Module – 2

1. What is Exploratory Testing ?

**Exploratory Testing** is a type of software testing where test cases are not created in advance, but testers actively explore the application while testing it. The main goal is to discover bugs or unexpected behavior by using creativity, intuition, and experience.

**Key Characteristics of Exploratory Testing:**

* 🧠 **Simultaneous Learning, Test Design, and Execution**: Testers learn about the application as they test it.
* 📋 **Less Documentation**: Unlike scripted testing, it doesn’t rely on predefined test cases.
* 🔄 **Dynamic and Adaptive**: Testers decide what to test next based on what they’ve already seen.
* 👩‍💻 **Tester-Centric**: Relies heavily on the tester’s domain knowledge and experience.
* 💡 **Uncovers Hidden Bugs**: Often finds issues that scripted tests might miss.

**When to Use Exploratory Testing:**

* The application is still evolving or not well-documented.
* You want to quickly find critical bugs.
* You need to test creatively and flexibly.
* There’s limited time for testing.

**Example:**

Imagine you're testing a login page. Instead of only checking valid and invalid credentials (as in scripted testing), you might try:

* Entering emojis or special characters in the username field.
* Pasting very long strings.
* Checking the behavior without an internet connection.

1. What is Boundary value testing ?

**Boundary Value Testing (BVT)** is a **black-box testing technique** where test cases are designed to include values at the **edges or boundaries** of input ranges. Since bugs often occur at the boundaries, this testing helps catch issues that might not be found with normal input values.

**✅ Why Use Boundary Value Testing?**

* Most errors occur at the **boundaries** rather than in the center of the input domain.
* It helps ensure the system behaves correctly at **minimum, maximum, just below, and just above** the boundary limits.

**🧪 Example:**

Suppose an input field accepts numbers **from 1 to 100**.

You would test with:

* **Lower boundary**: 0 (just below), **1** (minimum), **2** (just above)
* **Upper boundary**: 99 (just below), **100** (maximum), **101** (just above)

So, your test values:  
0, 1, 2, 99, 100, 101

**🧠 Benefits of Boundary Value Testing:**

* Simple yet effective.
* Finds edge-case bugs early.
* Works well when input is in a range (like age, quantity, etc.).

1. What is Integration testing ?

Integration Testing checks if different modules of a software system work together correctly. It helps find issues in data flow and communication between components.

**Example:**  
In an online shopping app, the **Login, Payment, and Order** modules must work together. If payment is successful but the order is not placed, there is an integration issue.

**Types of Integration Testing:**

1. **Top-Down** – Test higher-level modules first.
2. **Bottom-Up** – Test lower-level modules first.
3. **Big Bang** – Test everything at once.
4. **Incremental** – Test step by step.
5. What is Alpha testing ?

**Alpha Testing** is a type of software testing performed **before the product is released to the public**. It is conducted by **internal testers or developers** in a controlled environment to identify bugs and improve software quality.

**Key Points:**

* Done at the **end of development** but **before Beta Testing**.
* Conducted in a **lab or development environment**.
* Helps find **major bugs and usability issues**.

**Example:**

A company developing a **mobile app** tests it internally before releasing it to selected users for Beta Testing.

1. What is beta testing?

**Beta Testing** is a type of software testing where a nearly finished product is released to a **limited group of real users** to gather feedback before the official launch. It helps identify bugs, usability issues, and performance problems in a real-world environment.

**Key Points:**

* Conducted **after Alpha Testing** but **before the final release**.
* Performed by **real users in a real environment**.
* Helps improve **user experience and software reliability**.

**Example:**

A company developing a **new mobile app** releases it to a **small group of users** for feedback before launching it publicly.

1. What is component testing ?

**Component Testing** (also called **Module Testing**) is a software testing technique where **individual components or modules** of an application are tested separately to verify their functionality before integrating them with other components.

**Key Features of Component Testing:**

* Tests a **single, independent module** to ensure it works correctly.
* Typically done **before integration testing**.
* Helps in identifying **bugs early** within a specific module.
* Can be done **manually or using automation tools**.

**Purpose of Component Testing:**

* To **verify the functionality** of individual components.
* To ensure that each module **meets the design and requirements**.
* To detect **defects at an early stage**, reducing costs.

**Example:**

Consider an **e-commerce application** where the **shopping cart module** is tested separately to check:

* If products can be added and removed correctly.
* If the total price updates as expected.
* If emptying the cart resets all values.

**Types of Component Testing:**

1. **White Box Testing** – Tests the internal code and logic of a module.
2. **Black Box Testing** – Tests the functionality without checking the internal code.
3. What is functional system testing ?

**Functional System Testing** is a type of software testing that checks whether a complete system **meets functional requirements**. It ensures that all features work correctly as expected in an integrated environment.

**Key Features:**

* Tests the **entire system** rather than individual components.
* Focuses on **what the system should do** (functionality).
* Based on **requirements and specifications**.

**Example:**

In an **online banking system**, Functional System Testing would check:

* Whether users can log in successfully.
* If fund transfers work correctly.
* Whether account balance updates properly.

**Types of Functional System Testing:**

1. **Smoke Testing** – Quick checks to see if the system is stable.
2. **Sanity Testing** – Verifies specific functionality after small changes.
3. **Regression Testing** – Ensures new updates don’t break existing features.
4. **User Acceptance Testing (UAT)** – Confirms the system meets business needs.
5. What is Non-Functional Testing ?

**What is Non-Functional Testing?**

**Non-Functional Testing** is a type of software testing that checks the **performance, security, usability, and reliability** of a system rather than its functional correctness. It focuses on **how well** the software performs rather than **what** it does. This testing ensures that the system works efficiently under different conditions, providing a smooth user experience.

**Purpose of Non-Functional Testing:**

* To verify that the system meets **performance benchmarks** like speed and responsiveness.
* To ensure **security** by protecting data from threats.
* To check **usability**, making sure the application is easy to use.
* To test **scalability**, ensuring the system can handle more users over time.

**Examples of Non-Functional Testing:**

* **Web Application:** Checking if an e-commerce website can handle **5,000 users** at once without slowing down.
* **Desktop Application:** Ensuring that video editing software runs smoothly when processing **large 4K videos**.
* **Mobile Application:** Testing an online banking app to confirm that **login and transactions remain secure**.
* **Game Application:** Verifying that an online multiplayer game runs **without lag** when many players join.

**Types of Non-Functional Testing:**

1. **Performance Testing** – Measures system speed and response time.
2. **Security Testing** – Checks for vulnerabilities and data protection.
3. **Usability Testing** – Ensures the application is user-friendly.
4. **Load Testing** – Tests system behavior under high traffic.
5. **Scalability Testing** – Determines if the system can support growth.
6. What is GUI Testing ?

**What is GUI Testing?**

**GUI (Graphical User Interface) Testing** is a type of software testing that checks the **visual elements** of an application to ensure they function correctly and provide a good user experience. It verifies the appearance, behavior, and responsiveness of UI components like buttons, menus, text boxes, and images.

**Purpose of GUI Testing:**

* To ensure the **UI elements** are displayed correctly on different screen sizes and resolutions.
* To check that buttons, forms, and links work as expected.
* To verify **color, font, alignment, and design consistency**.
* To ensure the application is **user-friendly and accessible**.

**Examples of GUI Testing:**

* **Web Application:** Testing an e-commerce website to ensure the **"Add to Cart"** button works properly and the product images load correctly.
* **Desktop Application:** Checking if a **word processor** displays menus, text formatting, and icons correctly.
* **Mobile Application:** Verifying that a **food delivery app** adjusts properly to different screen sizes and touch interactions.
* **Game Application:** Ensuring that game UI elements like **scoreboards, health bars, and control buttons** are displayed correctly and function smoothly.

**Types of GUI Testing:**

1. **Manual Testing** – Testers visually inspect and interact with the UI.
2. **Automated Testing** – Tools like Selenium, TestComplete, or Appium test UI functionality.
3. **Cross-Browser Testing** – Ensures UI looks and works correctly on different browsers.
4. **Responsive Testing** – Verifies that UI adapts well to different screen sizes and devices.
5. What is load testing ?

**What is Load Testing?**

**Load Testing** is a type of performance testing that checks how a system behaves under a specific number of users, transactions, or data load. It helps determine whether the application can handle expected traffic without slowing down or crashing.

**Purpose of Load Testing:**

* To measure **system response time** under normal and peak conditions.
* To identify **performance bottlenecks** before deployment.
* To ensure the system remains **stable and functional** under load.
* To help in **scalability planning** for future traffic growth.

**Examples of Load Testing:**

* **Web Application:** Testing an **e-commerce website** to see if it can handle **5,000 users** shopping at the same time.
* **Desktop Application:** Checking how a **video editing software** performs when processing **large 4K video files**.
* **Mobile Application:** Evaluating a **food delivery app** when thousands of users place orders during peak hours.
* **Game Application:** Testing an **online multiplayer game** to check if the servers can support **1,000 players** in a single match.

**Load Testing Tools:**

* **JMeter** – Open-source tool for web applications.
* **LoadRunner** – Enterprise tool for large-scale testing.
* **Gatling** – Used for continuous performance testing.
* **Apache Bench (AB)** – Simple tool for testing web server performance.

1. What is stress Testing?

**Stress Testing** is a type of performance testing that evaluates how a system behaves under **extreme conditions**, such as heavy traffic, high data processing, or limited resources. It helps determine the system's **breaking point** and ensures it recovers gracefully after failure.

**Purpose of Stress Testing:**

* To check system **stability and reliability** under extreme load.
* To identify **performance bottlenecks** and failure points.
* To ensure the system **recovers smoothly** after failure.
* To test how the system handles **unexpected spikes in traffic**.

**Examples of Stress Testing:**

* **Web Application:** Testing an **online ticket booking system** by simulating **100,000 users** trying to buy tickets at the same time.
* **Desktop Application:** Checking a **photo editing software** by opening **hundreds of high-resolution images** at once.
* **Mobile Application:** Simulating **millions of users** logging into a banking app during a major financial event.
* **Game Application:** Stress-testing an **online multiplayer game server** by flooding it with **excessive requests** beyond its normal capacity.

**Stress Testing Tools:**

* **JMeter** – Used for extreme load testing.
* **LoadRunner** – Enterprise tool for large-scale performance testing.
* **NeoLoad** – Simulates high traffic loads.
* **Locust** – Python-based tool for stress testing web applications.

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

**What is White Box Testing?**

**White Box Testing** (also known as **Glass Box, Clear Box, or Structural Testing**) is a software testing method that examines the **internal structure, code, and logic** of an application. Testers have full knowledge of the **source code** and use it to design test cases.

**Purpose of White Box Testing:**

* To verify the **internal working** of the application.
* To ensure **code efficiency, correctness, and security**.
* To identify **logical errors, hidden bugs, and security vulnerabilities**.

**Types of White Box Testing:**

1. **Unit Testing** – Tests individual components or functions of the software.
2. **Integration Testing** – Ensures different modules work together correctly.
3. **Control Flow Testing** – Examines the logical flow of the program using control statements (if, else, loops).
4. **Data Flow Testing** – Checks how data moves within the application and detects unused or uninitialized variables.
5. **Branch Testing** – Ensures that every possible branch in the code (if-else conditions) is executed at least once.
6. **Loop Testing** – Focuses on validating loops (for, while, do-while) in the program to check for infinite or incorrect looping.
7. **Path Testing** – Ensures every possible execution path in the program is tested to avoid hidden errors.
8. **Mutation Testing** – Modifies small parts of the code (mutants) to check if the test cases detect these changes.
9. **Security Testing** – Analyzes the code for vulnerabilities like SQL injection, cross-site scripting (XSS), and buffer overflows.

**Example of White Box Testing:**

* In a **banking application**, a tester might check the **fund transfer function's code** to ensure that the correct balance is deducted from one account and added to another.
* In a **gaming application**, a tester might review the **collision detection algorithm** to verify that a player’s character interacts correctly with the environment.

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

**What is Black Box Testing?**

**Black Box Testing** is a software testing method that examines the functionality of an application **without looking at its internal code, structure, or logic**. The tester focuses on **inputs and expected outputs** to ensure the system behaves correctly.

**Purpose of Black Box Testing:**

* To verify that the software works as per **requirements**.
* To test the **user interface, functionality, and overall behavior**.
* To identify **functional bugs, UI issues, and incorrect outputs**.

**Different Black Box Testing Techniques:**

1. **Equivalence Partitioning** – Divides input data into groups (valid and invalid) and tests one value from each group.  
   *Example:* If a system accepts ages **18–60**, then test **18, 30, 60** as valid and **17, 61** as invalid.
2. **Boundary Value Analysis (BVA)** – Tests the system at the **extreme edges** of valid input ranges.  
   *Example:* For a password field that accepts **6–12 characters**, test inputs with **5, 6, 12, and 13** characters.
3. **Decision Table Testing** – Uses a table to test different combinations of inputs and their expected outcomes.  
   *Example:* In a **login system**, if a user enters a correct username but a wrong password, it should display an error.
4. **State Transition Testing** – Tests how the system behaves when transitioning between different states.  
   *Example:* In an ATM, after **three wrong PIN attempts**, the account should be locked.
5. **Use Case Testing** – Tests the system based on real-world user scenarios.  
   *Example:* In an **e-commerce app**, verify the complete purchase process: add to cart → checkout → payment → order confirmation.
6. **Error Guessing** – Relies on tester experience to guess potential errors.  
   *Example:* Entering **blank fields, special characters, or negative values** to see if the system handles them correctly.

**Example of Black Box Testing:**

* In a **banking application**, a tester enters a valid account number and amount to check if the transfer works correctly, without looking at the underlying code.
* In a **mobile app**, testers verify that tapping the "Sign Up" button leads to the registration page.

1. Mention what bigbang testing is?

**What is Big Bang Testing?**

**Big Bang Testing** is an **integration testing approach** where all components or modules of a system are combined and tested **together at once**, rather than in phases. It is performed after all individual units have been developed and integrated.

**Key Features of Big Bang Testing:**

* All modules are tested **simultaneously** after integration.
* No incremental testing; the entire system is tested in one go.
* Suitable for **small projects** but risky for large systems.

**Advantages of Big Bang Testing:**

✔ Simple to implement, as everything is tested at once.  
✔ Useful when all components are available at the same time.

**Disadvantages of Big Bang Testing:**

❌ Hard to **identify the root cause** of errors due to multiple modules being tested together.  
❌ **High risk** if major issues are found late in development.  
❌ Debugging and fixing issues can be time-consuming.

**Example of Big Bang Testing:**

Consider a **school management system** with modules like **Student Registration, Attendance, Fee Payment, and Exam Management**. In Big Bang Testing, all these modules are integrated first and tested together, rather than testing them one by one.

1. What is the purpose of exit criteria ?

**Exit criteria** in software testing are predefined conditions that must be met to conclude a testing phase or the entire testing process. They serve as benchmarks to ensure that all necessary testing activities have been completed and that the software meets the required quality standards before progressing to the next development stage or release.

**Purpose of Exit Criteria:**

* **Ensuring Completion of Testing Activities:** Confirm that all planned test cases have been executed and documented.
* **Defect Resolution:** Verify that all identified defects have been addressed, and no high-priority issues remain unresolved.
* **Assessing Software Quality:** Evaluate whether the software meets performance, stability, security, and usability standards.
* **Facilitating Informed Decision-Making:** Provide stakeholders with objective criteria to decide on moving the software to the next phase or release.

1. When should "Regression Testing" be performed ?

**Regression Testing** should be performed in the following situations to ensure that recent code changes haven't adversely affected existing functionalities:

1. **After Bug Fixes:** To confirm that resolved issues haven't reintroduced old defects or created new ones.
2. **Following New Feature Implementations:** To ensure that added functionalities integrate seamlessly without disrupting current operations.
3. **Post Performance Enhancements:** To verify that optimizations haven't negatively impacted existing features.
4. **During Integration Phases:** To check that combined modules function correctly as a whole.
5. **Before Major Releases:** To detect any last-minute issues that could affect user experience or software performance.
6. **After Configuration Changes:** To ensure that alterations in the environment or settings don't disrupt existing functionalities.
7. What is 7 key principles? Explain in detail?

In software testing, seven fundamental principles guide the testing process to ensure effective and efficient evaluation of software quality.

1. **Testing Shows the Presence of Defects:** Testing can reveal the existence of defects in software but cannot confirm their absence. It reduces the probability of undiscovered issues but doesn't guarantee a defect-free product.
2. **Exhaustive Testing is Impossible:** Testing all possible input combinations and scenarios is impractical due to resource and time constraints. Instead, risk-based and prioritized testing focuses on the most critical areas.
3. **Early Testing Saves Time and Money:** Initiating testing activities early in the software development lifecycle helps detect defects at initial stages, reducing the cost and effort required for fixes later.
4. **Defects Cluster Together:** A small number of modules often contain most of the defects. Identifying and focusing testing efforts on these defect-prone areas can enhance efficiency.
5. **Pesticide Paradox:** Repeating the same set of tests can become ineffective over time as they no longer find new defects. Regularly reviewing and updating test cases is essential to uncover different issues.
6. **Testing is Context-Dependent:** Testing approaches should be tailored to the specific context of the software application, considering factors like industry, regulatory requirements, and user expectations.
7. **Absence-of-Errors Fallacy:** Even if software is free of defects, it may not meet user needs or requirements. Ensuring that the software fulfills intended purposes is as crucial as identifying defects.
8. Difference between QA v/s QC v/s Tester ?

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | **Aspect** | **Quality Assurance (QA)** | **Quality Control (QC)** | **Tester** | | --- | --- | --- | --- | | **Focus** | Process-oriented; focuses on preventing defects by improving and ensuring the correct processes are in place. | Product-oriented; focuses on identifying defects in the actual products before they are released. | Execution of tests to detect defects in the software product. | | **Objective** | To enhance and standardize processes to prevent defects during the development lifecycle. | To detect and correct defects in the finished product before delivery. | To execute test cases and report defects to ensure the software functions as intended. | | **Activities** | - Process definition and implementation- Training and documentation- Process audits and improvements | - Actual testing of the product- Inspection and validation against requirements- Identifying and reporting defects | - Designing test cases- Executing tests- Reporting and verifying defects- Collaborating with developers to resolve issues | | **Timing** | Proactive; implemented throughout the development process to ensure quality standards are met from the beginning. | Reactive; performed after the product is developed to identify any defects before release. | Engaged during the testing phases of the development cycle, typically after the initial development and before the product release. | | **Orientation** | Preventive; aims to establish a quality management system and processes to avoid defects. | Detective; aims to identify defects in the actual products through testing. | Detective; focuses on finding defects through test execution. | | **Responsibility** | Involves everyone in the development process; it's a collective responsibility to follow established processes. | Typically the responsibility of a dedicated QC or testing team that evaluates the product. | Testers are responsible for carrying out tests, identifying defects, and ensuring that the software meets the specified requirements and standards. | | **Scope** | Encompasses the entire development process, ensuring methodologies and standards are in place. | Focuses on specific deliverables, ensuring they meet the required quality standards. | Concentrates on the operational aspects of the software, validating its functionality and performance through testing. | |

1. Difference between Smoke and Sanity ?

**Smoke Testing**

* **Type**: Build verification testing
* **Purpose**: To check whether the major functionalities are working or not
* **Scope**: Wide but not deep (basic functionality only)
* **Performed when**: After a new software build is released
* **Goal**: To ensure the build is stable enough for further testing
* **Example**: Checking if the application opens, login works, and main page loads

**Sanity Testing**

* **Type**: Subset of regression testing
* **Purpose**: To verify that a specific bug fix or feature works as expected
* **Scope**: Narrow and deep (focuses on a particular area)
* **Performed when**: After minor changes or bug fixes
* **Goal**: To confirm that the new changes didn't break existing functionality
* **Example**: After fixing the login bug, testing only the login functionality in detail

1. Difference between verification and Validation?

**Verification**

* It is the process of checking whether the software is developed correctly according to the design and requirements.
* It is process-oriented.
* It is done during the development phase.
* It includes activities like reviews, inspections, and walkthroughs.
* It does not involve actual execution of the code.
* It answers the question "Are we building the product right?"

**Validation**

* It is the process of checking whether the software meets the business needs and user expectations.
* It is product-oriented.
* It is done after the development is complete.
* It includes activities like functional testing, system testing, and user acceptance testing.
* It involves execution of the code.
* It answers the question "Are we building the right product?"

1. Explain types of Performance testing ?

**1. Load Testing**

* **Purpose**: To check how the system performs under expected number of users.
* **Goal**: Find system behavior under normal load.
* **Example**: Test a website with 1000 users logging in at the same time.

**2. Stress Testing**

* **Purpose**: To check how the system behaves under extreme or heavy load.
* **Goal**: Find the breaking point of the system.
* **Example**: Increase users from 1000 to 5000 to see when the site crashes.

**3. Spike Testing**

* **Purpose**: To check the system behavior when there is a sudden increase or decrease in load.
* **Goal**: See if system can handle sudden traffic changes.
* **Example**: Suddenly jump from 100 users to 2000 users for a few minutes.

**4. Endurance Testing (also called Soak Testing)**

* **Purpose**: To check how the system behaves under load for a long time.
* **Goal**: Identify memory leaks or performance degradation.
* **Example**: Keep 1000 users active for 12 hours and check if system slows down

**5. Volume Testing**

* **Purpose**: To check how the system handles a large amount of data.
* **Goal**: Test database or file handling capacity.
* **Example**: Upload 1 million records into the database and observe system behavior

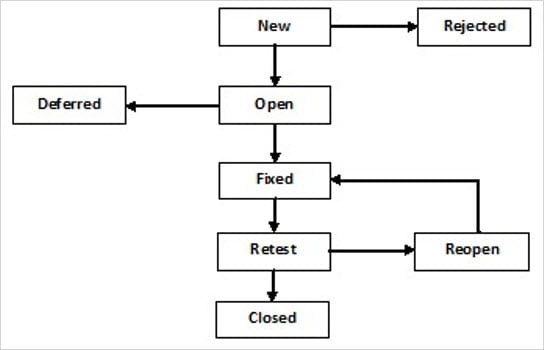
1. What is Error, Defect, Bug and failure ?

|  |
| --- |
|  |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Term** | **Meaning** | **Found By** | **Stage Found** | **Example** | | Error | A mistake made by a developer | Developer | During coding/design | Typing a = b + c instead of a = b - c | | Defect | A problem found during testing | Tester or Developer | During testing | Login button not working in testing | | Bug | A confirmed defect that needs fixing | Tester or Developer | During testing | Login issue reported, accepted by devs as a bug | | Failure | Software not working in real-world scenario | End User | After release (live) | App crashes when user tries to make a payment | |

1. Difference between Priority and Severity ?

|  |  |  |
| --- | --- | --- |
| SL.NO | **Severity** | **Priority** |
| 1. | Defined by the impact of a specific problem on any application’s functionality. | Defined by the impact on business. |
| 2. | Category decided by testers. | Category decided by developers or product owners. |
| 3. | Deals with the technical aspects of the application. | Deals with the timeframe or order to fix the defects. |
| 4. | The value does not change with time, it’s fixed. | The priority value is subjective and may change after comparing with other defects. |

1. What is Bug Life Cycle ?



**New** – A tester finds a bug and logs it.

**Assigned** – The bug is assigned to a developer for fixing.

**Open** – Developer starts working on the bug.

**Fixed** – Developer fixes the bug and updates the status.

**Retest** – Tester retests the bug to check if it is resolved.

**Verified** – If the bug is gone, tester marks it as verified.

**Closed** – Bug is fixed and confirmed, so it is closed

1. Explain the difference between Functional testing and Non Functional testing ?

**Functional Testing**

* **Definition**:  
  It checks whether the software functions as expected according to requirements.
* **Focus**:  
  What the system **does** (features and operations).
* **Examples**:  
  Login, Signup, Search, Payment, Form submission
* **Techniques used**:  
  Black-box testing, smoke testing, sanity testing, regression testing
* **Performed by**:  
  Testers

**Non-Functional Testing**

* **Definition**:  
  It checks how the software performs under certain conditions.
* **Focus**:  
  How the system **behaves** (performance, speed, security, usability)
* **Examples**:  
  Load testing, stress testing, security testing, compatibility testing
* **Techniques used**:  
  Performance testing tools like JMeter, LoadRunner
* **Performed by**:  
  Testers or performance/security specialists

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

**SDLC (Software Development Life Cycle)**

* **Definition**:  
  SDLC is the process followed for **developing** software from start to finish.
* **Focus**:  
  It focuses on **complete software development** – from requirement gathering to deployment and maintenance.
* **Phases include**:
  1. Requirement gathering
  2. Design
  3. Development
  4. Testing
  5. Deployment
  6. Maintenance
* **Participants**:  
  Business analysts, developers, testers, project managers

**STLC (Software Testing Life Cycle)**

* **Definition**:  
  STLC is the process followed to **test** the software and ensure it meets the quality standards.
* **Focus**:  
  It focuses only on **testing** activities within the SDLC.
* **Phases include**:
  1. Requirement analysis
  2. Test planning
  3. Test case design
  4. Test environment setup
  5. Test execution
  6. Test closure
* **Participants**:  
  Testers, QA leads, test managers

|  |  |  |
| --- | --- | --- |
| **Point** | **SDLC** | **STLC** |
| Focus | Complete software development | Only testing activities |
| Includes | Development, design, testing, deployment | Only testing phases |
| |  | | --- | |  |   Starts with | Requirement gathering | Requirement analysis (for testing) |
| Ends with | Maintenance | Test closure |

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

**Test Scenario**

Definition:  
A test scenario is a high-level idea of what needs to be tested.

Focus:  
It tells what to test.

Example:  
Verify the login functionality.

Used for:  
Quick understanding of the feature to test.

**Test Case**

Definition:  
A test case is a detailed document that includes test steps, input, expected result, and actual result.

Focus:  
It tells how to test.

Example:  
Step 1: Enter valid username  
Step 2: Enter valid password  
Step 3: Click on login  
Expected result: User should be redirected to the dashboard

**Test Script**

Definition:  
A test script is a set of instructions written in a programming or scripting language to automate test execution.

Focus:  
It is used in automated testing.

Example:  
Selenium script to test login functionality.

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

A **Test Plan** is a document that explains **how testing will be done** for a software project. It tells the team **what to test, how to test it, when to test it, and who will test it**. It helps everyone understand the testing process clearly

What Information Should Be Covered in a Test Plan?

**What to test** – The features or parts of the software that need testing.

**What not to test** – The parts that will not be tested.

**How to test** – The method and types of testing (like manual or automated).

**Who will test** – The names or roles of the people doing the testing.

**When testing will happen** – The time and schedule for testing activities.

**What tools will be used** – Like software or apps used for testing and bug tracking.

**What documents will be created** – Like test cases, bug reports, and final results.

**Start and end conditions** – When testing should begin and when it should stop.

**Risks and solutions** – What problems may come and how to handle them.

1. What is priority?

**Priority** in software testing refers to **how quickly** a bug should be **fixed and resolved**. It shows the **urgency** of the defect from the **business or user perspective**. High priority means the bug should be fixed **as soon as possible**, while low priority means the fix can be **delayed**.

**Who Sets Priority?**

Priority is usually set by:

* Testers
* Developers
* Project Managers
* Product Owners or Clients

They decide based on the **impact on the user**, **business needs**, and **release schedule**.

**Levels of Priority**

1. **High Priority**  
   Must be fixed immediately. It affects the main functions or user experience.
2. **Medium Priority**  
   Should be fixed, but not urgently. May not stop usage but needs attention.
3. **Low Priority**  
   Can be fixed later. Has little or no impact on current functionality.

**Example:**

1. If the **submit button on the login page is not working**, it is a high priority bug because users cannot log in.
2. If the **font size on the About Us page is small**, it may be low priority because it does not affect major features.

**Priority vs Severity (Simple Difference):**

* **Priority**: Business urgency of fixing the bug.
* **Severity**: Technical impact of the bug on the system.

Example:  
A spelling mistake in the app name = **High Priority, Low Severity**  
System crash on login = **High Severity, High Priority**

1. What is severity ?

Definition  
Severity shows how serious a bug is from a technical point of view. It tells how much the defect affects the working of the software.

Decided By  
Severity is decided by the tester or QA team based on the impact of the bug on the system.

Purpose  
It helps the team understand which bugs are more serious and need urgent technical attention.

Levels of Severity  
Critical – System crashes or major functionality is broken  
High – Important feature is not working but the system is running  
Medium – Some part is not working, but there is a workaround  
Low – Minor issue that does not affect main functions, like spelling or layout problems

Example  
If the login button crashes the app, it is critical severity.  
If there is a spelling mistake on a page, it is low severity.

1. Bug categories are ?

1. Functional Bug  
This type of bug occurs when the software does not work as expected based on requirements.  
Example – Login button does not submit the form.

2. Performance Bug  
These bugs are related to the speed, response time, or stability of the software.  
Example – Page takes too long to load.

3. Usability Bug  
A usability bug makes the software difficult or confusing for the user to use.  
Example – Button placement is not user-friendly.

4. Compatibility Bug  
These bugs appear when the software does not work properly on different devices, browsers, or operating systems.  
Example – Website works on Chrome but not on Firefox.

5. Security Bug  
These are related to security issues, where unauthorized access or data leakage can occur.  
Example – User can access admin panel without permission.

6. UI Bug (User Interface Bug)  
This bug is related to design, layout, alignment, font, or color problems.  
Example – Text is not aligned properly on the page.

7. Logical Bug  
Occurs when the logic of a feature or calculation is wrong.  
Example – Total price is calculated incorrectly in a shopping cart.

8. Crash Bug  
This bug causes the application to crash or close unexpectedly.  
Example – App crashes when clicking on a menu option.

1. **Advantage of Bugzila ?**

Open-source and free to use

Easy to track and manage bugs

Supports email notifications

Allows setting bug priority and severity

Good for team collaboration

Provides detailed bug reports

Supports custom workflows

Offers advanced search and filter options

Can integrate with other tools

Secure and reliable for large projects

1. Difference between priority and severity ?

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Severity** | **Priority** |
| **Definition** | |  | | --- | | **Measures the impact of a defect on the software's functionality.** |  |  | | --- | |  | | |  | | --- | | **Determines the urgency with which a defect should be fixed.** |  |  | | --- | |  | |
| |  | | --- | | **Determined by** |  |  | | --- | |  | | |  | | --- | | **Typically assessed by testers or QA engineers based on the defect's technical impact.** |  |  | | --- | |  | | |  | | --- | | **Usually set by project managers or clients, considering business needs and timelines.** |  |  | | --- | |  | |
| |  | | --- | | **Focus Area** |  |  | | --- | |  | | |  | | --- | | **Concentrates on the technical aspects and how the defect affects system operations.** |  |  | | --- | |  | | |  | | --- | | **Focuses on the business implications and the need for a prompt resolution.** |  |  | | --- | |  | |
| |  | | --- | | **Categories/Levels** |  |  | | --- | |  | | |  | | --- | | * **Critical: Complete system failure or crash.<br>- Major: Significant functionality impairment.<br>- Moderate: Partial loss of functionality with workarounds.<br>- Minor: Cosmetic or trivial issues.** |  |  | | --- | |  | | |  | | --- | | * **High: Immediate attention required; defect must be fixed as soon as possible.<br>- Medium: Should be fixed before the product release.<br>- Low: Fixing can be deferred; does not significantly impact business operations.** |  |  | | --- | |  | |
| **Changeability** | |  | | --- | | **Generally remains consistent unless the defect's impact is reassessed.** |  |  | | --- | |  | | |  | | --- | | **Can change over time based on shifting business priorities or project timelines.** |  |  | | --- | |  | |
| **Example** | |  | | --- | | **A system crash when submitting a form would be considered high severity because it affects core functionality.** |  |  | | --- | |  | | **A typo in the company name on the homepage would be high priority due to its impact on brand image, even though it's a minor issue.** |

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

In Agile development, several **methodologies** or **frameworks** have been developed based on the core principles of the Agile Manifesto. Each has its own processes and practices but shares the Agile values of **iterative development**, **customer collaboration**, and **responding to change**.

Here are some of the most popular **Agile methodologies**:

**1. Scrum**

* **Most widely used Agile methodology**
* Organizes work into **sprints** (usually 2–4 weeks)
* Roles: **Product Owner**, **Scrum Master**, and **Development Team**
* Events: **Sprint Planning**, **Daily Standup (Scrum)**, **Sprint Review**, **Sprint Retrospective**
* Focus: Transparency, inspection, and adaptation

**2. Kanban**

* Focuses on **visualizing the workflow** using a **Kanban board**
* Limits work in progress (WIP) to avoid overloading the team
* No fixed iterations or roles
* Good for continuous delivery and support teams

**3. Extreme Programming (XP)**

* Emphasizes **technical practices** and **developer discipline**
* Core practices: **Test-Driven Development (TDD)**, **Pair Programming**, **Continuous Integration**, **Refactoring**, **Small Releases**
* Focuses on **code quality** and **rapid feedback**

**4. Lean Software Development**

* Originated from Lean manufacturing (Toyota)
* Focus: **Eliminate waste**, **amplify learning**, **deliver fast**
* Encourages **empowered teams**, **deciding as late as possible**, and **optimizing the whole**

**5. Crystal**

* Family of lightweight Agile methods (Crystal Clear, Crystal Yellow, Crystal Orange, etc.)
* Tailored based on team size and project criticality
* Values **people**, **interactions**, and **communication** over rigid processes

**6. Feature-Driven Development (FDD)**

* Focuses on building & designing features
* Follows five processes: **Develop Overall Model**, **Build Feature List**, **Plan by Feature**, **Design by Feature**, and **Build by Feature**
* More structured, suitable for larger teams

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

* One of the earliest Agile methods
* Uses an iterative and incremental approach
* Principles: **Timeboxing**, **MoSCoW prioritization** (Must have, Should have, Could have, Won’t have), active user involvement

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

* A simplified version of Rational Unified Process (RUP)
* Embraces Agile principles
* Phases: **Inception**, **Elaboration**, **Construction**, and **Transition.**

1. **What istraceability matrix?**

A **Traceability Matrix** (often called **Requirement Traceability Matrix** or **RTM**) is a **document** used in software testing and project management to ensure that **all requirements are covered by test cases**.

**🔍 Definition:**

The Traceability Matrix is a table that maps and traces **user requirements** with **test cases**, ensuring that each requirement has been tested and verified.

**🧩 Purpose of Traceability Matrix:**

* To ensure **100% test coverage**
* To identify any **missing test cases** for a requirement
* To track the **requirements throughout the project lifecycle**
* To ensure **nothing is missed** during testing

**📊 Typical Columns in a Traceability Matrix:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Requirement ID** | **Requirement Description** | **Test Case ID** | **Test Case Description** | **Status (Pass/Fail)** | | REQ-01 | User should be able to log in | TC-01 | Verify valid login | Pass | | REQ-02 | Password must be encrypted | TC-05 | Check password encryption | Fail | |

**🧷 Types of Traceability:**

1. **Forward Traceability**  
   Tracing requirements → test cases  
   (Are all requirements tested?)
2. **Backward Traceability**  
   Tracing test cases → requirements  
   (Does each test case map to a requirement?)
3. **Bi-directional Traceability**  
   Both forward and backward tracing  
   (Full verification and validation)

**🧠 Why It's Important:**

* Helps in **impact analysis** when changes occur
* Ensures **compliance** in regulated industries (like healthcare, finance)
* Enhances **quality and completeness** of testing

1. **What is Equivalence partitioning testing**

**Equivalence Partitioning Testing** (also called **Equivalence Class Partitioning**) is a **black-box testing technique** used to reduce the number of test cases by dividing input data into **equivalence classes**.

**🔍 Definition:**

Equivalence partitioning is a method where input values are **divided into groups (partitions)** that are expected to be treated the **same** by the system. You just need to test **one value** from each group, assuming that if one works, the others will too.

**📦 Why Use It?**

* To **reduce the number of test cases**
* To **avoid redundancy**
* To **ensure better coverage** with fewer tests

**🧠 Key Points:**

* Works well when input is a range or set
* Helps in **early defect detection**
* Commonly used in combination with **Boundary Value Analysis**

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

**The level of risk in software testing (or in general risk management) is determined by evaluating two key factors:**

**🔐 1. Probability (Likelihood)**

**How likely is it that the risk will occur?**

* **High: Very likely to happen**
* **Medium: Might happen**
* **Low: Unlikely to happen**

**💣 2. Impact (Severity/Consequence)**

**What would be the effect or damage if the risk actually happens?**

* **High: Major failure, data loss, safety issue**
* **Medium: Some features broken, minor delays**
* **Low: Cosmetic issue, low inconvenience**

**🛠️ How is Risk Level Used?**

* **To prioritize testing efforts**
* **To decide resource allocation**
* **To plan mitigation and contingency**

1. **When to used Usablity Testing?**

Usability Testing is used when you want to check how easy and user-friendly your application or product is for real users. It's all about understanding the user experience (UX) — how people interact with your software.

🎯 When to Use Usability Testing:

1. During the Design Phase

* To validate early prototypes or wireframes
* Catch UX issues before development begins

2. Before Product Release (Pre-Launch)

* Ensure the product is intuitive and easy to use
* Confirm that the user can complete key tasks without confusion

3. After Major UI/UX Changes

* When a major redesign or feature update is done
* To see if the new design improves or worsens usability

4. When Users Are Facing Problems

* If users complain about difficulty or confusion
* To identify pain points and areas of frustration

5. When Introducing New Features

* To test if the new feature is discoverable and usable
* Make sure it fits naturally into the existing design

🧪 What Does It Check?

* Ease of navigation
* Clarity of content
* User satisfaction
* Task completion rate
* Errors made by users
* Time taken to complete tasks

👥 How is it Done?

* Real users perform specific tasks while observers watch
* Feedback is gathered via observation, interviews, or screen recordings

🛠️ Example Scenario:

You're building a food delivery app. Before launch, you do usability testing to see:

* Can users easily find and order food?
* Is the checkout process smooth?
* Can they track delivery easily?

If users get stuck, you go back and improve the design.

1. What is the procedure for GUI Testing?

**✅ 1. Review GUI Requirements**

* Understand the **design specifications**, **UI mockups**, or **wireframes**
* Know what the **expected layout, behavior, and style** of UI elements should be

**📝 2. Identify GUI Components to Test**

* List all interface elements such as:
  + Buttons, text boxes, dropdowns, checkboxes
  + Menus, icons, links, labels, dialog boxes
  + Forms, tables, pop-ups, tooltips

**📄 3. Write GUI Test Cases**

* Design test cases for:
  + **Functionality** (e.g., button click works)
  + **Appearance** (e.g., font size, color, alignment)
  + **Layout** (e.g., components are properly placed)
  + **Behavior** (e.g., error messages, hover effects)

**🛠️ 4. Set Up Test Environment**

* Install the application or launch the web/mobile app
* Use different screen sizes and browsers (if applicable)

**🧪 5. Execute GUI Tests**

* **Manually** or with **automation tools** (like Selenium, TestComplete, Katalon)
* Check for:
  + Visual consistency
  + Responsive layout
  + Element alignment
  + Navigation and usability
  + Field validation and behavior

**📸 6. Log Defects (if any)**

* Record test results and take **screenshots of issues**
* Report bugs using a **bug tracking tool** (Jira, Bugzilla, etc.)

**🔁 7. Retesting and Regression Testing**

* After fixes, **retest** failed cases
* Run a **regression test** to ensure nothing else broke

**📋 8. Sign Off**

* When all GUI elements work as expected and are visually correct, sign off the GUI testing

**⚠️ Common Issues Found in GUI Testing:**

* Misaligned buttons or labels
* Inconsistent font size or color
* Broken images or icons
* Overlapping text on smaller screens
* Non-responsive or disabled controls