Module – 2

1. What is Exploratory Testing?

- Exploratory testing is a software testing approach that emphasizes simultaneous learning, test design, and test execution. Instead of following a predefined set of test cases, testers explore the application in an ad-hoc manner, using their intuition and experience to uncover defects, usability issues, and other aspects of software quality.

2. What is traceability matrix?

- A traceability matrix is a tool used in project management and software development to ensure that all requirements are met throughout the lifecycle of a project. It maps requirements to their corresponding test cases, design elements, and other project artifacts.

3. What is Boundary value testing?

- Boundary value testing is a software testing technique that focuses on testing the boundaries or edge cases of input values rather than just the typical or expected values. The rationale behind this approach is that many errors in software occur at the boundaries of input ranges.

4. What is Equivalence partitioning testing?

- Equivalence partitioning is a software testing technique used to reduce the number of test cases while still maintaining effective test coverage. The idea is to divide input data into groups, or "partitions," where all the values in a group are expected to behave similarly. This allows testers to focus on representative values rather than testing every possible input.

5. What is Integration testing?

- Integration testing is a software testing phase where individual components or systems are combined and tested as a group. The primary goal is to identify defects in the interactions between integrated components and ensure they work together as intended.

6. 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.

- A Risk could be any future event with a negative consequence .You need to identify the risks associated with your project Risks are of two types Project Risks and Product Risk.

7. What is Alpha testing?

- Alpha testing is an early phase of software testing conducted to identify bugs and issues before the software is released to a larger audience. It typically takes place in the development environment and is usually performed by internal teams, such as developers and quality assurance personnel.

8. What is beta testing?

- Beta testing is a phase of software testing that follows alpha testing and involves real users testing the software in a production-like environment. The primary goal is to identify any remaining bugs or issues and gather feedback on the software's functionality, usability, and overall user experience before the final release.

9. What is component testing?

- Component testing, also known as unit testing, is a software testing practice that focuses on validating the functionality of individual components or modules of a software application. The primary goal is to ensure that each component performs as expected in isolation before they are integrated with other parts of the system.

10. What is functional system testing?

- Functional system testing is a type of software testing that evaluates the functionality of an entire software application to ensure that it meets specified requirements. This testing focuses on verifying that the system behaves as expected when used in a real-world environment.

11. What is Non-Functional Testing?

- Non-Functional Testing refers to testing aspects of a software application that do not relate to specific functionalities. Instead, it focuses on how the system performs under certain conditions.

12. What is GUI Testing?

- Graphical User Interface Testing, is a type of software testing that focuses on evaluating the visual elements of an application. The goal is to ensure that the user interface is functional, user-friendly, and visually appealing.

13. What is Adhoc testing?

- Adhoc testing is an informal and unstructured type of software testing conducted without any formal test plan or documentation. The primary goal is to identify defects in the application through random testing and exploratory techniques.

14. What is load testing?

- Load testing is a type of performance testing that evaluates how a system behaves under a specific expected load. The primary goal is to determine the application's capacity to handle a defined number of users or transactions simultaneously and to identify any performance bottlenecks.

15. What is stress Testing?

- Stress testing is a type of performance testing that evaluates how a system behaves under extreme conditions, beyond its normal operational capacity. The primary goal is to determine the system's breaking point and to assess how it recovers from failures.

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

- White box testing, also known as clear box, glass box, or structural testing, is a software testing method that examines the internal workings and structure of the application. Testers have knowledge of the code, algorithms, and architecture, allowing them to design test cases based on the software's internal logic.

- Types of White Box Testing

1. Unit Testing: Tests individual components or modules of the code in isolation to ensure they function correctly.
2. Integration Testing: Evaluates the interactions between different modules to identify interface defects.

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

- Black box testing is a software testing method where the tester evaluates the functionality of an application without any knowledge of its internal code structure or implementation details. The focus is on testing the application's outputs based on various inputs, ensuring that it meets the specified requirements and behaves as expected from a user's perspective.

- Different Black Box Testing Techniques

1. Equivalence Partitioning: Divides input data into equivalent partitions to reduce the number of test cases. Each partition should yield the same output, so testing one value from each partition is sufficient.
2. Boundary Value Analysis: Focuses on testing values at the boundaries of input ranges. Since errors often occur at the edges, this technique checks just above, below, and at the boundaries.
3. Decision Table Testing: Uses a table to represent different input combinations and their corresponding outputs. It helps ensure that all possible scenarios are tested.
4. Regression Testing: Ensures that new code changes do not adversely affect existing functionalities. It tests previously developed and tested software to confirm that it still performs correctly after updates.

18. Mention what are the categories of defects?

- Defects in software can be categorized based on various criteria, such as their nature, severity, and origin. Here are some common categories of defects:

### 1. Functional Defects

* Incorrect Functionality: The feature does not work as specified.
* Missing Functionality: A required feature is absent or not implemented.

### 2. Performance Defects

* Slow Response Time: The application takes too long to respond to user actions.
* Memory Leaks: The application consumes more memory over time, leading to resource exhaustion.

### 3. Usability Defects

* Poor User Interface: The interface is confusing or not user-friendly.
* Accessibility Issues: The application does not accommodate users with disabilities.

### 4. Compatibility Defects

* Browser Compatibility: The application behaves inconsistently across different web browsers.
* Device Compatibility: Issues arise when using the application on various devices (e.g., mobile vs. desktop).

### 5. Security Defects

* Vulnerabilities: Weaknesses that could be exploited by attackers (e.g., SQL injection, cross-site scripting).
* Improper Authentication/Authorization: Flaws in user access controls that allow unauthorized access.

### 6. Logical Defects

* Algorithm Errors: Incorrect logic in calculations or processing.
* Conditional Errors: Mistakes in conditional statements that lead to unexpected behavior.

### 7. Data Defects

* Data Integrity Issues: Problems with data accuracy or consistency.
* Data Loss: Information is lost or corrupted during processing.

### 8. Regression Defects

* New Bugs in Existing Features: Issues that arise in previously working
* functionalities due to code changes.

19. Mention what bigbang testing is?

- Big Bang Testing is a type of software testing approach where all the components or modules of a software application are integrated and tested simultaneously, rather than following a phased integration process. This method is typically used when development is done in an unstructured or ad-hoc manner, leading to a lack of a systematic integration strategy.

20. What is the purpose of exit criteria?

- Exit criteria are the predefined conditions that must be met before a testing phase or the entire testing process can be considered complete. They help ensure that testing is thorough and that the software is ready for release or the next phase of development. The purpose of exit criteria includes:

### **1. Quality Assurance**

* Ensures that the software meets the required quality standards before it is released to users.

### **2. Risk Management**

* Helps identify any remaining risks or defects that could impact the software's performance or user experience.

### **3. Decision-Making**

* Provides a basis for stakeholders to make informed decisions about whether the software is ready for deployment.

### **4. Resource Optimization**

* Allows for efficient allocation of resources by clearly defining when testing efforts should cease, preventing unnecessary expenditure of time and effort.

### **5. Measuring Progress**

* Serves as a benchmark for assessing the progress of testing activities, helping teams stay on track.

### **6. Documentation and Accountability**

* Establishes a documented agreement among stakeholders about when testing can be considered complete, fostering accountability and transparency.

21. When should "Regression Testing" be performed?

- Regression testing should be performed in several key scenarios to ensure that new code changes do not adversely affect existing functionalities. Here are the primary situations when regression testing is necessary:

### **1. After Bug Fixes**

* When defects have been identified and fixed, regression testing verifies that the fixes work and have not introduced new issues.

### **2. After New Features or Enhancements**

* When new features are added or existing features are modified, regression testing ensures that these changes do not impact the current functionality.

### **3. Before Releases or Deployments**

* Prior to deploying the application to production, regression testing helps confirm that the software remains stable and that all features function as expected.

### **4. After Code Refactoring**

* When the code is refactored for optimization or improvement, regression testing checks that the refactor did not introduce any unintended defects.

### **5. Following Integration of New Components**

* When integrating new modules or third-party services, regression testing ensures that the overall system still functions correctly with the new additions.

### **6. After Environmental Changes**

* If there are changes to the software environment, such as updates to libraries, frameworks, or the underlying operating system, regression testing confirms that the application remains compatible and functional.

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

- The "7 key principles of software testing" provide a foundational framework for effective testing practices. These principles help guide testers in their approach, ensuring thorough and efficient testing. Here’s a detailed explanation of each principle:

### **1. Testing Shows the Presence of Defects**

* **Explanation**: Testing can demonstrate that defects exist in the software but cannot prove that there are no defects. A successful test may indicate that a specific functionality works correctly, but it doesn’t guarantee that all functionalities are defect-free.
* **Implication**: This principle underscores the importance of thorough testing and encourages teams to adopt a mindset focused on finding defects rather than assuming that a lack of failures indicates quality.

### **2. Exhaustive Testing is Impossible**

* **Explanation**: Due to the vast number of possible inputs, conditions, and paths within a software application, it is impractical to test every possible scenario. Complete testing would require infinite resources and time.
* **Implication**: Testers must prioritize testing based on risk, focusing on critical functionalities and high-impact areas rather than attempting to achieve exhaustive coverage.

### **3. Early Testing**

* **Explanation**: Testing should begin as early as possible in the software development lifecycle (SDLC). Early testing can help identify defects sooner, reducing costs and effort associated with fixing issues later in the process.
* **Implication**: Implementing practices such as unit testing and continuous integration can help identify and address issues early, leading to improved quality and reduced rework.

### **4. Defect Clustering**

* **Explanation**: A small number of modules or components often contain the majority of the defects. This phenomenon is known as defect clustering. Understanding where defects are likely to occur helps testers focus their efforts more effectively.
* **Implication**: By analyzing past defect data and focusing on high-risk areas, testing efforts can be optimized to address the most problematic components.

### **5. Pesticide Paradox**

* **Explanation**: Running the same set of tests repeatedly will not uncover new defects. Just as pests develop resistance to pesticides, software can become resistant to specific tests, meaning they become less effective over time.
* **Implication**: Testers must continuously update and evolve their testing strategies and test cases to include new scenarios, techniques, and data to uncover hidden defects.

### **6. Testing is Context-Dependent**

* **Explanation**: The approach to testing can vary significantly based on the context, including the type of application, its criticality, regulatory requirements, and user expectations. For example, testing a safety-critical application requires a different approach than testing a simple mobile app.
* **Implication**: Testers should tailor their strategies and methodologies to fit the specific context of the project, considering factors such as user needs, business goals, and technical requirements.

### **7. Absence of Errors is a Fallacy**

* **Explanation**: Just because a product passes all tests does not mean it is the right product for users or that it meets their needs. The absence of defects does not equate to quality if the software does not fulfill user requirements or business objectives.
* **Implication**: Testing should not only focus on finding defects but also ensure that the application meets functional requirements and delivers a satisfactory user experience.

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

- Quality Assurance (QA), Quality Control (QC), and Tester are related concepts in the software development and testing field, but they have distinct roles and purposes. Here’s a breakdown:

### **Quality Assurance (QA)**

* **Focus:** Process-oriented.
* **Purpose:** Ensures that the processes involved in software development are followed correctly to prevent defects.
* **Activities:** Includes defining processes, standards, and methodologies. QA aims to improve and establish effective processes for development and testing.
* **Outcome:** A consistent process that helps deliver high-quality products.

### **Quality Control (QC)**

* **Focus:** Product-oriented.
* **Purpose:** Involves the actual testing of the product to identify defects.
* **Activities:** Includes reviewing, inspecting, and testing the final product to ensure it meets the specified requirements and standards.
* **Outcome:** A validated product that meets quality standards and user expectations.

### **Tester**

* **Focus:** Specific role within the testing phase.
* **Purpose:** Executes tests on the software product to identify bugs and issues.
* **Activities:** Writing test cases, performing tests (manual or automated), reporting defects, and verifying fixes.
* **Outcome:** Detailed feedback on the software’s functionality and quality, contributing to the QC process.

24. Difference between Smoke and Sanity?

- Smoke testing and sanity testing are both types of software testing, but they serve different purposes and contexts.

- smoke testing is broad and assesses the overall health of the application, while sanity testing is narrow and checks specific functionalities after changes. Both are essential in ensuring software quality but are used at different stages and for different purposes.

25. Difference between verification and Validation

- Verification and validation are two crucial concepts in software testing and development, often confused but distinctly different. Here’s a breakdown of their differences:

### **Verification**

* **Definition:** The process of evaluating work products (like requirements, design, and code) to ensure they meet the specified requirements and are correct.
* **Focus:** Ensures that the product is being built correctly according to specifications.
* **Activities:** Includes reviews, inspections, and static analysis. It typically involves checking documents and processes.
* **Outcome:** Confirms that the product design and development processes align with the requirements before moving on to the next phase.
* **Example:** Reviewing requirement documents to ensure they are complete and consistent.

### **Validation**

* **Definition:** The process of evaluating the finished product to ensure it meets the user’s needs and requirements in a real-world environment.
* **Focus:** Ensures that the right product has been built.
* **Activities:** Involves dynamic testing, including functional testing, system testing, and user acceptance testing.
* **Outcome:** Confirms that the software works as intended in the actual use cases and meets user expectations.
* **Example:** Conducting user acceptance testing to verify that the software satisfies user requirements and works in a real-world scenario.

26. Explain types of Performance testing.

- **1. Load Testing**

* **Purpose:** To determine how the application performs under expected user loads.
* **Focus:** Assessing system behavior and performance under normal and peak load conditions.
* **Activities:** Gradually increasing the load (number of concurrent users) until the application reaches its maximum capacity.
* **Outcome:** Identifies the maximum load the system can handle without performance degradation.

### **2. Stress Testing**

* **Purpose:** To evaluate how the application behaves under extreme load conditions.
* **Focus:** Pushing the system beyond its limits to see how it reacts to stress.
* **Activities:** Introducing a higher-than-expected load, often sudden spikes, to observe how the system handles the overload.
* **Outcome:** Identifies breaking points, system failures, and recovery capabilities.

### **3. Endurance Testing (Soak Testing)**

* **Purpose:** To verify how the system performs under a sustained load over an extended period.
* **Focus:** Assessing performance, stability, and resource usage over time.
* **Activities:** Running the application with a normal load for an extended period (hours or days).
* **Outcome:** Detects potential memory leaks, resource depletion, and long-term performance issues.

### **4. Spike Testing**

* **Purpose:** To analyze how the system reacts to sudden and extreme increases in load.
* **Focus:** Observing performance when there’s a sharp rise in user activity.
* **Activities:** Quickly increasing the load and then decreasing it to see how the system handles the abrupt change.
* **Outcome:** Evaluates the application’s ability to manage sudden traffic spikes and recover afterward.

### **5. Volume Testing**

* **Purpose:** To assess how the system handles a large volume of data.
* **Focus:** Evaluating the application's performance when processing a significant amount of data.
* **Activities:** Feeding the system with large volumes of data to test its processing capabilities.
* **Outcome:** Identifies performance issues related to data storage, retrieval, and processing.

### **6. Configuration Testing**

* **Purpose:** To determine how different configurations of the system impact performance.
* **Focus:** Testing the application under various hardware and software configurations.
* **Activities:** Varying parameters like server specifications, network settings, and software versions.
* **Outcome:** Identifies optimal configurations for maximum performance.

### **7. Scalability Testing**

* **Purpose:** To assess how well the application can scale up or down in response to changes in load.
* **Focus:** Evaluating the application’s ability to handle growth.
* **Activities:** Gradually increasing the load while monitoring system performance and resource usage.
* **Outcome:** Determines if the application can scale effectively without performance loss.

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

- The terms error, defect, bug, and failure are often used interchangeably in software development, but they have distinct meanings. Here’s a breakdown of each:

### **1. Error**

* **Definition:** A mistake made by a programmer during coding or designing.
* **Context:** Errors are often human mistakes, such as syntax errors, logical errors, or incorrect assumptions.
* **Example:** A programmer accidentally uses the wrong variable name, leading to incorrect logic in the code.

### **2. Defect**

* **Definition:** A flaw in the software that causes it to behave unexpectedly or incorrectly.
* **Context:** Defects can arise from errors in coding, design, or requirements. They are typically identified through testing.
* **Example:** A feature that does not function as intended, such as a button that does not trigger the expected action when clicked.

### **3. Bug**

* **Definition:** A common term for a defect in the software.
* **Context:** Bugs are often the result of errors in the code that lead to unexpected behavior or incorrect results.
* **Example:** An application crashing when a user tries to save a file due to an unhandled exception.

### **4. Failure**

* **Definition:** The manifestation of a defect when the software fails to perform its intended function in a real-world scenario.
* **Context:** A failure occurs during execution when the software does not work as expected, typically observed by users.
* **Example:** A system outage that occurs because the application cannot handle a surge in users, leading to a complete inability to access the service.

28. Difference between Priority and Severity

- **Severity** is about the impact of the defect on the application’s functionality, while **Priority** is about the urgency of fixing that defect based on business considerations.

* A defect can have high severity but low priority (e.g., a crash in a seldom-used feature) or low severity but high priority (e.g., a bug affecting a major product launch). Understanding both helps teams effectively manage defects and deliver quality software.

29. What is Bug Life Cycle?

- The bug life cycle (or defect life cycle) is the process that a bug goes through from its initial identification to its resolution and closure. Here’s an overview of the typical stages in the bug life cycle:

### **1. New**

* **Description:** The bug has been identified and reported, but it has not yet been reviewed or assigned.
* **Action:** The bug report is created and logged into the tracking system.

### **2. Assigned**

* **Description:** The bug has been reviewed and assigned to a developer or team for investigation and resolution.
* **Action:** The responsible team member begins analyzing the bug.

### **3. Open**

* **Description:** The developer is currently working on fixing the bug.
* **Action:** This stage indicates active engagement with the issue.

### **4. Fixed**

* **Description:** The developer has made the necessary changes to resolve the bug.
* **Action:** The fix is typically committed to the codebase, and the bug is marked as fixed.

### **5. Retest**

* **Description:** The bug is re-evaluated by the testing team to confirm that the fix is effective.
* **Action:** The testers run the relevant test cases to ensure the bug no longer exists.

### **6. Closed**

* **Description:** The bug has passed retesting and is considered resolved.
* **Action:** The bug is marked as closed in the tracking system, indicating it is no longer an issue.

### **7. Reopened**

* **Description:** If the bug persists after retesting, it can be reopened.
* **Action:** The bug is sent back to the developer for further investigation and fixes.

### **8. Deferred**

* **Description:** The bug is acknowledged but not scheduled for immediate resolution, often due to prioritization or resource allocation.
* **Action:** The team decides to address it in a future release or sprint.

### **9. Rejected**

* **Description:** The reported issue is determined not to be a bug, either due to a misunderstanding of functionality or other reasons.
* **Action:** The bug report is marked as rejected and closed, often with an explanation.

30. Explain the difference between Functional testing and NonFunctional testing

- Functional testing and non-functional testing are two essential categories of software testing, each focusing on different aspects of an application. Here’s a breakdown of their differences:

### **Functional Testing**

* **Definition:** Testing that verifies the software application against the functional requirements/specifications.
* **Focus:** Ensures that the application behaves as expected and that all functionalities work correctly.
* **Types:** Includes unit testing, integration testing, system testing, user acceptance testing (UAT), and regression testing.
* **Activities:** Involves checking specific functionalities such as user login, data processing, and feature interactions.
* **Example:** Verifying that a user can successfully log into an application, submit a form, or make a purchase.

### **Non-Functional Testing**

* **Definition:** Testing that evaluates the non-functional aspects of a software application, such as performance, usability, reliability, and scalability.
* **Focus:** Ensures that the application meets certain criteria regarding its performance and user experience rather than its specific functionalities.
* **Types:** Includes performance testing, load testing, stress testing, usability testing, security testing, and compatibility testing.
* **Activities:** Involves assessing how the application performs under load, its response time, user interface design, and security vulnerabilities.
* **Example:** Measuring how many users the application can handle simultaneously without crashing or assessing the application’s response time under various conditions.

31. To create HLR & TestCase of 1)(Instagram , Facebook) only first page

- <https://docs.google.com/spreadsheets/d/103J4LJCCjzzIt5mM4BlYNd2em5qehSvOAB4YkXmAYCc/edit?usp=sharing>

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

- The Software Testing Life Cycle (STLC) and Software Development Life Cycle (SDLC) are both essential processes in software development, but they focus on different aspects.

### Software Development Life Cycle (SDLC)

1. Definition: SDLC refers to the complete process of software development, from planning to deployment and maintenance.
2. Phases:
   * Requirement Analysis: Gathering and analyzing requirements.
   * Design: Architectural and detailed design of the software.
   * Implementation: Writing the actual code.

### Software Testing Life Cycle (STLC)

1. Definition: STLC focuses specifically on the testing process, ensuring that the software is free of defects and meets quality standards.
2. Phases:
   * Requirement Analysis: Understanding testing requirements based on software specifications.
   * Test Planning: Creating a strategy and plan for testing.
   * Test Case Development: Designing test cases and scripts.
   * Test Environment Setup: Preparing the environment where testing will occur.
   * Test Execution: Running the tests and documenting results.

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

- Test scenarios, test cases, and test scripts are all important components of software testing, but they serve different purposes and have distinct characteristics. Here's a breakdown of each:

### **Test Scenarios**

* **Definition**: A test scenario is a high-level description of a functionality or a feature that needs to be tested. It outlines what will be tested without going into detailed steps.
* **Purpose**: To provide an overview of the functionality and ensure that all aspects are covered during testing.
* **Example**: "Verify user login functionality."

### **Test Cases**

* **Definition**: A test case is a detailed set of conditions or variables under which a tester will determine whether a system or part of a system is working as intended. It includes specific inputs, execution steps, and expected results.
* **Purpose**: To provide a clear and structured approach for executing tests and documenting results.
* **Components**:
  + **Test Case ID**: Unique identifier.
  + **Description**: Brief explanation of what the test case is for.
  + **Preconditions**: Any setup required before executing the test.
  + **Test Steps**: Detailed instructions on how to perform the test.
  + **Expected Result**: What should happen if the system works correctly.
* **Example**: "Enter valid username and password, then click 'Login'. Expected result: User should be directed to the dashboard."

### **Test Scripts**

* **Definition**: A test script is a set of instructions written in a programming or scripting language to automate testing. It can be used to execute test cases automatically.
* **Purpose**: To automate the execution of tests, which saves time and ensures consistency, especially for regression testing.
* **Components**: Similar to test cases, but formatted for execution in a testing tool or framework.
* **Example**: A script written in Selenium to automate the login process, specifying the actions to input credentials and validate the outcome.

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

- A **Test Plan** is a formal document that outlines the strategy, scope, resources, and schedule for testing activities within a software project. It serves as a blueprint for the testing process, ensuring that all aspects of testing are systematically addressed.

### **Key Components of a Test Plan**

1. **Test Plan Identifier**: A unique ID for the test plan for reference.
2. **Introduction**: Overview of the project and the purpose of the test plan.
3. **Objectives and Scope**:
   * **Objectives**: What the testing aims to achieve (e.g., identify defects, verify functionality).
   * **Scope**: What will and will not be included in testing (features, modules, platforms).
4. **Test Items**: List of features or components to be tested, including versions or configurations.
5. **Test Approach**: The overall testing strategy, including types of testing (functional, performance, security) and methods (manual, automated).
6. **Resource Requirements**: Identification of resources needed, including personnel, tools, and environments.
7. **Roles and Responsibilities**: Clear definition of who is responsible for different testing tasks (testers, developers, project managers).
8. **Test Environment**: Description of the environment where testing will take place, including hardware, software, and network configurations.
9. **Test Schedule**: Timeline for the testing activities, including key milestones and deadlines.
10. **Risk Assessment**: Identification of potential risks that could impact testing and mitigation strategies.

35. What is priority?

- **Key Aspects of Priority**

1. **Definition**: Priority indicates how soon a test case should be executed or how quickly a defect should be resolved. It is usually defined by stakeholders, including project managers, developers, and testers.
2. **Categories**:
   * **High Priority**: These issues or test cases should be addressed immediately because they significantly impact the project or user experience. For example, critical bugs that prevent the application from functioning.
   * **Medium Priority**: Important but not urgent issues that should be fixed in the normal course of development. They may affect functionality but don't halt progress.
   * **Low Priority**: Issues or test cases that can be addressed later, as they have minimal impact on the project or user experience. These may include cosmetic issues or less critical functionalities.

36. What is severity?

- **Key Aspects of Severity**

1. **Definition**: Severity measures the degree to which a defect affects the system's functionality, performance, or usability. It is typically assessed by testers and is often classified on a scale.
2. **Categories**:
   * **Critical**: The defect causes a complete system failure or crashes the application, preventing users from performing any tasks. Immediate resolution is required.
   * **Major**: The defect severely impairs functionality but does not crash the system. Workarounds may exist, but they significantly affect user experience.
   * **Moderate**: The defect affects functionality in a noticeable way but is not critical. It may cause inconvenience, but the system can still be used effectively.
   * **Minor**: The defect has little to no impact on functionality. It may be a cosmetic issue or a minor annoyance that does not hinder overall use.
   * **Trivial**: The defect is negligible and does not affect the user experience significantly, such as typos or slight formatting issues.
3. **Assessment**: Severity is usually determined by the testing team based on the defect's impact on the application's core functionalities and the user experience.

37. Bug categories are

- Bug categories refer to the different classifications of defects found in software applications. These categories help in organizing, prioritizing, and managing bugs effectively. Here are some common bug categories:

### **1. Functional Bugs**

* **Definition**: Issues that occur when the software does not behave as specified in the requirements.
* **Examples**: Incorrect calculations, missing features, or failure to execute commands.

### **2. Performance Bugs**

* **Definition**: Problems that affect the speed, responsiveness, or efficiency of the application.
* **Examples**: Slow load times, memory leaks, or high resource consumption.

### **3. Usability Bugs**

* **Definition**: Issues that affect the user experience and make the application difficult to use.
* **Examples**: Confusing navigation, poor layout, or unclear error messages.

### **4. Security Bugs**

* **Definition**: Vulnerabilities that can be exploited to compromise the integrity, confidentiality, or availability of the application.
* **Examples**: SQL injection flaws, improper authentication, or lack of encryption.

### **5. Compatibility Bugs**

* **Definition**: Issues that arise when the software does not function correctly across different environments, devices, or platforms.
* **Examples**: Browser compatibility problems or application crashes on certain operating systems.

### **6. Integration Bugs**

* **Definition**: Problems that occur when different modules or systems fail to work together as expected.
* **Examples**: Data not being shared correctly between integrated systems or APIs returning unexpected results.

### **7. Localization/Internationalization Bugs**

* **Definition**: Issues related to the adaptation of the software for different languages and regions.
* **Examples**: Incorrect translations, formatting issues with dates or currencies.

### **8. Configuration Bugs**

* **Definition**: Problems caused by incorrect settings or configurations in the application or environment.
* **Examples**: Misconfigured servers or improper environment settings leading to failures.

### **9. Regression Bugs**

* **Definition**: Defects introduced after changes or updates in the code that affect previously working functionality.
* **Examples**: A feature that worked in an earlier version suddenly fails after a new release.

38. Advantage of Bugzila .

- Bugzilla is a popular open-source bug tracking system that offers several advantages for managing software defects and project issues. Here are some key benefits:

### **1. User-Friendly Interface**

* Bugzilla provides an intuitive and straightforward interface, making it easy for users to report, track, and manage bugs.

### **2. Customization**

* The tool is highly customizable, allowing teams to tailor workflows, fields, and categories to fit their specific processes and requirements.

### **3. Robust Reporting Features**

* Bugzilla offers comprehensive reporting and querying capabilities, enabling users to generate detailed reports on bug status, trends, and team performance.

### **4. Advanced Search Functionality**

* Users can easily search for bugs using various criteria, such as status, priority, severity, and assigned personnel, which helps in quickly locating relevant issues.

### **5. Email Notifications**

* The system can send automatic email notifications about bug updates, comments, and status changes, keeping all team members informed in real time.

### **6. Version Control Integration**

* Bugzilla can be integrated with version control systems, making it easier to associate bugs with specific code changes and track their resolution.

### **7. Support for Multiple Projects**

* The tool supports managing multiple projects simultaneously, making it suitable for teams that work on various applications or systems.

### **8. Open Source and Free**

* Being an open-source tool, Bugzilla is free to use and can be modified as needed, which is beneficial for organizations with limited budgets.

### **9. Community Support**

* Bugzilla has a strong user community that provides support, documentation, and plugins, enhancing its usability and functionality.

### **Conclusion**

Overall, Bugzilla is a powerful tool for managing bugs and project issues, providing teams with the features and flexibility needed to improve software quality and streamline their development processes.

39. Difference between priority and severity

- **Priority** and **severity** are two important concepts in software testing and defect management, but they serve different purposes. Here’s a breakdown of their differences:

### **Priority**

* **Definition**: Priority indicates the urgency or importance of addressing a defect. It reflects how quickly a bug needs to be fixed based on business needs or user impact.
* **Focus**: Priority is determined by the stakeholders, often considering business goals, user experience, and project deadlines.
* **Categories**:
  + **High Priority**: Needs immediate attention; critical for business operations.
  + **Medium Priority**: Important but can be scheduled for later.
  + **Low Priority**: Can be fixed at a later time without significant impact.

### **Severity**

* **Definition**: Severity refers to the impact level of a defect on the functionality of the application. It assesses how serious the defect is in terms of its effect on the system and users.
* **Focus**: Severity is typically determined by the testing team, based on the technical aspects of the defect and its effect on system behavior.
* **Categories**:
  + **Critical**: Causes complete failure or crashes the system.
  + **Major**: Significantly impairs functionality but does not crash the system.
  + **Moderate**: Affects usability but is not critical.

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

- Agile development encompasses several methodologies, each with its own practices, principles, and frameworks. Here are some of the most prominent methodologies within the Agile development model:

### **1. Scrum**

* **Description**: Scrum is a framework that emphasizes iterative progress through a series of fixed-length sprints (typically 2-4 weeks).
* **Key Components**: Roles (Scrum Master, Product Owner, Development Team), events (Sprint Planning, Daily Scrum, Sprint Review, Sprint Retrospective), and artifacts (Product Backlog, Sprint Backlog, Increment).

### **2. Kanban**

* **Description**: Kanban focuses on visualizing work in progress and managing flow, emphasizing continuous delivery without fixed iterations.
* **Key Components**: Kanban board (to visualize tasks), Work In Progress (WIP) limits, and continuous improvement practices.

### **3. Extreme Programming (XP)**

* **Description**: XP is a methodology aimed at improving software quality and responsiveness to changing requirements through frequent releases and customer feedback.
* **Key Practices**: Pair programming, test-driven development (TDD), continuous integration, and collective code ownership.

### **4. Lean Software Development**

* **Description**: Lean focuses on optimizing efficiency and reducing waste in the development process, borrowing principles from lean manufacturing.
* **Key Principles**: Eliminate waste, amplify learning, decide as late as possible, deliver as fast as possible, empower the team, and build integrity in.

### **5. Feature-Driven Development (FDD)**

* **Description**: FDD is a model-driven, short-iteration process that focuses on delivering tangible, working software features.
* **Key Components**: Building a feature list, planning by feature, and designing and building features in iterations.

### **6. Crystal**

* **Description**: Crystal is a family of methodologies that emphasize the importance of people and interactions over processes. It adapts to the size and criticality of the project.
* **Key Variants**: Crystal Clear, Crystal Yellow, Crystal Orange, etc., each suited for different team sizes and project complexities.

### **7. Dynamic Systems Development Method (DSDM)**

* **Description**: DSDM is an Agile project delivery framework that emphasizes the full project lifecycle and collaboration between stakeholders.
* **Key Features**: Focus on delivering business value, iterative development, and user involvement throughout the process.

### **8. Agile Unified Process (AUP)**

* **Description**: AUP is an Agile version of the Rational Unified Process (RUP), integrating Agile practices into the RUP framework.
* **Key Phases**: Inception, Elaboration, Construction, and Transition, with iterative and incremental development.

### **9. Scrumban**

* **Description**: A hybrid of Scrum and Kanban, Scrumban combines the structure of Scrum with the flexibility of Kanban.
* **Key Features**: Iterations (like Scrum) while allowing continuous flow (like Kanban), including WIP limits and visual management.

### **10. DevOps**

* **Description**: While not strictly an Agile methodology, DevOps emphasizes collaboration between development and operations teams to enhance deployment frequency and reliability.
* **Key Practices**: Continuous integration, continuous delivery, and infrastructure as code.

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

- **Authentication**

* **Definition**: This is the process of verifying the identity of a user. It ensures that the user is who they claim to be.
* **Methods**: Common methods include username and password, multi-factor authentication (MFA), and biometric verification.
* **Testing Focus**: Web testers focus on ensuring that the authentication mechanisms are secure, reliable, and functioning correctly. This includes testing for vulnerabilities like brute-force attacks, session hijacking, and ensuring password policies are enforced.

### **Authorization**

* **Definition**: This refers to the process of determining what an authenticated user is allowed to do. It defines permissions and access levels for different users.
* **Methods**: Authorization can be role-based (RBAC), attribute-based (ABAC), or based on specific permissions.
* **Testing Focus**: In testing, the focus is on verifying that users can only access resources and perform actions that they are permitted to. This includes testing for unauthorized access and privilege escalation vulnerabilities.

### **Common Problems in Web Testing**

1. **Cross-Site Scripting (XSS)**: Attackers inject malicious scripts into web pages viewed by users.
2. **SQL Injection**: Improper input validation allows attackers to manipulate queries to databases.
3. **Session Management Flaws**: Issues with session expiration, fixation, and hijacking can compromise security.
4. **Broken Authentication**: Weaknesses in authentication mechanisms can allow unauthorized access.
5. **Insecure Direct Object References (IDOR)**: Users access resources they shouldn't have permission to, by manipulating URLs or parameters.
6. **Poorly Implemented Authorization**: Users may gain access to features or data outside their permissions.

42. To create HLR & TestCase of WebBased (WhatsApp web , Instagram) 1. WhatsApp Web

- <https://docs.google.com/spreadsheets/d/1vxm3qhOOimnFWcIcxjJe7QWQqKU0bv8S-dktVOdeeuE/edit?usp=sharing>

43. To create HLR and TestCase on this Link. https://artoftesting.com/

- <https://docs.google.com/spreadsheets/d/1iR4BGPeYeA6D6S69K8mtyq0dDgeGtx56HOBR0O3mXjQ/edit?usp=sharing>

44. Write a scenario of only Whatsapp chat messages

-

<https://docs.google.com/spreadsheets/d/1K4R_Xrr1IAPyg4W4zc8M3TPQXtO0Wb8A4e1Df7GjIx4/edit?usp=sharing>

45. Write a Scenario of Pen

<https://docs.google.com/spreadsheets/d/1K4R_Xrr1IAPyg4W4zc8M3TPQXtO0Wb8A4e1Df7GjIx4/edit?usp=sharing>

46. Write a Scenario of Pen Stand

<https://docs.google.com/spreadsheets/d/1K4R_Xrr1IAPyg4W4zc8M3TPQXtO0Wb8A4e1Df7GjIx4/edit?usp=sharing>

47. Write a Scenario of Door

<https://docs.google.com/spreadsheets/d/1K4R_Xrr1IAPyg4W4zc8M3TPQXtO0Wb8A4e1Df7GjIx4/edit?usp=sharing>

48. Write a Scenario of ATM

[https://docs.google.com/spreadsheets/d/1K4R\_Xrr1IAPyg4W4zc8M3TPQXtO0Wb8A4e1Df7GjIx4/edit?usp=sharing](https://docs.google.com/spreadsheets/d/1K4R_Xrr1IAPyg4W4zc8M3TPQXtO0Wb8A4e1Df7GjIx4/edit?usp=sharing%20%20)

49. When to used Usablity Testing?

- Usability testing should be used at various stages of the product development process to ensure that the final product meets user needs and expectations. Here’s when to conduct usability testing:

### **1. Early in the Design Process**

* **Concept Validation**: Test initial ideas, sketches, or wireframes to gather feedback before full development begins.
* **Identify User Needs**: Understand user expectations and pain points early on.

### **2. During Prototyping**

* **Low-Fidelity Prototypes**: Test basic layouts and interactions to refine concepts based on user feedback.
* **Iterative Design**: Conduct multiple rounds of testing as prototypes evolve, allowing for ongoing adjustments.

### **3. Before Development**

* **Design Confirmation**: Validate usability of final designs before coding begins, ensuring the design is user-friendly.

### **4. During Development**

* **Ongoing Testing**: Test functional prototypes or builds to identify usability issues as features are developed.
* **Feature Testing**: Ensure new features are intuitive and enhance user experience.

### **5. Post-Launch**

* **Real-World Evaluation**: Assess how users interact with the product in a live environment, identifying any lingering issues.
* **User Feedback for Improvements**: Gather insights to inform future updates or enhancements.

### **6. Before Major Updates or Redesigns**

* **Impact Assessment**: Test significant changes to ensure they improve usability and do not create new problems.
* **User Adaptation**: Understand how changes affect existing user workflows.

50. What is the procedure for GUI Testing?

- GUI (Graphical User Interface) testing is essential to ensure that the user interface of an application is intuitive, functional, and visually appealing. Here’s a typical procedure for conducting GUI testing:

### **1. Planning**

* **Define Objectives**: Identify what aspects of the GUI need testing (e.g., usability, layout, functionality).
* **Select Tools**: Choose appropriate testing tools for automated GUI testing, if necessary.

### **2. Test Case Design**

* **Identify Test Scenarios**: Outline the key functionalities of the application that require testing.
* **Create Test Cases**: Develop detailed test cases that cover:
  + Layout and design (alignment, colors, fonts)
  + Functional elements (buttons, links, forms)
  + Responsiveness (behavior on different devices and screen sizes)
  + Accessibility (compliance with accessibility standards)
* **Prioritize Test Cases**: Focus on critical paths that impact user experience.

### **3. Environment Setup**

* **Prepare Test Environment**: Set up the testing environment that mirrors the production environment.
* **Install Required Software**: Ensure all necessary software and tools are installed.

### **4. Execution of Test Cases**

* **Manual Testing**: Execute the test cases manually to evaluate the GUI’s functionality and appearance.
* **Automated Testing**: Use automation tools to run test scripts for repetitive tasks and regression testing.
* **Record Results**: Document any defects, inconsistencies, or issues encountered during testing.

### **5. Defect Reporting**

* **Log Defects**: Report any identified issues with detailed descriptions, steps to reproduce, and screenshots if necessary.
* **Prioritize Issues**: Categorize defects based on severity and impact on user experience.

### **6. Retesting and Regression Testing**

* **Retest Fixed Issues**: Verify that reported defects have been resolved.
* **Perform Regression Testing**: Ensure that new changes haven’t affected existing functionalities.

### **7. User Acceptance Testing (UAT)**

* **Involve End Users**: If possible, conduct UAT to gather feedback from real users about the GUI.
* **Collect Feedback**: Use feedback to make any necessary adjustments before final deployment.

### **8. Final Evaluation**

* **Review Testing Results**: Analyze the results of the testing phase and ensure all critical issues are addressed.
* **Prepare Documentation**: Create a summary report detailing testing activities, findings, and resolutions.

### **9. Deployment and Monitoring**

* **Deploy the Application**: Release the application to the production environment.
* **Monitor User Feedback**: Keep an eye on user feedback post-launch for any unforeseen GUI issues.

51. Write a scenario of Microwave Owen

<https://docs.google.com/spreadsheets/d/1K4R_Xrr1IAPyg4W4zc8M3TPQXtO0Wb8A4e1Df7GjIx4/edit?usp=sharing>

52. Write a scenario of Coffee vending Machine

<https://docs.google.com/spreadsheets/d/1K4R_Xrr1IAPyg4W4zc8M3TPQXtO0Wb8A4e1Df7GjIx4/edit?usp=sharing>

53. Write a scenario of chair

<https://docs.google.com/spreadsheets/d/1K4R_Xrr1IAPyg4W4zc8M3TPQXtO0Wb8A4e1Df7GjIx4/edit?usp=sharing>

54. To Create Scenario (Positive & Negative)

1> Gmail (receiving mail)

<https://docs.google.com/spreadsheets/d/1K4R_Xrr1IAPyg4W4zc8M3TPQXtO0Wb8A4e1Df7GjIx4/edit?usp=sharing>

2> Online shopping to buy product (flipkart)

<https://docs.google.com/spreadsheets/d/1K4R_Xrr1IAPyg4W4zc8M3TPQXtO0Wb8A4e1Df7GjIx4/edit?usp=sharing>

55. Write a Scenario of Wrist Watch

<https://docs.google.com/spreadsheets/d/1K4R_Xrr1IAPyg4W4zc8M3TPQXtO0Wb8A4e1Df7GjIx4/edit?usp=sharing>

56. Write a Scenario of Lift(Elevator)

<https://docs.google.com/spreadsheets/d/1K4R_Xrr1IAPyg4W4zc8M3TPQXtO0Wb8A4e1Df7GjIx4/edit?usp=sharing>

57. Write a Scenario of whatsapp Group (generate group)

<https://docs.google.com/spreadsheets/d/1K4R_Xrr1IAPyg4W4zc8M3TPQXtO0Wb8A4e1Df7GjIx4/edit?usp=sharing>

58. Write a Scenario of Whatsapp payment

<https://docs.google.com/spreadsheets/d/1K4R_Xrr1IAPyg4W4zc8M3TPQXtO0Wb8A4e1Df7GjIx4/edit?usp=sharing>

S