**Software Testing Assignment**

**Module: -2 [Manual Testing]**

**1: - What is Exploratory Testing?**

Exploratory testing is a software testing approach where testers simultaneously design, execute, and learn about the software under test.

**2: - What is traceability matrix?**

A requirements traceability matrix is a document that demonstrates the relationship between requirements and other artifacts.

**3: - What is Boundary value testing?**

Boundary value testing is a software testing technique where test cases are designed to check the behaviour of a system at the boundaries of input domains.

**4: - What is Equivalence partitioning testing?**

Equivalence Partitioning Testing is a software testing technique where input values are divided into groups (partitions), and only one value from each group is tested because all values in the same group are expected to behave the same way.

**5: - What is Integration testing?**

Integration Testing is a type of software testing where individual modules or components of a system are combined and tested as a group to check if they work together correctly.

**6: - What determines the level of risk?**

The level of risk in software testing (or project management in general) is determined by two main factors:

1. Probability (Likelihood) → How likely is the risk to occur?
2. Impact (Consequence/Severity) → What is the effect or damage if the risk occurs?

**7: - What is Alpha testing?**

Alpha Testing is a type of acceptance testing done by the internal team (developers + testers) before releasing the product to real users.

**8: - What is beta testing?**

Beta Testing is a type of acceptance testing where the software is released to a limited number of real users outside the company to use in their real environment and give feedback.

**9: - What is component testing?**

Component Testing is testing of individual parts of the software separately to make sure each one works as expected.

**10: - What is functional system testing?**

Functional System Testing is a type of system testing where the complete software system is tested against its functional requirements to make sure it behaves as expected.

**11: - What is Non-Functional Testing?**

Non-Functional Testing (NFT) is a type of software testing that checks the non-functional aspects of a system, such as performance, speed, security, usability, reliability, and scalability, rather than what the system does.

**12: - What is GUI Testing?**

GUI Testing (Graphical User Interface Testing) is a type of software testing that checks whether the user interface (UI) of an application works correctly and looks as expected.

**13: - What is Adhoc testing?**

Adhoc Testing is an informal type of software testing where testers try to find defects in the system without any predefined test cases or plans. It relies on the tester’s intuition, experience, and creativity.

**14: - What is load testing?**

Load Testing is a type of performance testing where the system is tested by applying a specific amount of expected user load to see how it behaves under normal and peak conditions.

**15: - What is stress Testing?**

Stress Testing is a type of performance testing where the system is tested under extreme or beyond normal load conditions to check how it behaves when pushed to its limits.

**16: -** **What is white box testing and list the types of white box testing?**

White Box Testing is a software testing technique where the tester looks into the internal structure, design, and code of the software to design test cases.

* **Types of White Box Testing:**

1. **Unit Testing** → Testing individual functions or methods in the code.
2. **Integration Testing (white-box style)** → Checking how different code modules interact internally.
3. **Code Coverage Testing** (to measure how much of the code is tested):
   * **Statement Coverage** → Every line of code executes at least once.
   * **Branch Coverage (Decision Coverage)** → Every possible decision (true/false) is tested.
   * **Path Coverage** → Every possible execution path is tested.
   * **Condition Coverage** → Tests all Boolean expressions (true/false outcomes).
4. **Control Flow Testing** → Checks loops, conditions, and execution order in the code.
5. **Data Flow Testing** → Checks how data values are defined, used, and modified in the code.
6. **Mutation Testing** → Making small changes in the code (mutants) to see if test cases detect them.

**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 checks the functionality of the software without knowing its internal code or structure.

* **Black Box Testing Techniques:**

1. **Equivalence Partitioning (EP)**
   * Divide input data into groups (valid/invalid) and test one value from each group.
   * Example: Age field (1–120) → Valid (25), Invalid (-5, 150).
2. **Boundary Value Analysis (BVA)**

* Test at the boundaries (minimum, maximum, just inside, just outside).
* Example: Age 1–120 → Test 0, 1, 120, 121.

1. **Decision Table Testing**
   * Useful for systems with **multiple conditions** and **different outcomes**.
   * Example: Loan approval depends on income + credit score.
2. **State Transition Testing**
   * Tests how the system behaves when it moves from one **state** to another.
   * Example: ATM → (Card Inserted → PIN Entered → Withdraw Cash).
3. **Use Case Testing**
   * Derived from **use cases** → tests end-to-end scenarios from a user’s perspective.
   * Example: "Place an order" use case in an e-commerce app.
4. **Error Guessing** (experience-based)
   * Testers use their **intuition and past experience** to guess where defects might occur.
   * Example: Entering invalid characters in a numeric field.

**18: -** **Mention what are the categories of defects?**

* **Main Categories of Defects:**

1. **By Severity (Impact on the system):**
   * Critical Defect → Causes system crash or major failure (e.g., payment not processed).
   * Major Defect → Main functionality fails but system still runs (e.g., login not working).
   * Minor Defect → Small functionality issue (e.g., search filter not working properly).
   * Trivial/Low Defect → Cosmetic issues (e.g., spelling mistake, misaligned button).
2. **By Nature, /Type:**
   * Functional Defects → When the software doesn’t behave as per requirements (e.g., wrong calculations).
   * Performance Defects → System is slow or cannot handle expected load.
   * Usability Defects → Poor user experience (e.g., confusing navigation).
   * Compatibility Defects → Works on one browser/device but not on another.
   * Security Defects → Vulnerabilities like weak authentication or data leaks.
   * Integration Defects → Issues when modules interact (e.g., wrong data passed).
3. **By Priority (Business urgency):**
   * High Priority → Needs to be fixed immediately (e.g., system crash on login).
   * Medium Priority → Important but can wait until the next release.
   * Low Priority → Minor issue, fix can be delayed.

**19: -** **Mention what big bang testing is?**

Big Bang Testing is an integration testing approach where all modules are combined together at once and then tested as a complete system, instead of testing them step by step.

**20: -** **What is the purpose of exit criteria?**

Exit Criteria are the set of conditions that must be met before testing can be considered complete and the software can move to the next phase (or be released).

**21: - When should "Regression Testing" be performed?**

**Regression Testing is done to ensure that new changes (bug fixes, updates, or new features) have not broken the existing functionality of the software.**

* **When should Regression Testing be performed?**

1. After fixing defects → To confirm the fix didn’t affect other parts of the system.
2. After adding new features → To ensure existing features still work properly.
3. After code changes/enhancements → When developers modify the codebase.
4. After performance improvements or configuration changes → To check stability.
5. During maintenance releases/patches → To verify nothing else got impacted.
6. Before major releases → As part of final testing cycle to ensure system stability

**22: -** **What is 7 key principles? Explain in detail?**

**1. Testing shows the presence of defects, not their absence**

* Testing helps us find defects, but it cannot guarantee that the software is 100% defect-free.
* Even after extensive testing, some hidden bugs may remain.

**2. Exhaustive testing is impossible**

* It is not possible to test **all possible inputs, paths, and scenarios** because they are infinite.
* Instead, testers use **techniques (like equivalence partitioning, boundary value analysis)** to test a selected set of meaningful cases.

**3. Testing should start as early as possible**

* Defects found **early in the software life cycle** (requirements/design stage) are cheaper and easier to fix than those found later.
* Early testing prevents defect propagation.

**4. Defects cluster together**

* In most projects, a small number of modules contain the **majority of defects**.
* Testers can focus more effort on these **high-risk or complex areas**.

**5. Testing is context dependent**

* Testing is done differently depending on the **type of software** and its use.
* Safety-critical software (like medical systems) requires more rigorous testing than a simple website.

**6. Absence-of-errors fallacy**

* Just because the software has **no known defects**, it doesn’t mean it is useful or meets customer needs.
* The software must be tested against **requirements and user expectations**, not just for bugs.

**7. Testing is a creative and intellectual activity**

* Testing is not just about following steps — it needs **creativity, critical thinking, and experience**.
* Testers design scenarios, anticipate risks, and think like end users.

**23: -** **Difference between QA v/s QC v/s Tester**

|  |  |  |  |
| --- | --- | --- | --- |
| **Aspect** | **Quality Assurance (QA)** | **Quality Control (QC)** | **Tester** |
| **Definition** | A **process-oriented** activity that ensures quality in the processes used to develop software. | A **product-oriented** activity that ensures the actual product meets the quality requirements. | A **person/role** who performs testing activities to find defects in the software. |
| **Focus** | **Prevention** of defects (making sure processes are correct). | **Detection** of defects (finding issues in the product). | **Execution** of testing (running tests and reporting bugs). |
| **Goal** | To improve and define development and testing processes. | To verify if the developed product is correct and meets requirements. | To identify defects by executing test cases or exploratory testing. |
| **Activities** | - Process definition |  |  |

**24: -** **Difference between Smoke and Sanity?**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Smoke Testing** | **Sanity Testing** |
| **Definition** | A shallow and wide testing to check whether the basic functionalities of the application are working or not. | A narrow and deep testing to verify that specific functionality or bug fix works correctly. |
| **Purpose** | To check the stability of the build before going for detailed testing. | To check the rationality of changes/bug fixes in the build. |
| **When performed?** | Done on initial builds to ensure the application is ready for testing. | Done on new builds/patches after bug fixes or minor changes. |
| **Scope** | Broad but shallow → tests all major modules in brief. | Focused and detailed → tests only affected areas. |
| **Performed by** | Testers (sometimes developers). | Testers. |
| **Automation** | Usually automated (quick checks). | Usually manual (focused checks). |
| **Example** | In an e-commerce app: Checking if the app launches, login works, products can be searched, and cart opens. | After fixing a payment bug, only testing the payment module to confirm it works correctly. |

**25: -** **Difference between verification and Validation**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Verification** | **Validation** |
| **Meaning** | Ensures the software is built correctly (follows process, design, requirements). | Ensures the right product is built (meets user needs). |
| **Focus** | Process-oriented | Product-oriented |
| **Stage** | Early stage (reviews, design, requirement checks) | Later stage (testing the actual product) |
| **Type** | Static (without executing code) | Dynamic (executing code) |
| **Performed by** | QA team, developers, analysts | Testers, end users |
| **Example** | Checking if the login requirement document is correct and complete. | Actually, logging into the system to see if it works as expected. |

**26: -** **Explain types of Performance testing.**

1. **Load Testing**
   * Checks how the system performs under expected load.
   * Ensures it can handle the normal number of users/transactions.
   * *Example:* 1,000 users logging into a shopping app at the same time.
2. **Stress Testing**
   * Tests the system under extreme or beyond normal load.
   * Finds the breaking point (when the system fails).
   * *Example:* Forcing 20,000 users on a system that supports 5,000.
3. **Spike Testing**
   * Tests how the system reacts to a sudden increase or decrease in load.
   * Checks stability during unexpected traffic spikes.
   * *Example:* Ticket booking site during a flash sale.
4. **Endurance (Soak) Testing**
   * Checks if the system can handle a normal load for a long period without performance issues like memory leaks or slowdowns.
   * *Example:* Running an e-commerce site with 500 users continuously for 72 hours.
5. **Scalability Testing**
   * Tests the system’s ability to scale up or down when load increases.
   * Ensures the system can handle growth.
   * *Example:* Adding more users gradually to see if servers can scale horizontally.
6. **Volume Testing (Flood Testing)**
   * Checks how the system handles a large volume of data.
   * Focuses on database performance.
   * *Example:* Uploading 1 million records into the database and checking response time.

**27: -** **What is Error, Defect, Bug and failure?**

**1. Error (Mistake)**

* A human mistake made by a developer, tester, or analyst while coding, designing, or understanding requirements.

**2. Defect (Fault)**

* A **flaw in the software** that is introduced because of an error in coding or design.
* Found during **development or testing**.

**3. Bug**

* A **defect reported by testers or users** during testing or after release.
* In practice, “defect” and “bug” are often used interchangeably.

**4. Failure**

* When the **software does not perform as expected during execution** in real use.
* Happens when a defect is **actually triggered** in operation.

**28: -** **Difference between Priority and Severity**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Severity** | **Priority** |
| **Meaning** | How serious the defect is → impact on system functionality. | How urgently the defect should be fixed → business importance. |
| **Focus** | Technical impact on the system. | Business impact / customer needs. |
| **Decided by** | Testers (based on functionality). | Product manager / client (based on business needs). |
| **Types** | - Critical |  |

**29: -** **What is Bug Life Cycle?**

1. **New** 
   * When a tester finds a defect for the first time and reports it.
2. **Assigned** 
   * The bug is assigned to a developer for fixing.
3. **Open** 
   * The developer starts analysing and working on the bug.
4. **Fixed/Resolved** 
   * The developer makes code changes and marks the bug as fixed.
5. **Retest** 
   * Tester checks the application again to verify if the bug is really fixed.
6. **Closed** 
   * If the bug is fixed successfully and working fine, the tester marks it as closed.

**30: -** **Explain the difference between Functional testing and Non-functional testing**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Functional Testing** | **Non-Functional Testing** |
| **Definition** | Verifies *what the system does* (features & functions). | Verifies *how the system performs* (quality attributes). |
| **Focus** | Business requirements (correctness of functions). | Performance, usability, reliability, scalability, security, etc. |
| **Objective** | Ensure the software behaves as expected. | Ensure the software meets quality standards. |
| **Checks** | Specific actions and outputs (e.g., login, signup, search). | Speed, load handling, responsiveness, user experience. |
| **Question Answered** | “Does the system do what it is supposed to do?” | “How well does the system work under conditions?” |
| **Types** | Unit, Integration, System, UAT, Smoke, Sanity. | Performance, Load, Stress, Security, Usability, Reliability. |
| **Execution** | Usually, manual + automation scripts. | Mostly tool-based (e.g., JMeter, LoadRunner, Security scanners). |
| **Example** | Test if “Add to Cart” works correctly. | Test how fast “Add to Cart” works when 1,000 users do it at once. |

**31: -** **What is the difference between the STLC (Software Testing Life Cycle) and SDLC (Software Development Life Cycle)?**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **SDLC (Software Development Life Cycle)** | **STLC (Software Testing Life Cycle)** |
| **Definition** | Process followed for developing software (end-to-end). | Process followed for testing software (ensuring quality). |
| **Focus** | Covers the *entire development* of software: planning → building → deploying → maintaining. | Covers the *entire testing* process: planning → designing → executing → closing. |
| **Objective** | Deliver a fully functional software system. | Ensure the software meets quality standards and is defect-free. |
| **Phases** | 1. Requirement Analysis 2. Planning 3. Design 4. Development 5. Testing 6. Deployment & Maintenance | 1. Requirement Analysis 2. Test Planning 3. Test Case Design 4. Test Environment Setup 5. Test Execution 6. Test Cycle Closure |
| **Performed By** | Developers, Business Analysts, Project Managers, Testers, etc. | Testers / QA team. |
| **Output** | A working software product. | A quality-assured, tested product + test reports. |
| **End Goal** | Software that meets customer/business needs. | Defect-free software with verified functionality & performance. |

**32: - What is the difference between test scenarios, test cases, and test script?**

|  |  |  |  |
| --- | --- | --- | --- |
| **Aspect** | **Test Scenario** | **Test Case** | **Test Script** |
| **Definition** | High-level functionality to be tested | Step-by-step instructions with inputs & expected results | Automated/manual implementation of test case |
| **Detail Level** | Broad, abstract | Detailed & structured | Executable (manual steps or code) |
| **Purpose** | Defines *what* to test | Defines *how* to test | Actually, *executes* the test |
| **Audience** | BA, Dev, QA | QA Testers | QA/Automation Engineers |
| **Example** | Test login functionality | Enter valid credentials and login | Selenium script for login |

**33: -** **Explain what Test Plan is? What is the information that should be covered.**

A **Test Plan** is a formal document that describes the **scope, approach, resources, schedule, and activities** of testing for a software project

* **Information Covered in a Test Plan**

**1. Test Plan Identifier**

* A unique ID or version for the test plan document (e.g., *TP\_LoginModule\_V1.0*).

**2. Introduction**

* Overview of the project/module to be tested.
* Objectives of testing.
* Scope (what’s included) and **Out of Scope** (what’s excluded).

**3. Test Items**

* The features, modules, or systems that need to be tested.
* Example: Login, Payment Gateway, Cart Module, etc.

**4. Features to be Tested**

* A list of specific functionalities and requirements to validate.

**5. Features NOT to be Tested**

* Areas explicitly excluded from testing (due to time, dependency, or non-relevance)

**6. Test Approach / Strategy**

* High-level approach of testing:
  + Manual vs. Automation
  + Types of testing: Functional, Regression, Performance, Security, UAT, etc.
  + Testing levels: Unit, Integration, System, Acceptance.

**7. Test Environment**

* Details about hardware, software, network, test tools, and environment setup.
* Example: OS version, browsers, database, test data, staging server details.

**8. Test Deliverables**

* What documents/artifacts will be produced:
  + Test Scenarios
  + Test Cases
  + Test Scripts
  + Test Data
  + Test Summary Reports

**9. Roles & Responsibilities**

* Who will do what? (Test Manager, QA Engineer, Automation Engineer, Dev Support, etc.)

**10. Schedule / Timeline**

* Testing phases with start and end dates.
* Example: Test Case Design – 1 week, Execution – 2 weeks, Regression – 3 days.

**11. Entry and Exit Criteria**

* **Entry Criteria:** What conditions must be met before testing begins? (e.g., build deployed, test environment ready, test data prepared).
* **Exit Criteria:** What conditions signify testing is complete? (e.g., all high-severity bugs fixed, 95% test cases executed, no blocker defects).

**12. Suspension & Resumption Criteria**

* When testing should be stopped (e.g., too many blocker bugs, environment not stable).

**13. Risks & Mitigation**

* Possible risks:
  + Delay in environment setup
  + Dependency on third-party APIs
  + Limited testing resources
* Mitigation strategies to reduce risks.

**14. Tools**

* Any testing tools used:
  + Test Management: JIRA, TestRail, Zephyr
  + Automation: Selenium, Cypress, JUnit
  + Performance: JMeter, LoadRunner

**15. Approval**

* Sign-off by project stakeholders (QA Lead, Project Manager, Client).

**34: -** **What is priority?**

|  |  |  |
| --- | --- | --- |
| **Priority Level** | **Meaning** | **Example** |
| **High (P1)** | Must be fixed immediately; blocks business-critical functions or release. | Login button not working, payment gateway failure, app crash on startup. |
| **Medium (P2)** | Important but not urgent; can be fixed in the next build/release. | "Forgot Password" link not working, minor performance delay, search filter issue. |
| **Low (P3)** | Nice-to-have fix; does not impact business flow; can be delayed. | Typo in UI text, misaligned icons, cosmetic colour mismatch. |

**35: -** **What is severity?**

|  |  |  |
| --- | --- | --- |
| **Severity Level** | **Meaning** | **Example** |
| **Critical (S1)** | Causes complete failure of the system or a major module; no workaround available. | Application crash after login, system not starting, data loss. |
| **Major (S2)** | Severely impacts functionality, but system is still partially usable; workaround may exist. | Checkout not working, form not saving, broken API integration. |
| **Minor (S3)** | Defect causes some inconvenience, but core functionality works; easy workaround available. | Error message not displayed correctly, misaligned UI component. |
| **Trivial (S4)** | Very small issue, cosmetic or negligible impact on functionality. | Typo in label, font size mismatch, extra space in text. |

**36: -** **Bug categories are…**

|  |  |  |
| --- | --- | --- |
| **Bug Category** | **Description** | **Example** |
| **Functional** | Feature doesn’t work | Login button not working |
| **Performance** | Speed/response issues | Page load takes 10s |
| **Usability** | User experience issues | Confusing navigation |
| **Compatibility** | Cross-device/OS issues | Works in Chrome, not in Safari |
| **Security** | Vulnerabilities | SQL Injection possible |
| **UI/Cosmetic** | Look & feel issues | Misaligned icons |
| **Logical** | Business logic errors | Wrong discount calculation |
| **Data Integrity** | Data-related issues | Corrupted/duplicate data |
| **Crash/Blocker** | System crash/blocking | App crashes on startup |

**37: -** **Advantage of Bugzilla.**

Bugzilla is one of the oldest and most widely used open-source defect tracking systems. Its main advantages are:

* **1. Open Source & Free**
* Bugzilla is **free to use** under the Mozilla Public License.
* No licensing cost compared to paid tools like JIRA or HP ALM.
* **2. Web-Based Interface**
* Works on any OS (Windows, Linux, macOS) via a browser.
* No need to install heavy desktop clients.
* **3. Customizable Workflow**
* Highly flexible and supports **custom bug life cycles, fields, and permissions**.
* Organizations can tailor it to their testing process.
* **4. Advanced Search & Reporting**
* Strong **search filters** to find bugs by ID, status, priority, severity, assignee, etc.
* Generates reports, graphs, and bug statistics for management.
* **5. Email Notifications**
* Sends automated email alerts to team members when bugs are reported, updated, or resolved.
* **6. Integration Support**
* Can integrate with **version control systems** (like Git, CVS, SVN) and other tools.
* Supports plugins for CI/CD pipelines.
* **7. Role-Based Access**
* Different access levels (Admin, Developer, Tester, Reporter, Guest).
* Helps control who can view, report, or edit bugs.
* **8. Scalability**
* Can handle **thousands of bugs and users** efficiently, suitable for large projects.
* **9. Audit & History Tracking**
* Maintains a **complete history of bug changes** (status, comments, updates).
* Helps in accountability and traceability.
* **10. Multi-Project Support**
* Can manage bugs for **multiple projects simultaneously** in a single installation

**38: -** **Difference between priority and severity**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Priority** | **Severity** |
| **Definition** | Defines how urgently a bug should be fixed (business urgency). | Defines how serious the bug’s impact is on the system (technical impact). |
| **Focus** | Business perspective → *When to fix?* | Technical perspective → *How bad is it?* |
| **Set By** | Product Manager / Project Manager / Business Analyst (sometimes QA Lead) | Testers / QA Team |
| **Levels** | High, Medium, Low (P1, P2, P3) | Critical, Major, Minor, Trivial (S1–S4) |
| **Time Factor** | Decides the order of fixing (fix immediately or later). | Decides the extent of damage to the system. |
| **Example 1** | Company logo missing on homepage → High Priority (urgent for branding), but Low Severity (doesn’t brake system). | Login button not working → High Severity (functionality blocked), also High Priority (must fix immediately). |
| **Example 2** | Typo in footer text → Low Priority (not urgent), Low Severity (small issue). | App crashes only when entering 15-digit phone number → High Severity (crash), but Low Priority (rare scenario). |

* **Priority = When to fix (business urgency)**
* **Severity = How bad is it (technical seriousness)**

**39: -** **What are the different Methodologies in Agile Development Model?**

|  |  |
| --- | --- |
| **Methodology** | **Key Focus** |
| **Scrum** | Sprint-based, roles & ceremonies |
| **Kanban** | Visual workflow, continuous delivery |
| **Extreme Programming (XP)** | Engineering practices, TDD, CI |
| **Lean** | Eliminate waste, deliver value |
| **Crystal** | People and communication |
| **DSDM** | Full lifecycle, business needs |
| **FDD** | Feature-based, good for large teams |
| **AUP** | Agile + Rational Unified Process |

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

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Authentication** | **Authorization** |
| **Definition** | Process of verifying the identity of a user (Who are you?). | Process of verifying what resources/permissions a user has (What can you do?). |
| **Purpose** | Ensures only legitimate users can log in. | Ensures logged-in users can access only what they are allowed. |
| **Checks** | Username, password, OTP, biometric, captcha. | Roles, permissions, access levels (Admin, User, Guest). |
| **Example** | Entering correct username & password to log in. | Admin can delete users, but a normal user cannot. |
| **When performed** | First step – before granting access. | After authentication – once user is logged in. |

* **Authentication = Login (identity check)**
* **Authorization = Permissions (access rights check)**
* **Common Problems Faced in Web Testing**

When testing web applications, testers often face challenges such as:

**1. Functional Issues**

* Broken links.
* Incorrect form validations.
* Session timeouts not working.
* Shopping cart not updating correctly.

**2. Usability Issues**

* Poor navigation flow.
* Unclear error messages.
* Inconsistent UI design across pages.
* Accessibility issues (not usable with screen readers).

**3. Compatibility Issues**

* Browser compatibility (Chrome vs Firefox vs Safari).
* OS compatibility (Windows, Linux, macOS).
* Device compatibility (desktop, tablet, mobile).

**4. Performance Issues**

* Slow page loading times.
* Server crashes under heavy traffic.
* Memory leaks or bottlenecks.

**5. Security Issues**

* Weak authentication (e.g., no 2FA).
* Improper authorization (users accessing restricted areas).
* SQL Injection, XSS (Cross-Site Scripting), CSRF attacks.
* Session hijacking or insecure cookies.

**6. Localization & Globalization Issues**

* Incorrect language translations.
* Date, time, currency format mismatches.
* Text overlapping in different languages.

**7. Data Integrity Issues**

* Data not saved correctly in DB.
* Duplicate or missing records.
* Inconsistent data when multiple users access simultaneously.

**41: -** **When to used Usability Testing?**

* **Early →** to validate design (prototypes, wireframes).
* **Before releases →** to ensure smooth user experience.
* **On new features or user groups →** to confirm adoption.
* **When issues are suspected →** to find pain points.
* **For accessibility checks →** to make product inclusive.

**42: -** **What is the procedure for GUI Testing?**

|  |  |  |
| --- | --- | --- |
| **Step** | **Description** | **Example** |
| **1. Requirement Analysis** | Understand UI/UX requirements, design mock-ups, and specifications. | Login page should have Username, Password fields, and Login button. |
| **2. Test Planning** | Define scope, approach (manual/automation), tools, environment. | Plan to test UI on Chrome, Firefox, and mobile browsers. |
| **3. Test Case Design** | Create test cases covering UI functionality, layout, navigation, responsiveness, and accessibility. | Test case: Click Login without credentials → Show error message. |
| **4. Environment Setup** | Prepare browsers, devices, test data, and network conditions. | Setup app on Windows, Android, and iOS devices. |
| **5. Test Execution** | Execute manual/automated GUI test cases. Verify alignment, fonts, colours, and element behaviour. | Check if “Forgot Password” link redirects correctly. |
| **6. Defect Reporting** | Log GUI defects with screenshots and details. | Report: "Login button misaligned in Safari browser." |
| **7. Regression Testing** | Re-test GUI after bug fixes to ensure no new issues introduced. | After fixing button alignment, check entire login page again. |
| **8. Test Closure** | Summarize test results, defects found/fixed, and provide sign-off. | Report: 50 test cases executed, 47 passed, 3 defects logged. |