctly, meeting the needs of users and stakeholders.

☑ **Not Concerned with Internal Structure:**

Functional testing doesn't investigate the quality, security, or performance of the application's underlying source code.

☑ **Types of Functional Testing:**

There are various types, including:

- **Unit Testing:** Testing individual components or modules in isolation.
- **Integration Testing:** Testing the interaction between different modules or components.
- **User Acceptance Testing (UAT):** Testing the system by end-users to ensure it meets their needs.

Que (16) Difference between Priority and Severity

- In software testing, priority determines the order in which bugs should be fixed, while severity assesses the impact of a bug on the software's functionality.
- Priority considers factors like business impact and project timelines, while severity is primarily concerned with the technical impact of a bug on the user or system.

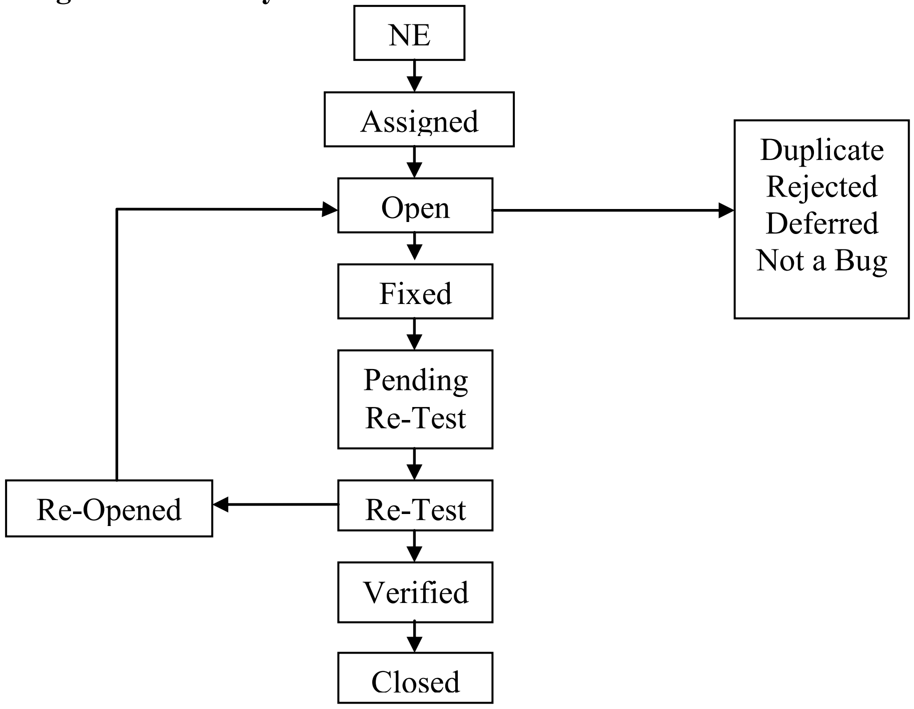
Priority	Severity
• **Definition**: Priority determines how urgently a bug needs to be addressed, considering factors like business impact, project deadlines, and release timelines.	• **Definition**: Severity refers to the degree of harm or impact a defect has on the software's functionality and user experience.
• **Focus**: Priority is more about the scheduling of bug fixes and how quickly they need to be resolved, considering the overall project goals.	• **Focus**: Severity is primarily concerned with the technical impact of a bug and how it affects the user or system
• **Example**: A bug with a high severity might have a low priority if it's not critical for the current release and a workaround is available.	• **Example:** A critical bug that makes a website inaccessible would have high severity, while a minor typo might have

	low severity.
• **Contextual:** Priority can be influenced by various factors, including the severity of the bug, its impact on the user, and its potential effect on the business.	• **Relativity:** Severity can be assessed relatively, meaning that a bug with a high severity in one context might have a lower severity in another.

Que (17) what is Bug Life Cycle?

- "A computer bug is an error, flaw, mistake, failure, or fault in a computer program that prevents it from working correctly or produces an incorrect result. Bugs arise from mistakes and errors, made by people, in either a program's source code or its design."
- The duration or time span between the first time defects is found and the time that it is closed successfully, rejected, postponed or deferred is called as 'Defect Life Cycle'.
- When a bug is discovered, it goes through several states and eventually reaches one of the terminal states, where it becomes inactive and closed.
- The process by which the defect moves through the life cycle is shown as following

☑ **Bug/Defect Life Cycle**



- As you can see from above diagram, a defect's state can be divided into Open or Closed.
- When a bug reaches one of the Closed or Terminal states, its lifecycle ends. Each state has one or more valid states to move to.
- This is to ensure that all necessary steps are taken to resolve or investigate that defect. For example, a bug should not move from Submitted state to resolved state without having it open.
- In a typical scenario, as soon as a bug is identified, it is logged into the bug tracking system with status as Submitted. After ascertaining the validity of the defect, it is given the "Open" Status.

☑ **Key Stages in the Bug Life Cycle:**

1. **New**: A bug is first reported and logged into the bug tracking system.

2. **Assigned**: The bug is assigned to a developer or team for investigation and fixing.

3. **Open**: The developer starts working on the bug, often involving code changes and fixes.

4. **Fixed**: The developer confirms that the bug has been resolved and the code is updated.

5. **Pending Retest**: The bug is awaiting retesting by a tester to verify the fix.

6. **Retest**: The tester verifies that the bug is truly fixed, and the application behaves as expected.

7. **Verified:** If the retesting is successful, the bug is marked as verified, indicating it's fully resolved.

8. **Reopen**: If the bug reappears during retesting, its status is reopened, and it's sent back to the developer for further investigation and fixing.

9. **Closed:** Once the bug is confirmed as fixed and no longer exists, it's marked as closed.

Que (18) Explain the difference between Functional testing and Non-Functional testing

Functional testing	Non-functional testing
• performed using the functional specification provided by the client & verifies the system against the functional requirements	• Checks the Performance, reliability, scalability and other non-functional aspects of the software system.
• Functional testing is executed first	• Non functional testing should be performed after functional testing
• Manual testing or automation tools can be used for functional testing	• Using tools will be elective for this testing
• Business requirements are the inputs to functional testing	• Performance parameters like speed, scalability are inputs to non-functional testing.
• Functional testing describes what the product does	• Non-functional testing describes how good the product works
• Easy to do manual testing	• Tough to do manual testing
• Types of Functional testing are (i) Unit Testing (ii) Smoke Testing (iii) Sanity Testing (iv) Integration Testing (v) White box testing (vi) Black Box testing (vii) User Acceptance testing (UAT) (viii) Regression Testing	• Types of Non-Functional testing are (ix) Performance Testing (x) Load Testing (xi) Volume Testing (xii) Stress Testing (xiii) Security Testing (xiv) Installation Testing (xv) Penetration Testing (xvi) Compatibility Testing (xvii) Migration Testing

Que (19) when should "Regression Testing" be performed?

- Regression testing should be performed whenever there are changes to the software, whether it's adding new features, fixing bugs, or making other code modifications. It's crucial to ensure that these changes haven't introduced any unintended side effects or broken existing functionalities.

☑ **Elaboration:**

- **After Code Changes:**

Whenever new features are added, existing features are enhanced, or code is refactored, regression testing validates that core functionalities remain intact.

- **Post Bug Fixes:**

After fixing bugs, regression testing helps ensure that the fixes haven't introduced new issues in other areas of the software.

- **During Integration:**

When integrating new modules or components, regression testing verifies that the integration hasn't caused any unexpected behaviours.

- **Pre-release Releases:**

Before releasing new versions to production, regression testing ensures the software is stable and reliable.

- **After Environment Changes:**

Changes to the environment (e.g., hardware upgrades, software updates) can also necessitate regression testing to ensure the software still functions as expected.

- **Periodic Maintenance:**

Even without recent code changes, periodic regression testing helps identify hidden regressions that may have emerged due to system updates or infrastructure changes, according to Lambda Test.

- **After Unit, Integration, and System Testing:**

Regression testing often follows and verifies the success of unit, integration, and system testing, according to Cyber Panel.

Que (20) what is Equivalence partitioning testing?

- Equivalence partitioning is a black-box testing technique that divides input data into groups (partitions) based on similar behaviour, reducing the need to test every single input. It assumes that the system will handle all inputs within a partition the same way, so testing a single representative input from each partition is sufficient. This method minimizes redundancy and improves test efficiency while ensuring thorough coverage.
- Aim is to treat groups of inputs as equivalent and to select one representative input to test them all

☑ EP can be used for all Levels of Testing

- Equivalence partitioning is the process of defining the optimum number of tests by:

(i) Reviewing documents such as the Functional Design Specification and Detailed Design Specification, and identifying each input condition within a function,

(ii) Selecting input data that is representative of all other data that would likely invoke the same process for that particular condition.

☑ Here's a more detailed explanation:

1. Identifying Input Ranges: Determine the valid and invalid ranges for the input data.
2. Partitioning: Divide the input range into logical partitions where each partition represents a set of inputs that the system should treat similarly.
3. Defining Representative Test Cases: Choose representative values from each partition for testing.
4. Boundary Testing: Test at the boundaries of the partitions to ensure the system handles edge cases correctly.
5. Invalid Case Testing: Test invalid input values that fall outside the valid partitions to verify error handling.

Example:
Consider a login field that accepts usernames and passwords.

- Valid Input: Usernames and passwords within the allowed length and character sets.
- Invalid Input: Empty usernames or passwords, usernames exceeding the maximum length, or invalid characters.

Que (21) what is Integration testing?

- Integration testing in software testing is the process of combining and testing individual modules or components of a software application to verify how they interact and work together. It follows unit testing, which focuses on individual units of code, and precedes system testing, which tests the entire application.

☑ **Purpose of Integration Testing:**

- **Ensuring modules interact correctly:** Integration testing verifies that modules communicate effectively, share data, and function as a cohesive unit.
- **Identifying interface defects:** It helps uncover issues related to data flows, formatting, communication, and exception handling between modules.
- **Checking hardware compatibility:** Integration testing can also verify if modules are compatible with the intended hardware environment.
- **Maintaining code uniformity:** It ensures that different modules adhere to a consistent coding style and interface standards.
- **Improving software quality:** By verifying the overall functionality of the combined modules, integration testing contributes to higher software quality and reliability.
- **Facilitating Continuous Integration:** In Agile environments, integration tests are run regularly to ensure new code changes don't break existing functionality, says Browser Stack.

☑ **Types of Integration Testing:**

- **Top-down:** This approach starts with the main module and integrates it with lower-level modules gradually, simulating lower-level components with stubs.
- **Bottom-up:** This approach begins with the lowest-level modules and integrates them upwards, eventually reaching the top-level module.
- **Big Bang:** All modules are integrated and tested at once, as a single unit, which is suitable for small and simple systems.
- **Sandwich (Hybrid):** This approach combines both top-down and bottom-up testing to integrate modules from both directions.

☑ **Benefits of Integration Testing:**

- **Early bug detection:** Integration testing helps identify defects early in the development process, reducing the cost of fixing them later.
- **Improved code quality:** It ensures that the code is well-designed and that modules can interact effectively.
- **Reduced risk of errors:** By verifying that modules work together, integration testing minimizes the risk of errors during system integration.
- **Enhanced user experience:** When modules work seamlessly together, it leads to a better overall user experience.

Que (22) what determines the level of risk?

- A properly designed test that passes, reduces the overall level of Risk in a system

- Risk – 'A factor that could result in future negative consequences; usually expressed as impact and likelihood'
- When testing does find defects, the Quality of the software system increases when those defects are fixed
- The Quality of systems can be improved through Lessons learned from previous projects
- Analysis of root causes of defects found in other projects can lead to Process Improvement
- Process Improvement can prevent those defects reoccurring
- Which in turn, can improve the Quality of future systems
- Testing should be integrated as one of the Quality assurance activities

☑ **Elaboration**

- **Likelihood**: This refers to the probability of a particular risk manifesting. It can be expressed as a percentage or on a scale (e.g., 1-5 or 1-10)
- **Impact**: This refers to the consequences of a risk if it were to occur. It can be measured by the severity of the issue, the number of users affected, or the cost of fixing it. Impact is also often classified as low, medium, or high, or on a scale.
- **Prioritization**: By assessing the likelihood and impact of various risks, testers can prioritize which issues to address first. Risks with high likelihood and high impact should be addressed early on.
- **Risk-Based Testing**: This approach focuses testing efforts on the areas with the highest risk.
- **Risk Mitigation:** Understanding the level of risk is crucial for developing effective risk mitigation strategies.

Que (23) what is Alpha testing?

- It is always performed by the developers at the software development site.
- Sometimes it is also performed by Independent Testing Team.
- Alpha Testing is not open to the market and public
- It is conducted for the software application and project.
- It is always performed in Virtual Environment.
- It is always performed within the organization.
- It is the form of Acceptance Testing.
- Alpha Testing is definitely performed and carried out at the developing organizations location with the involvement of developers.
- It comes under the category of both White Box Testing and Black Box Testing.

☑ **Key aspects of Alpha Testing:**

- Internal Testing: Alpha testing is performed by the development team, QA engineers, and sometimes internal stakeholders.
- Controlled Environment: It's typically conducted in a test lab or controlled environment where fixes can be made quickly.
- Focus on Functionality and Performance: Alpha testing aims to uncover and address issues with the software's functionality, usability, performance, and stability.
- Pre-Beta Testing: It precedes beta testing, where a limited group of users outside the development team tests the software.
- Identifying Critical Issues: The primary goal is to find and fix "showstoppers" and other major bugs before the software is exposed to a broader audience.

- White Box Testing: In a white box setting, testers can examine the internal workings of the software, providing additional insights into potential issues.
- Iterative Process: Bugs and issues identified during alpha testing are typically addressed immediately by the development team, and the software is retested in the test environment.

Que (24) what is beta testing?

- It is always performed by the customers at their own site.
- It is not performed by Independent Testing Team.
- Beta Testing is always open to the market and public.
- It is usually conducted for software product.
- It is performed in Real Time Environment.
- It is always performed outside the organization.
- It is also the form of Acceptance Testing.
- Beta Testing (field testing) is performed and carried out by users or you can say people at their own locations and site using customer data.
- It is only a kind of Black Box Testing.

Que (25) what is component testing?

- Component testing, also called module (unit) testing, focuses on testing specific units of an application—such as functions, classes, or modules—without the rest of the system. This method ensures that each component works properly per design specifications by focusing on particular components.
- Component (Unit) – A minimal software item that can be tested in isolation. It means "A unit is the smallest testable part of software."
- Component Testing – The testing of individual software components.
- Unit Testing is a level of the software testing process where individual units/components of a software/system are tested. The purpose is to validate that each unit of the software performs as designed.
- Unit testing is the first level of testing and is performed prior to Integration Testing.
- Sometimes known as Unit Testing, Module Testing or Program Testing
- Component can be tested in isolation – stubs/drivers may be employed
- Unit testing frameworks, drivers, stubs and mock or fake objects are used to assist in unit testing.
- Functional and Non-Functional testing
- Unit tests are typically written and run by software developers to ensure that code meets its design and behaves as intended with debugging tool.
- A unit is the smallest testable part of an application like functions/procedures, classes, interfaces.
- The goal of unit testing is to isolate each part of the program and show that the individual parts are correct.
- A unit test provides a strict, written contract that the piece of code must satisfy. As a result, it affords several benefits.
- Unit tests find problems early in the development cycle.
- Unit testing is performed by using the White Box Testing method.

Que (26) Mention what are the categories of defects?

- Software defects, also known as bugs or faults, can be broadly categorized into functional, performance, usability, compatibility, and security defects, along with other types like interface, logic, and database defects.
- Functional defects occur when the software fails to meet specified requirements
- While performance defects relate to issues like speed, stability, or scalability.
- Usability defects impact the ease of use
- compatibility defects arise when software doesn't work across different environments
- Security defects involve vulnerabilities that can be exploited.

☑ The Defects can be classified in following categories

(iii) Functional Defects
(iv) Performance Defects
(v) Usability Defects
(vi) Compatibility Defects
(vii) Security Defects

Que (27) Mention what big bang testing is?

- Big Bang integration testing is a software testing method where all components of a system are integrated and tested together at once.
- This approach, also known as non-incremental integration testing, is typically used when components have minimal interdependence or when development constraints limit incremental integration.
- It's a straight forward approach, but can make debugging difficult if issues arise.

☑ **Definition**: Big Bang integration testing involves combining all modules or components of a software application into a single, integrated system for testing.

☑ **Purpose:** It's used to verify that all components work together correctly as a whole after they've been individually unit tested.

☑ **Timing:** This approach is typically done after unit testing is complete, but before system testing.

☑ **Advantages**:

- It's simple to implement and doesn't require extensive planning, according to Bug Raptors.
- It's suitable for small, relatively simple systems where components have minimal interaction.

☑ **Disadvantages:**

- Debugging can be challenging if errors are found because it's difficult to pinpoint the source of the issue.
- The entire system must be recompiled and tested when a change is made, which can be time-consuming.

☑ **Alternatives:**

Other integration testing approaches include incremental integration testing, where components are integrated one by one, and sandwich integration testing, which combines top-down and bottom-up approaches.

Que (28) what is the purpose of exit criteria?

- Exit criteria in software testing define the conditions that must be met before a testing phase or the entire testing process can be considered complete.

- They essentially act as a guide for when testing should conclude, ensuring the software meets quality standards and project requirements.

☑ Here's a more detailed look at their purpose:

(i) Ensuring Quality and Completeness: Exit criteria help ensure that all testing goals are met and that the software is ready for the next stage of the development lifecycle.

(ii) Facilitating Decision-Making: They provide a clear framework for determining when testing is sufficient and when it's time to move forward, such as transitioning to a new testing phase or releasing the software.

(iii) Managing Risk: By specifying what needs to be addressed before testing is complete, exit criteria help minimize the risk of releasing software with critical defects.

(iv) Promoting Consistency: They ensure that all testers are working towards the same goals and that testing is performed in a consistent and efficient manner.

(v) Supporting Agile Development: In Agile, exit criteria are particularly important for defining when a sprint or iteration is complete and ready for delivery.

Que (29) what is white box testing and list the types of white box testing?

- White box testing, also known as glass-box or clear-box testing, involves testing a software application by examining its internal structure, code, and design.
- It's a structural testing method that allows testers to see the application's internal workings and use that knowledge to design test cases.
- The goal is to ensure that the code functions correctly and is optimized, identifying issues early on and improving reliability and security.

☑ Types of White Box Testing:

- **Unit Testing:** This type of testing focuses on individual units or components of the software, verifying that they function as expected in isolation.
- **Integration Testing:** Focuses on testing how different units or modules of a software system interact with each other to ensure they work together seamlessly.
- **Regression Testing:** Involves retesting existing code after modifications or bug fixes to ensure that the changes haven't introduced new issues.
- **Path Testing:** This technique involves testing all possible paths within a program's code, ensuring that all logic branches are covered.
- **Branch Testing:** This method focuses on ensuring that all possible branches (conditional statements) in the code are executed.
- **Loop Testing:** This type of testing is specifically designed to check the behaviour and logic of loops within the code.
- **Statement Coverage:** This technique aims to ensure that every statement within the code is executed during testing.
- **Data Flow Coverage:** This method tests the flow of data through the code, verifying that data is processed correctly.
- **Condition Coverage:** This technique ensures that all logical conditions within the code are tested and evaluated.
- **Mutation Testing:** This involves introducing slight errors or changes (mutations) to the code to determine if the existing tests can detect these errors.
- **Static Code Analysis:** This involves analyzing the code without executing it, using tools to identify potential vulnerabilities or coding errors.

- **Dynamic Analysis:** This involves analyzing the code while it is running, using tools to monitor its behaviour and identify potential issues.

White box testing is a crucial aspect of software development, as it provides a detailed view of the internal workings of the code and helps identify defects early on.

Que (30) what is black box testing? What are the different black box testing techniques?

- Black-box testing: Testing, either functional or non-functional, without reference to the internal structure of the component or system.
- Black-box testing is software testing methods where the internal workings or code structure of the system are not known to the tester. Testers focus solely on the inputs and outputs to ensure the software meets specific requirements and functions as intended.

☑ **Black Box Testing Techniques:**
There are four specification-based or black-box techniques:

(i) Equivalence partitioning
(ii) Boundary value analysis
(iii) Decision tables
(iv) State transition testing
(v) Use-case Testing
(vi) Other Black Box Testing:

- Syntax or Pattern Testing

Que (31) to create HLR & TestCase of

1) (Instagram, Facebook) only first page & Chat Functionality

- Instagram : <u>HLR</u> & <u>Testcase</u>
- Facebook : <u>HLR</u> & <u>Testcase</u>

Que (32) What is the difference between the STLC (Software Testing Life Cycle) and SDLC (Software Development Life Cycle)?

SDLC	STLC
• SDLC is mainly related to software development.	• STLC is mainly related to software testing.
• Besides development other phases like testing is also included.	• It focuses only on testing the software.
• SDLC involves total six phases or steps.	• STLC involves only five phases or steps.
• In SDLC, more number of members (developers) are required for the whole process.	• In STLC, less number of members (testers) are needed.
• In SDLC, development team makes the plans and designs based on the requirements.	• In STLC, testing team(Test Lead or Test Architect) makes the plans and designs.
• Goal of SDLC is to complete successful development of software.	• Goal of STLC is to complete successful testing of software.
• helps in developing good quality software.	• helps in making the software defects free.
• Post deployment support ,	• Regression tests are run by QA

enhancement , and update are to be included if necessary.	team to check deployed maintenance code and maintains test cases and automated scripts.
• Creation of reusable software systems is the end result of SDLC.	• A tested software system is the end result of STLC.

Que (33) What is the difference between test scenarios, test cases, and test script?

Test Scenario	**Test Cases**	**Test Script**
High-level functionality to be tested	Detailed steps to verify a scenario	Code to automate the test case
Identify all testable areas	Validate specific condition and workflow	Automate execution for efficiency
Broad and general	Specific and detailed	Very detailed and technical
Written in simple language	Written in step-by-step format	Write in programming/scripting language.
Used by Testers and QA leads	Used by Testers	Used by Automation testers/developers
EX. Check User login	EX. Open login page-Enter valid details-click login-check redirection	Selenium code to perform login and validate successful login

Que (34) Explain what Test Plan is? What is the information that should be covered.

- A Test Plan is a document that outlines how testing will be conducted for a specific project or software. It serves as a roadmap for the testing team, detailing the scope, approach, resources, and schedule for the testing process. The information covered in a test plan typically includes:

☑ **Key Information in a Test Plan:**

- **Scope:** Defining what will be tested and what will not be tested.
- **Objectives:** Identifying the goals of the testing process.
- **Test Strategy:** Outlining the approach and types of testing that will be used.
- **Schedule and Milestones:** Establishing a timeline for testing activities.
- **Test Environment:** Describing the environment in which testing will be performed.
- **Resources:** Identifying the personnel, tools, and data required for testing.
- **Roles and Responsibilities:** Assigning responsibilities for testing activities.
- **Entry and Exit Criteria:** Defining the conditions for starting and stopping testing.
- **Risks and Mitigation:** Identifying potential risks and outlining strategies to mitigate them.
- **Test Cases:** Developing specific test scenarios and cases.
- **Success Criteria:** Defining how the test results will be evaluated.
  In essence, a Test Plan provides a structured approach to ensure that testing is comprehensive, efficient, and aligned with project requirements.

Que (35) What is priority?

- In software testing, priority refers to the urgency or importance of fixing a defect, determining the order in which issues should be addressed.
- It's a way of prioritizing bug fixes based on their business impact, customer impact, and how much they hinder the development process.

☑ **Key aspects of Priority in Software Testing:**

☑ **Focus on Business Needs:**

Priority focuses on how much a bug impacts the business and how soon it needs to be fixed.

☑ **Determining the Order of Fixes:**

It helps decide which bugs should be addressed immediately and which can be postponed.

☑ **Prioritization is a Collaborative Effort:**

Assigning priorities often involves stakeholders, project managers, and product owners, who consider the impact of defects on business goals and user experience.

☑ **Examples of Priority Levels:**

Common levels include Critical (P1), High (P2), Medium (P3), and Low (P4).

☑ **Impact on Development Timeline:**

Prioritizing bugs helps ensure that the most critical issues are resolved first, allowing the development team to focus their efforts where they are needed most.

☑ **Separate from Severity:**
While related, priority and severity are distinct. Severity refers to the seriousness of a bug, while priority refers to how soon it needs to be fixed.

Que (36) What is severity?

- In software testing, severity refers to the impact a bug or defect has on the functionality or usability of the application or system.
- It's a way to classify bugs based on how much they disrupt the software's performance or how much they affect the user experience.
- Severity levels are often categorized as critical, major, moderate, minor, or trivial, with critical bugs causing a complete system failure and trivial bugs being minor, cosmetic issues.

☑ Key aspects of Priority in Software Testing:

- **Impact on Functionality:**

Severity measures how much a bug deviates from the expected behavior of the software. A higher severity level indicates a greater impact on the application's functionality.

- **User Experience:**

Severity also considers how a bug affects the end-user's experience. A bug that makes the software unusable would have a higher severity than one that causes only minor cosmetic issues.

- **Severity Levels:**

- **Critical:** These bugs cause the system to crash, lose data, or render essential functionalities unusable.

- **Major:** These bugs significantly impair functionality, but the system may still be partially usable.

- **Moderate:** These bugs cause partial loss of functionality or might affect some features, but the system can still be used.

- **Minor:** These bugs cause small issues that don't significantly impact the system's usability.

- **Trivial:** These bugs are very minor, often cosmetic issues, and don't affect the functionality of the system.

Relationship to Priority:

Severity and priority are related but distinct concepts. Severity measures the impact of a bug, while priority determines the order in which bugs should be addressed. A high-severity bug might be low-priority if it can be fixed later without significant impact on project timelines or business goals, according to Browser Stack.

Que (37) Bug categories are…

- In software testing, bug categories help classify software defects for better management and prioritization.
- They can be broadly categorized by their nature, impact, and severity, allowing testers and developers to focus on the most critical issues first.

☑ **Common Bug Categories:**

- **Functional Bugs:**

These bugs occur when a software feature doesn't work as intended, leading to incorrect or unexpected behavior.

- **Performance Bugs:**

These bugs affect the speed, stability, response time, or resource consumption of the software.

- **Security Bugs:**

These bugs create vulnerabilities that can be exploited by attackers, potentially leading to data breaches or other security incidents.

- **Usability Bugs:**

These bugs make the software difficult or confusing to use, affecting the user experience.

- **Logic Bugs:**

These bugs arise from errors in the design or implementation of the software's logic, leading to incorrect calculations or decision-making.

- **Compatibility Bugs:**

These bugs occur when the software doesn't function correctly with specific hardware, software, or operating systems.

- **Syntax Errors:**

These bugs are caused by mistakes in the syntax or structure of the code, such as typos or incorrect grammar.

- **System-Level Integration Bugs:**

These bugs occur when different components or modules of a software system fail to work together seamlessly.

- **Unit-Level Bugs:**

These bugs are isolated to specific units or modules of code within the software.

- **Regression Bugs:**

These bugs arise when new code changes introduce errors in previously working features or functionalities.

- **Boundary Bugs:**

These bugs occur when the software fails to handle edge cases or inputs at the boundaries of its expected input range.

- **Workflow Bugs:**

These bugs relate to errors in the sequence or flow of operations within the software.

- Severity and Priority:

Bugs are also categorized by their severity and priority to help prioritize bug fixing efforts.

- **Severity:**

This refers to the impact of the bug on the software's functionality and user experience. Examples include critical (showstoppers), major, moderate, and minor.

- **Priority:**

This refers to how urgently a bug needs to be addressed. Examples include urgent, high, medium, and low.

Que (38) Advantage of Bugzila....

☑ Advantage of Bugzila are following

- Enhanced Communication: Bugzilla facilitates clear and efficient communication between testers, developers, and other stakeholders by providing a centralized platform for reporting, tracking, and resolving issues.
- Comprehensive Reporting: The tool's robust reporting features allow users to generate various reports, charts, and graphs to analyze bug trends, project progress, and overall software quality.
- Robust Bug Tracking: Bugzilla's advanced search, filtering, and categorization options enable efficient tracking of bug lifecycle, including status, severity, and priority.
- Open-Source and Cost-Effective: Being an open-source tool, Bugzilla is free to use and deploy, making it a budget-friendly option for teams with limited resources.
- User-Friendly Interface: Bugzilla boasts a user-friendly interface that is easy to learn and use, even for those without extensive technical knowledge.
- Customization and Integration: The tool offers extensive customization options and can be integrated with various test management and other systems, enabling teams to tailor it to their specific needs.
- Increased Software Quality: By providing a structured framework for bug reporting and tracking, Bugzilla helps teams identify and resolve defects more efficiently, leading to improved software quality.
- Automation and Automation Integration: Bugzilla offers automation features like email notifications, and it can be integrated with automation testing frameworks to streamline the bug reporting process.

Que (39) Difference between priority and severity

Aspect	**Priority**	**Severity**
Meaning	Urgency to fix the bug	Impact of the bug on functionality
Focus	Business or customer urgency	Technical seriousness of the defect
Decided by	Project Manager, Client, Product Owner	Tester, Developer, QA lead
Ex1	Wrong company name on homepage->High	Login button not working->Critical
Ex2	Minor UI issue on main page->Medium	App crashes on specific action->Major
Ex3	Small typo on contact page-	Incorrect calculation in billing-

	>Low	>Major
Levels	High, Medium, Low, Critical	Major, Moderate, Minor, Cosmetic

Que (40) What are the different Methodologies in Agile Development Model?

- In the Agile Development Model, several methodologies can be employed for software testing, including Scrum, Kanban, Extreme Programming (XP), and others.
- These methodologies differ in their approach to testing and how it integrates with the development process.

☑ **Key Agile Methodologies and Their Impact on Testing**

- **Scrum:**

Scrum is a framework that emphasizes iterative development and testing. During each sprint, testing is integrated into the development cycle, ensuring that defects are identified early and addressed quickly.

- **Kanban:**

Kanban is a visual workflow management system that emphasizes continuous delivery and feedback. Testing is integrated into the Kanban board, allowing for a fluid and adaptive approach to testing throughout the development process.

- **Extreme Programming (XP):**

XP is a software development methodology that emphasizes testing as a core practice. Test-driven development (TDD) is a key component of XP, where tests are written before the code, driving the development process.

- **Feature-Driven Development (FDD):**

FDD focuses on developing software by prioritizing and delivering features. Testing is integrated into the feature development process, ensuring that each feature is thoroughly tested before being released.

- **Behavior-Driven Development (BDD):**

BDD uses a collaboration between developers, testers, and business stakeholders to define the desired behavior of the software. This allows for a more user-centric approach to testing, ensuring that the software meets the needs of the end-users.

Que (41) Explain the difference between Authorization and Authentication in Web testing.What are the common problems faced in Web testing?

☑ Difference between Authorization and Authentication in Web testing are following

- **Authentication**: Checks who the user is (example: username, password, OTP).
- **Authorization**: Checks what the user is allowed to do (example: user can view profile but not admin panel).

☑ Common problems faced in Web testing are following

- Broken links
- Slow page loading
- Poor UI on different devices or browsers
- Security issues (like SQL injection, XSS)
- Session timeout or logout not working
- Form input validation errors
- Incorrect error or success messages
- Compatibility issues on different browsers or screen sizes

Que (42) Write a Scenario of Pen

1. Verify that the length and the diameter of the pen are as per the specifications.
2. Verify the outer body material of the pen. Check if it is metallic, plastic, or any other material specified in the requirement specifications.
3. Verify that the brand name and/or logo of the company creating the pen should be clearly visible.
4. Verify the type of pen, whether it is a ballpoint pen, ink pen, or gel pen.
5. Verify that the user is able to write clearly over different types of papers.
6. Check the weight of the pen. It should be as per the specifications. In case not mentioned in the specifications, the weight should not be too heavy to impact its smooth operation.
7. Verify if the pen is with a cap or without a cap.
8. Verify the color of the ink on the pen.
9. Check the odor of the pen's ink on writing over a surface.
10. Verify the surfaces over which the pen is able to write smoothly apart from paper e.g. cardboard, rubber surface, etc.
11. Verify if the text written by the pen is erasable or not.
12. Check if the text written by the pen is waterproof or not.
13. Verify if the pen can support multiple refills or not.
14. For ink pens, verify that the mechanism to refill the pen is easy to operate.
15. In the case of a ball and gel pen, verify that the user can change the refill of the pen easily.
16. Check how fast the user can write with the pen over supported surfaces.
17. Verify the performance or the functioning of a pen when used continuously without stopping (called as Endurance Testing)
18. Verify the number of characters a user can write with a single refill in the case of ballpoint & gel pens and with full ink, in the case of ink or fountain pens.

Que (43) Write a Scenario of Pen Stand

1. Put different pens (ball pen, ink pen, gel pen) inside the pen stand.
2. Check if the pens fit well and don't fall or tilt.
3. Try putting other items like pencils, markers, or scissors into the stand.
4. Check if the pen stand stays steady on the table and doesn't fall over.
5. Take the pens out and put them back in several times to check if the stand stays strong.
6. Check if the stand can hold the number of pens, it says it can hold.
7. Look at the material (plastic, metal or wood) and check if it feels strong.
8. Check if the pen stand is easy to clean (remove dust or ink marks easily).
9. Check the design - make sure there are no sharp or loose parts that can hurt someone.
10. Check if the pen stand is light and easy to move.

Que (44) Write a Scenario of Door

1. Verify if the door is single door or bi-folded door.
2. Check if the door opens inwards or outwards.
3. Verify that the dimension of the doors are as per the specifications.
4. Verify that the material used in the door body and its parts is as per the specifications.
5. Verify that color of the door is as specified.
6. Verify if the door is sliding door or rotating door.
7. Check the position, quality and strength of hinges.
8. Check the type of locks in the door.
9. Check the number of locks in the door interior side or exterior side.
10. Verify if the door is having peek-hole or not.

11. Verify if the door is having stopper or not.
12. Verify if the door closes automatically or not – spring mechanism.
13. Verify if the door makes noise when opened or closed.
14. Check the door condition when used extensively with water.
15. Check the door condition in different climatic conditions- temperature, humidity etc.
16. Check the amount of force- pull or push required to open or close the door.

Que (45) Write a Scenario of ATM

- Check the informative text written displayed on the screen is clearly visible and legible.
- Verify that the application's UI is responsive i.e. it should adjust to different screen resolutions of ATM machines
- Verify the type of ATM machine, if it has a touch screen, both keypad buttons only, or both.
- Verify that on properly inserting a valid card different banking options appear on the screen.
- Check that no option to continue and enter credentials is displayed to the user when the card is inserted incorrectly.
- Verify that the touch of the ATM screen is smooth and operational.
- Verify that the user is presented with the option to choose a language for further operations.
- Check that the user is asked to enter a pin number before displaying any card/bank account detail.
- Verify that there is a limited number of attempts up to which the user is allowed to enter the pin code.
- Verify that if the total number of incorrect pin attempts gets surpassed then the user is not allowed to continue further. And operations like temporary blocking of the card, etc get initiated.
- Check that the pin is displayed in masked form when entered.
- Verify that the user is presented with different account type options like- saving, current, etc.
- Verify that the user is allowed to get account details like available balance
- Check that the correct amount of money gets withdrawn as entered by the user for cash withdrawal.
- Check that the user cannot withdraw more amount than the total available balance and a proper message should be displayed.
- Verify that the user's session timeout is maintained.
- Check that the user is not allowed to exceed one transaction limit amount.
- Verify that the user is not allowed to exceed the one-day transaction limit amount.
- Check that in case the ATM machine runs out of money, a proper message is displayed to the user. Check that in case the ATM machine runs out of money, a proper message is displayed to the user.
- Verify that the applicable fee gets deducted along with the withdrawn amount in case the user exceeds the limit of the number of free transactions in a month.
- Check that the user is not allowed to proceed with the expired ATM card and that a proper error message gets displayed.

Que (46) When to used Usablity Testing?

- Usability testing in software development should ideally be conducted throughout the product lifecycle, not just at the end.

- It's crucial to identify usability issues early and iteratively refine the design to improve user experience.
- This means testing during prototyping, before launch, and even post-launch to address evolving needs.

Que (47) What is the procedure for GUI Testing?

- Check all the GUI elements for size, position, width, length and acceptance of characters or numbers. For instance, you must be able to provide inputs to the input fields.
- Check you can execute the intended functionality of the application using the GUI
- Check Error Messages are displayed correctly
- Check for Clear demarcation of different sections on screen
- Check Font used in application is readable
- Check the alignment of the text is proper
- Check the Color of the font and warning messages is aesthetically pleasing
- Check that the images have good clarity
- Check that the images are properly aligned
- Check the positioning of GUI elements for different screen resolution.

Que (48) Write a scenario of Microwave Owen

1. Verify that the dimensions of the oven are as per the specification provided.
2. Verify that the oven's material is optimal for its use as an oven and as per the specification.
3. Verify that the oven heats the food at the desired temperature properly.
4. Verify that the oven heats food at the desired temperature within a specified time duration.
5. Verify the ovens functioning with the maximum attainable temperature.
6. Verify the ovens functioning with minimum attainable temperature.
7. Verify that the oven's plate rotation speed is optimal and not too high to spill the food kept over it.
8. Verify that the oven's door gets closed properly.
9. Verify that the oven's door opens smoothly.
10. Verify the battery requirement of the microwave oven and check that it function's smoothly at that power.
11. Verify that the text written over the oven's body is clearly readable.
12. Verify that the digital display is clearly visible and functions correctly.
13. Verify that the temperature regulator is smooth to operate.
14. Verify that the temperature regulator works correctly.
15. Check the maximum capacity of the oven and test its functioning with that volume of food.
16. Check the oven's functionality with different kinds of food – solid, and liquid.
17. Check the oven's functionality with different food at different temperatures.
18. Verify the oven's functionality with different kinds of container material.
19. Verify that the power cord of the oven is long enough.
20. Verify that the usage instruction or user manuals have clear instructions.

Que (49) Write a scenario of Coffee vending Machine

1. UI scenario – Verify that the dimension of the coffee machine is as per the specification.
2. Verify that outer body, as well as inner part's material, is as per the specification.

3. Verify that the machine's body color as well brand is correctly visible and as per specification.
4. Verify the input mechanism for coffee ingredients-milk, water, coffee beans/powder, etc.
5. Verify that the quantity of hot water, milk, coffee powder per serving is correct.
6. Verify the power/voltage requirements of the machine.
7. Verify the effect of suddenly switching off the machine or cutting the power. The machine should stop in that situation and in power resumption, the remaining coffee should not get come out of the nozzle.
8. Verify that coffee should not leak when not in operation.
9. Verify the amount of coffee served in single-serving is as per specification.
10. Verify that the digital display displays correct information.
11. Check if the machine can be switched on and off using the power buttons.
12. Check for the indicator lights when the machine is switched on-off.
13. Verify that the functioning of all the buttons work properly when pressed.
14. Verify that each button has an image/text with it, indicating the task it performs.
15. Verify that complete quantity of coffee should get poured in a single operation, no residual coffee should be present in the nozzle.
16. Verify the mechanism to clean the system work correctly- foamer.
17. Verify that the coffee served has the same and correct temperature each time it is served by the machine.
18. Verify that system should display an error when it runs out of ingredients.
19. Verify that pressing the coffee button multiple times leads to multiple serving of coffee.
20. Verify that there is the passage for residual/extra coffee in the machine.
21. Verify that machine should work correctly in different climatic, moistures and temperature conditions.
22. Verify that machine should not make too much sound when in operation.
23. Performance test – Check the amount of time the machine takes to serve a single serving of coffee.
24. Performance test – Check the performance of the machine when used continuously until the ingredients run out of the requirements.
25. Negative Test – Check the functioning of the coffee machine when two/multiple buttons are pressed simultaneously.
26. Negative Test – Check the functioning of coffee machine with a lesser or higher voltage than required.
27. Negative Test – Check the functioning of the coffee machine if the ingredient container's capacity is exceeded.

Que (50) Write a scenario of chair.

1. Verify that the chair is stable enough to take an average human load.
2. Check the material used in making the chair-wood, plastic etc.
3. Check if the chair's leg are level to the floor.
4. Check the usability of the chair as an office chair, normal household chair.
5. Check if there is back support in the chair.
6. Check if there is support for hands in the chair.
7. Verify the paint's type and color.
8. Verify if the chair's material is brittle or not.
9. Check if cushion is provided with chair or not.
10. Check the condition when washed with water or effect of water on chair.
11. Verify that the dimension of chair is as per the specifications.
12. Verify that the weight of the chair is as per the specifications.

13. Check the height of the chair's seat from floor.

Que (51) Write a scenario of only Whatsapp chat messages

☑ Messaging Features

- Verify that users can send and receive text messages in individual and group chats.
- Test for real-time message delivery indicators, including single ticks (sent), double ticks (delivered), and blue ticks (read).
- Check the ability to send and receive multimedia files like images, videos, documents, and voice notes.

☑ Voice and Video Calls

- Validate the ability to initiate, receive, and end voice and video calls.
- Test call quality under different network conditions.
- Ensure that group calls function correctly, including adding or removing participants during a call.

☑ Group Features

- Test group creation, adding and removing members, and assigning admin roles.
- Validate notifications for group-specific actions, such as name or icon changes.
- Check message delivery and interaction within groups.

☑ Media Sharing

- Verify the sharing of images, videos, audio files, documents, and location.
- Test media compression and quality after sharing.
- Check the proper display and playback of shared content.

☑ Privacy and Security

- Test end-to-end encryption for messages and calls.
- Validate privacy settings for last seen, profile photo, and status updates.
- Check the functionality of two-step verification.

☑ Notifications

- Ensure timely notifications for new messages, calls, and updates, even when the app is in the background.
- Test customizable notification settings, such as tones and vibrations.

☑ Cross-Device Functionality

- Verify seamless syncing of messages and updates across devices.
- Test the logout and login functionality for linked devices.

☑ Performance

- Check the app's responsiveness during heavy usage, such as high traffic in group chats.
- Test the app's performance in low-network conditions.

These scenarios ensure that WhatsApp remains reliable, secure, and user-friendly across all its features and functionalities.

Que (52) Write a Scenario of Wrist Watch

- Verify the type of watch – analog or digital.
- In the case of an analog watch, check the correctness time displayed by the second, minute, and hour hand of the watch.
- In the case of a digital watch, check the digital display for hours, minutes, and seconds is correctly displayed.
- Verify the material of the watch and its strap.
- Check if the shape of the dial is as per specification.
- Verify the dimension of the watch is as per the specification.
- Verify the weight of the watch.
- Check if the watch is waterproof or not.
- Verify that the numbers in the dial are clearly visible or not.
- Check if the watch is having a date and day display or not.
- Verify the color of the text displayed in the watch – time, day, date, and other information.
- Verify that clock's time can be corrected using the key in case of an analog clock and buttons in case of a digital clock.
- Check if the second hand of the watch makes ticking sound or not.
- Verify if the brand of the watch and check if its visible in the dial.
- Check if the clock is having stopwatch, timers, and alarm functionality or not.
- In the case of a digital watch, verify the format of the watch 12 hours or 24 hours.
- Verify if the watch comes with any guarantee or warranty.
- Verify if the dial has glass covering or plastic, check if the material is breakable or not.
- Verify if the dial's glass/plastic is resistant to minor scratches or not.
- Check the battery requirement of the watch.

Que (53) Write a Scenario of Lift(Elevator)

- Verify the dimensions of the lift.
- Verify the type of door of the lift is as per the specification.
- Verify the type of metal used in the lift interior and exterior.
- Verify the capacity of the lift in terms of the total weight.
- Verify the buttons in the lift to close and open the door and numbers as per the number of floors.
- Verify that the lift moves to the particular floor as the button of the floor is clicked.
- Verify that the lift stops when the up/down buttons on a particular floor are pressed.
- Verify if there is an emergency button to contact officials in case of any mishap.
- Verify the performance of the floor – the time taken to go to a floor.
- Verify that in case of power failure, the lift doesn't free-fall and gets halted on the particular floor.
- Verify lifts working in case the button to open the door is pressed before reaching the destination floor.
- Verify that in case the door is about to close and an object is placed between the doors if the doors sense the object and again open or not.
- Verify the time duration for which the door remains open by default.
- Verify if the lift interior is having proper air ventilation.
- Verify lighting in the lift.
- Verify that at no point the lift door should open while in motion.
- Verify that in case of power loss, there should be a backup mechanism to safely get into a floor or a backup power supply.

- Verify that in case the multiple floor number button is clicked, the lift should stop on each floor.
- Verify that in case of capacity limit is reached users are prompted with a warning alert- audio/visual.
- Verify that inside lift users are prompted with the current floor and direction information the lift is moving towards- audio/visual prompt.

Que (54) Write a Scenario of Whatsapp payment.

- Verify that the WhatsApp app is updated to the latest version.
- Verify that the payment option is visible in the chat or attachment menu.
- Verify that the user's bank account is linked to WhatsApp.
- Check if the user can select a contact to send payment.
- Verify that the user can enter the amount correctly.
- Verify that the app asks for the UPI PIN before sending money.
- Check if the payment completes successfully after entering the correct PIN.
- Verify that the user and receiver both get payment confirmation messages.
- Check if the payment history shows the correct details.
- Verify that if the user enters the wrong PIN, the payment fails with a proper error message.
- Check if payment fails when the internet is off or slow, and the user sees an error.
- Verify that the app shows a proper message if the bank server is down or busy.