**SECTION A**

**Very Short Answer Questions**

**Attempt all questions (7\*2=14)**

1. **Differentiate between quality assurance and quality control.**

Quality Assurance (QA): QA focuses on preventing defects by ensuring that the processes used to manage and create deliverables are performed correctly. It is a proactive process that involves the establishment of processes and standards to improve software quality.

Quality Control (QC): QC is about identifying defects in the actual products produced. It is a reactive process that involves the inspection and testing of the final product to ensure it meets the required quality standards.

1. **Define Software development lifecycle.**

The Software Development Lifecycle (SDLC) is a systematic process for planning, creating, testing, and deploying software. It includes stages such as requirement analysis, design, implementation (coding), testing, deployment, and maintenance. SDLC provides a structured approach for developing software products, ensuring that they are delivered with high quality and within the time and budget constraints.

1. **Why is it important for a tester to follow communication etiquette?**

Clarity: Ensures that the information shared is clear and understandable.

Professionalism: Maintains a professional environment and fosters respect.

Efficiency: Reduces misunderstandings and errors, leading to more efficient problem-solving.

Collaboration: Enhances teamwork and collaboration among stakeholders.

Creditability: It also enhances the credibility and reliability of the tester's reports and findings.

1. **Differentiate between bug leakage and bug release**

Bug Leakage: Bug leakage occurs when a defect is found in production that should have been detected during earlier testing phases. It indicates a gap in the testing process.

Bug Release: Bug release refers to a situation where a known defect is intentionally left in the software due to time constraints or low impact, and the product is released with that defect. The decision is usually made with the understanding that the bug will not significantly affect the user experience or critical functionalities.

1. **Bug priority is subjective while severity is a constant factor. Justify this statement with a suitable example**

Severity refers to the impact a defect has on the system, and it is a constant factor because the impact does not change. For example, a crash in the system is always severe, regardless of when it is found.

Priority refers to how soon the defect should be fixed, which is subjective and can change based on factors like release deadlines or business needs. For instance, a typo on the homepage may have low severity but could be assigned high priority if the page is about to be shown to a key stakeholder.

1. **What are the unique aspects of mobile application testing?**

Device Diversity: Testing across various devices with different screen sizes, resolutions, and hardware configurations.

Operating Systems: Ensuring compatibility with multiple OS versions (e.g., iOS, Android).

Network Conditions: Testing under different network conditions (e.g., 3G, 4G, Wi-Fi).

Battery Usage: Assessing the app’s impact on battery life.

Touch Interface: Ensuring the app responds correctly to touch gestures.

1. **Prepare the test cases if you are given the task to test the registration page of a mobile application. Users can register using a valid email address and a password which should be at least 6 characters long.**

Test Case 1:

Title: Verify registration with a valid email and a password of exactly 6 characters.

Expected Result: Registration successful.

Test Case 2:

Title: Verify registration with a valid email and a password less than 6 characters.

Expected Result: Registration should fail, displaying an error message about the password length requirement.

Test Case 3:

Title: Verify registration with an invalid email format.

Expected Result: Registration should fail, displaying an error message about the invalid email format.

Test Case 4:

Title: Verify registration with an already registered email address.

Expected Result: Registration should fail, displaying an error message about the email already being in use.

**SECTION B**

**Short Answer Questions**

**Attempt any seven (7) questions out of nine (9) questions (7\*8=56)**

1. **Why are test data important? How would you manage and create test data repositories?**

Test data is critical in software testing because it is the input that is used to verify that the software functions as expected. The quality, relevance, and completeness of test data directly impact the accuracy and effectiveness of testing processes.

Importance of Test Data:

Validation of functionality: Ensures that the software functions correctly under various conditions.

Coverage: Helps in covering all possible scenarios, including edge cases which help to identify defects that might occur in unusual scenarios.

Performance Testing: In performance and load testing, large volumes of realistic test data are used to simulate real-world usage, helping in assessing the application’s performance under stress.

Regression Testing: Test data is crucial in regression testing to ensure that new changes have not adversely affected existing functionality.

Security Testing: In security testing, test data can include both valid and invalid inputs to check how the system handles data integrity, confidentiality, and other security concerns.

User Experience Testing: By using realistic test data, testers can assess the application's usability and ensure it meets the end-users' needs.

Managing and Creating Test Data Repositories:

Managing and creating test data repositories is essential for ensuring that test data is available, consistent, and usable across various testing scenarios. Here’s how we can manage and create test data repositories:

* Identify Requirements: Understand the data requirements based on test cases. Categorize data into different types such as static, dynamic and sensitive.
* Data Generation: Use automated tools or scripts to generate synthetic data that mimics real-world data but is anonymized. This is particularly useful when dealing with large datasets or when real data is not available. When applicable, use copies of production data for testing, ensuring that it is anonymized and complies with data protection regulations.
* Data Masking: Protect sensitive information by masking real data while preserving integrity.
* Version Control: Create a centralized test data repository that is accessible to all team members. This repository should be well-organized and categorized by project, module, or functionality. Implement version control for test data to track changes and maintain different versions for different test cases or environments.
* Storage: Store test data in databases, spreadsheets, or specialized test data management tools.

1. **Explain bug life cycle.**

Bug Life Cycle in software testing is the specific set of states that defect or bug goes through in its entire life. The purpose of Defect life cycle is to easily coordinate and communicate current status of defect which changes to various assignees and make the defect fixing process systematic and efficient.

New: When a new defect is logged and posted for the first time. It is assigned a status as NEW.

Assigned: Once the bug is posted by the tester, the lead of the tester approves the bug and assigns the bug to the developer team

Open/Active: The developer starts analyzing and works on the defect fix

Fixed: When a developer makes a necessary code change and verifies the change, he or she can make bug status as “Fixed.”

Verified: The tester re-tests the bug after it got fixed by the developer. If there is no bug detected in the software, then the bug is fixed, and the status assigned is “verified.”

Retest: Tester does the retesting of the code at this stage to check whether the defect is fixed by the developer or not and changes the status to “Re-test.”

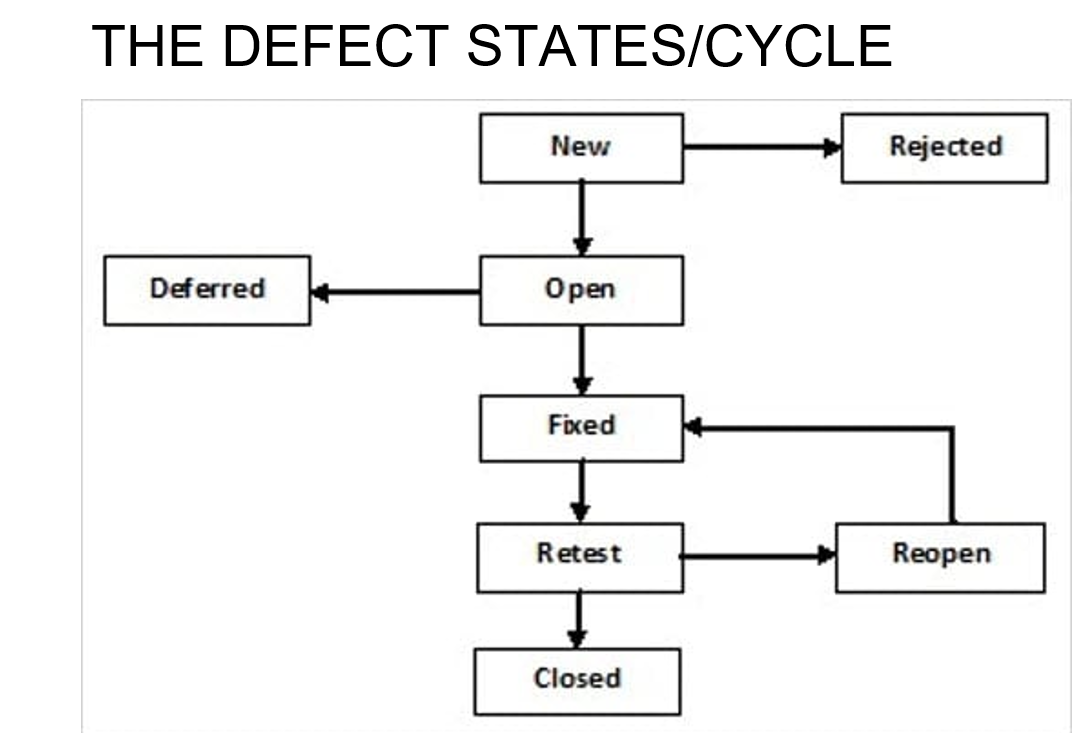
Reopen: If the bug persists even after the developer has fixed the bug, the tester changes the status to “reopened”. Once again, the bug goes through the life cycle.

Closed: If the bug is no longer exists then tester assigns the status “Closed.”

Duplicate: If the defect is repeated twice or the defect corresponds to the same concept of the bug, the status is changed to “duplicate.”

Rejected: If the developer feels the defect is not a genuine defect, then it changes the defect to “rejected.” A defect can be rejected for any of the 3 reasons: viz - duplicate defect, NOT a Defect, Non-Reproducible.

Deferred: If the present bug is not of a prime priority and if it is expected to get fixed in the next release, then status “Deferred” is assigned to such bugs



1. **What is performance testing? What are the various aspects that should be evaluated while performing a comprehensive performance test?**

Performance testing is testing of an application’s stability and response time by applying load. It Identifies performance bottlenecks and ensures the system’s reliability, scalability, and responsiveness.

The various aspects that should be evaluated while performing a comprehensive performance test are as follows:

* Load Testing

It is testing of an application’s stability and response time by applying load, which is equal to or less than the designed number of users for an application.

For example, your application handles 100 users at a time with a response time of 3 seconds, then load testing can be done by applying a load of the maximum of 100 or less than 100 users. The goal is to verify that the application is responding within 3 seconds for all the users.

* Stress Testing

It is testing an application’s stability and response time by applying load, which is more than the designed number of users for an application.

For example, your application handles 1000 users at a time with a response time of 4 seconds, then stress testing can be done by applying a load of more than 1000 users.

Test the application with 1100,1200,1300 users and notice the response time. The goal is to verify the stability of an application under stress.

* Scalability Testing

Scalability testing is testing an application’s stability and response time by applying load, which is more than the designed number of users for an application.

For example, your application handles 1000 users at a time with a response time of 2 seconds, then scalability testing can be done by applying a load of more than 1000 users and gradually increasing the number of users to find out where exactly my application is crashing.

Let’s say my application is giving response time as follows:

1000 users -2 sec

1400 users -2 sec

4000 users -3 sec

5000 users -45 sec

5150 users- crash – This is the point that needs to identify in scalability testing

* Volume Testing

Volume testing is testing an application’s stability and response time by transferring a large volume of data to the database.

Basically, it tests the capacity of the database to handle the data.

* Endurance Testing (Soak Testing)

Endurance testing is testing an application’s stability and response time by applying load continuously for a longer period to verify that the application is working fine.

For example, car companies soak testing to verify that users can drive cars continuously for hours without any problem.

**4. If you encounter two different defects listed below, how would you label their severity and priority? Also provide the justification for your answers**

a. The contents of the page render slowly

Severity: Medium

It affects user experience but does not prevent functionality.

Slow rendering impacts user experience but does not make the page completely unusable. Users can still access the content, but they may face delays, which could lead to frustration, especially if the page is critical for user engagement or conversion. However, it does not cause any loss of functionality, so it is considered a medium severity issue.

Priority: High

Needs to be fixed quickly to improve user satisfaction.

In most cases, slow page rendering is a high-priority issue because it directly affects user satisfaction and retention. If the page is part of a critical user flow, like a checkout process, this defect could lead to a loss of business, so fixing it promptly is essential. However, if it's on a less frequently visited page, the priority might be adjusted accordingly.

b. The webpage is not responsive

Severity: High

A non-responsive webpage can severely impact usability across various devices, particularly mobile devices, which make up a significant portion of web traffic. If users cannot interact with the webpage properly on their devices, it could render the application or site unusable for a large segment of users, making this a high-severity issue.

Priority: High

Ensuring responsiveness is critical in today’s multi-device environment. A non-responsive webpage can lead to a poor user experience, reduced accessibility, and potentially lost revenue, especially if the site is customer-facing. This issue would likely be prioritized for a quick fix to maintain usability across different platforms.

1. **Create a test case scenario for an e-commerce platform if you are asked to perform the security test.**

**It is done to check how the software, application, or website is secure from internal and/or external threats. E.g. Penetration testing**

**Test Case 1: SQL Injection Vulnerability**

**Steps: Enter SQL injection string (‘ ' OR '1'='1’) in the username and password fields.**

**Expected Result: The system should not allow login and should display an error message.**

**Test Case 2: Cross-Site Scripting (XSS)**

**Steps: Enter <script>alert('XSS')</script> in the comment section, search bar, or other input fields.**

**Expected Result: The system should sanitize the input and not execute the script.**

**Test Case 3: Brute Force Attack**

**Steps: Attempt multiple login attempts with different passwords.**

**Expected Result: The system should lock the account after a certain number of failed attempts.**

**Test Case 4: Secure Password Storage**

**Steps: Register a new user and check the database for password storage.**

**Expected Result: Passwords should be stored in a hashed format, not plain text.**

**Test Case 5: Sensitive Data Exposure**

**Description: Verify that sensitive data (e.g., credit card numbers, passwords) is not exposed in logs, error messages, or through the UI.**

**Steps:**

**Simulate a failed transaction and observe the error message.**

**Check the browser's developer tools for any exposed sensitive data.**

**Monitor server logs during transactions and error scenarios.**

**Expected Results: No sensitive data should be displayed in error messages, visible in the UI, or logged in plaintext. Sensitive data should be encrypted or masked.**

**Test Case 6: User Authentication Security**

**Description: Verify that the platform enforces strong password policies during user registration and password changes.**

**Steps:**

**Attempt to register a new user with a weak password (e.g., "12345").**

**Attempt to register with a password that meets the platform's complexity requirements (e.g., minimum 8 characters, including uppercase, lowercase, numbers, and special characters).**

**Attempt to change the password to a weak password.**

**Expected Results:**

**The platform should reject weak passwords during registration.**

**The platform should accept only passwords that meet the complexity requirements.**

**The platform should reject any attempts to change the password to a weak password.**

**Test Case 7: Data Encryption Validation**

**Description: Verify that all sensitive data transmissions (e.g., login credentials, payment information) are encrypted.**

**Steps:**

**Monitor network traffic using tools like Wireshark during the login process and checkout.**

**Inspect the data to ensure it is encrypted (e.g., HTTPS).**

**Expected Results: All sensitive data should be transmitted over an encrypted channel (e.g., SSL/TLS) and should not be viewable in plaintext.**

1. **Why is it necessary to prepare a defect report? Write a sample defect report (assume all necessary information on your own)**

A defect report is a crucial document in the software development lifecycle, particularly in the testing phase. It serves several important purposes:

Communication: A defect report acts as a formal communication tool between testers, developers, project managers, and other stakeholders. It ensures that everyone is aware of the defect and understands its impact on the project.

Tracking and Management: It helps in tracking the status of defects from detection to resolution. Defect reports are often managed in a bug-tracking system, where they are prioritized, assigned, and monitored until they are fixed.

Accountability: Defect reports create a record of the issues found during testing, which can be used to hold team members accountable for addressing these issues in a timely manner.

Documentation: It serves as documentation that can be referred to in future development cycles. This helps in preventing the recurrence of similar defects and improving the overall quality of the software.

Decision-Making: Defect reports provide data that can be analyzed to make informed decisions about the software's release. For example, if a critical defect is found, the release might be delayed until the issue is resolved.

Quality Assurance: It contributes to the overall quality assurance process by ensuring that defects are systematically identified, documented, and addressed before the software is delivered to the end users.

**Example: Defect Report for Employee Login Page**

Defect id: D001

Project Name: MyASP

Module Name: Login

Sub Module Name: Employee Login

Type of Defect: Missing

Status: New

Priority: High

Summary: Employee Login Page Not Opening

Description:

Enter URL

Click On the Employee Login Option

Expected Result: Employee login page should get open

Actual Results: Employee login page does not get open

Reported By: ABC Tester

Assign To: XYZ Developer

Date & Time: 12/28/2019

1. **How can you create a collection and environment in Postman?** What **two files are necessary to execute test cases using Postman's command line interface.**

Creating a collection in Postman:

Open Postman.

Click on the Collections tab.

Click New Collection.

Name the collection and add a description.

Add requests to the collection by clicking Add Request.

Creating an Environment in Postman:

Click on the Environments tab.

Click New Environment.

Name the environment.

Add variables and their values.

Save the environment.

Files Necessary for Postman’s Command Line Interface (Newman):

Collection File: A JSON file containing the collection of requests. This file contains all the API requests, test scripts, and folders in the collection. You can export this file from Postman by selecting the collection, clicking the three dots next to the collection name, and choosing Export.

Environment File: A JSON file containing environment variables. This file contains all the environment variables defined in the Postman environment. You can export this file by going to the Manage Environments section, selecting the environment, and clicking Export.

To run your tests using Newman, you can use the following command:

newman run <path\_to\_collection\_file.json> -e <path\_to\_environment\_file.json>

1. **When is it preferable to use Selenium? List out necessary steps to automate tests using Selenium web driver.**

Selenium is a popular open-source tool used for automating web browsers. It is preferable to use Selenium in the following scenarios:

Web Application Testing: Selenium is ideal for testing web applications across different browsers (Chrome, Firefox, Safari, etc.) and platforms (Windows, macOS, Linux).

Cross-Browser Testing: Selenium supports multiple browsers, making it suitable for cross-browser testing to ensure consistent behavior across different browsers.

Regression Testing: Selenium is effective for automating repetitive test cases, such as regression tests, where you need to verify that new changes haven’t broken existing functionality.

Continuous Integration/Continuous Deployment (CI/CD): Selenium integrates well with CI/CD tools like Jenkins, enabling automated testing as part of the build process.

Data-Driven Testing: Selenium supports data-driven testing, where tests are executed using multiple sets of input data, typically read from external sources like Excel, CSV files, or databases.

Complex User Interactions: Selenium can handle complex user interactions like drag-and-drop, double-click, hover, and right-click, which are difficult to automate using simpler tools.

Custom Browser Behavior: Selenium allows for fine control over browser behavior, such as setting cookies, managing sessions, and handling JavaScript alerts, which are essential for certain testing scenarios.

1. **Explain functional testing.**

**Functional Testing is a type of software testing that focuses on verifying that the software application performs its functions correctly according to the specified requirements. It is primarily concerned with testing the software's functionality rather than its internal structures or workings.**

**It ensures that the application behaves as expected and meets user requirements.**

**Unit Testing:**

* **testing which is done on an individual unit or component to test its corrections**
* **done by the developer at the application development phase**
* **each unit in unit testing can be viewed as a method, function, procedure, or object.**
* **developers often use test automation tools such as NUnit, Xunit, JUnit for the test execution.**
* **Important: because we can find more defects at the unit test level.**
* **For example, there is a simple calculator application. The developer can write the unit test to check if the user can enter two numbers and get the correct sum for addition functionality.**

**White Box Testing**

**A test technique in which the internal structure or code of an application is visible and accessible to the tester**

**it is easy to find loopholes in the design of an application or fault in business logic**

**Gorilla Testing**

**A test technique in which the tester and/or developer test the module of the application thoroughly in all aspects.**

**Gorilla testing is done to check how robust your application is.**

**For example, the tester is testing the pet insurance company’s website, which provides the service of buying an insurance policy, tag for the pet, Lifetime membership. The tester can focus on any one module, let’s say, the insurance policy module, and test it thoroughly with positive and negative test scenarios.**

**Integration Testing**

* **a type of software testing where two or more modules of an application are logically grouped together and tested as a whole.**
* **The focus of this type of testing is to find the defect on interface, communication, and data flow among modules.**
* **This type of testing is done on integrating modules of a system or between systems**
* **For example, a user is buying a flight ticket from any airline website. Users can see flight details and payment information while buying a ticket, but flight details and payment processing are two different systems. Integration testing should be done while integrating of airline website and payment processing system.**

**Gray Box Testing**

**gray box testing is a combination of white-box testing and black-box testing. Testers have partial knowledge of the internal structure or code of an application.**

**System Testing:**

**System testing is types of testing where tester evaluates the whole system against the specified requirements.**

**End To End Testing**

* **It involves testing a complete application environment in a situation that mimics real-world use, such as interacting with a database, using network communications, or interacting with other hardware, applications, or systems if appropriate.**
* **For example, a tester is testing a pet insurance website. End to End testing involves testing of buying an insurance policy, LPM, tag, adding another pet, updating credit card information on users’ accounts, updating user address information, receiving order confirmation emails and policy documents.**

**Black Box Testing**

* **testing technique in which testing is performed without knowing the internal structure, design, or code of a system under test.**
* **Testers should focus only on the input and output of test objects.**

**Smoke Testing**

* **performed to verify that basic and critical functionality of the system under test is working fine at a very high level.**
* **whenever a new build is provided by the development team, then the Software Testing team validates the build and ensures that no major issue exists.**
* **testing team will ensure that the build is stable, and a detailed level of testing will be carried out further.**

**Sanity Testing**

* **Sanity testing is performed on a system to verify that newly added functionality or bug fixes are working fine.**
* **Sanity testing is done on stable build. It is a subset of the regression test.**
* **For example, a tester is testing a pet insurance website. There is a change in the discount for buying a policy for second pet. Then sanity testing is only performed on buying insurance policy module.**

**Happy Path Testing**

* **The objective of Happy Path Testing is to test an application successfully on a positive flow.**
* **It does not look for negative or error conditions.**
* **The focus is only on valid and positive inputs through which the application generates the expected output.**

**Monkey Testing**

* **It is carried out by a tester, assuming that if the monkey uses the application, then how random input and values will be entered by the Monkey without any knowledge or understanding of the application.**
* **The objective of Monkey Testing is to check if an application or system gets crashed by providing random input values/data.**

**Acceptance testing:**

**Alpha Testing**

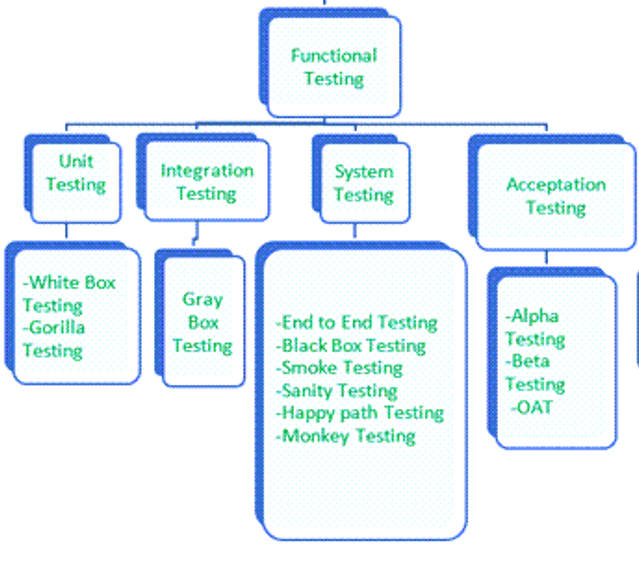
* **Alpha testing is a type of acceptance testing performed by the team in an organization to find as many defects as possible before releasing software to customers.**

**Beta Testing**

* **It is a type of software testing which is carried out by the clients/customers. It is performed in the Real Environment before releasing the product to the market for the actual end-users.**
* **Beta Testing is carried out to ensure that there are no major failures in the software or product, and it satisfies the business requirements from an end-user perspective.**
* **Beta Testing is successful when the customer accepts the software.**

**OAT (Operational Acceptance Testing)/UAT (User Acceptance Testing)**

* **Operational acceptance testing of the system is performed by operations or system administration staff in the production environment.**
* **The purpose of operational acceptance testing is to make sure that the system administrators can keep the system working properly for the users in a real-time environment.**



**SECTION C**

**Long Answer Questions**

**Attempt any two (2) questions out of three (3) questions (2\*15=30)**

1. **Suppose you are a Test Manager of a leading e-commerce platform. So, what are all the static and dynamic tests you will perform in order to deliver a bug free application?**

As a Test Manager for a leading e-commerce platform, ensuring a bug-free application involves a comprehensive testing strategy that includes both static and dynamic testing techniques. Here's an outline of the static and dynamic tests that should be performed:

Static Testing

Static testing involves examining the software's code, documentation, and other artifacts without executing the program. This can help identify issues early in the development process.

Code Review:

Peer Reviews: Conduct regular peer reviews of code to identify potential defects and ensure adherence to coding standards.

Pair Programming: Encourage pair programming where two developers work together at one workstation, which can help in identifying defects and improving code quality.

Static Code Analysis:

Automated Tools: Use static analysis tools (e.g., SonarQube, Checkmarx) to analyze the source code for common issues such as coding standard violations, security vulnerabilities, and potential bugs.

Code Metrics: Review metrics like code complexity, cyclomatic complexity, and code duplication to ensure code maintainability.

Design and Architecture Review:

Design Documents: Review design documents to ensure that the architecture supports the functional and non-functional requirements.

Architecture Evaluation: Evaluate the architecture for scalability, performance, and security considerations.

Requirement Review:

Requirement Analysis: Review requirement documents to ensure that they are complete, clear, and unambiguous.

Traceability: Ensure that there is proper traceability between requirements and test cases.

Documentation Review:

Test Plans and Cases: Review test plans and test cases to ensure that they cover all requirements and are well-documented.

User Manuals: Verify user manuals and other documentation for accuracy and completeness.

Static Security Analysis:

Security Policies: Review security policies and configurations to ensure they align with industry best practices and security standards.

Dynamic Testing

Dynamic testing involves executing the software to find defects through functional and non-functional testing. This approach helps in validating the actual behavior of the software.

Functional Testing:

* Unit Testing: Verify individual units or components of the application for correct functionality. Tools like JUnit (Java), NUnit (.NET), and pytest (Python) can be used.
* Integration Testing: Test the interactions between integrated modules or components to ensure they work together as expected.
* System Testing: Perform end-to-end testing of the entire application to validate that all components work together and meet the specified requirements.
* Acceptance Testing: Conduct user acceptance testing (UAT) to ensure the application meets business requirements and is ready for production.

Performance Testing:

Load Testing: Assess how the application performs under normal and peak load conditions using tools like JMeter or LoadRunner.

Stress Testing: Evaluate how the application handles extreme conditions and stress scenarios.

Scalability Testing: Test the application’s ability to scale up or down to handle varying loads.

Performance Profiling: Measure the performance of different parts of the application to identify bottlenecks and optimize performance.

Security Testing:

Vulnerability Scanning: Use tools (e.g., OWASP ZAP, Burp Suite) to identify security vulnerabilities such as SQL injection, cross-site scripting (XSS), and other common threats.

Penetration Testing: Perform penetration testing to simulate attacks and assess the application’s security posture.

Security Code Review: Review code for security vulnerabilities and ensure adherence to secure coding practices.

Usability Testing:

User Interface Testing: Verify that the user interface is intuitive, user-friendly, and aligns with design guidelines.

Accessibility Testing: Ensure the application is accessible to users with disabilities, complying with standards such as WCAG (Web Content Accessibility Guidelines).

Compatibility Testing:

Cross-Browser Testing: Ensure the application functions correctly across different web browsers (Chrome, Firefox, Safari, Edge).

Cross-Device Testing: Validate that the application works well on various devices (desktops, tablets, smartphones) and screen sizes.

Operating System Testing: Test the application on different operating systems (Windows, macOS, Linux) to ensure compatibility.

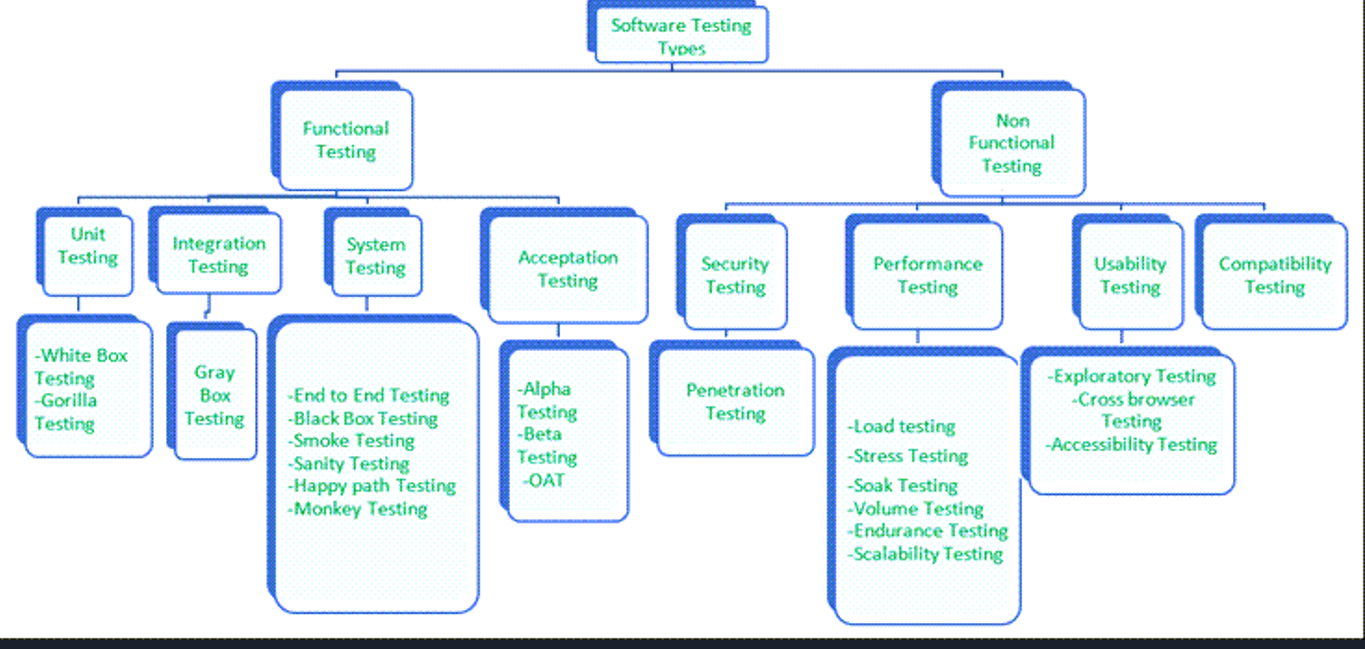
Regression Testing:

Automated Regression Tests: Re-run previously executed test cases to ensure that recent changes have not introduced new defects. Automation tools like Selenium can be used for this purpose.

Exploratory Testing:

Ad-Hoc Testing: Conduct exploratory testing to discover defects that may not be covered by formal test cases. This involves testers using their knowledge and experience to explore the application and identify issues.

1. **Explain in detail all the different levels of testing.**



Functional testing as above QS 9.

Non-Functional testing:

1. Security Testing

* It is a type of testing performed by a special team. Any hacking method can penetrate the system.
* It is done to check how the software, application, or website is secure from internal and/or external threats.
* This testing includes how much software is secure from malicious programs, viruses and how secure & strong the authorization and authentication processes are.

Penetration Testing

Pen testing is performed by outside contractors, generally known as ethical hackers.

That is why it is also known as ethical hacking.

Contractors perform different operations like SQL injection, URL manipulation, Privilege Elevation, session expiry, and provide reports to the organization.

1. Performance Testing: Q no 3
2. Usability Testing

* It is testing an application from the user’s perspective to check the look and feel and user-friendliness.
* The main idea of usability testing of this kind of app is that as soon as the user opens the app, the user should get a glance at the market.

Exploratory Testing

* Exploratory Testing is informal testing performed by the testing team.
* The objective of this testing is to explore the application and look for defects that exist in the application.
* Testers use the knowledge of the business domain to test the application.

Test charters are used to guide the exploratory testing.

Cross Browser Testing

* Cross browser testing is testing an application on different browsers, operating systems, mobile devices to see look and feel and performance.
* We need this testing because different users use different operating systems, different browsers, and different mobile devices. The goal of the company is to get a good user experience regardless of those devices.

Accessibility Testing

* The aim of this kind of testing is to determine whether the software or application is accessible for disabled people or not.
* Here, disability means deafness, color blindness, mentally disabled, blind, old age, and other disabled groups.
* Various checks are performed, such as font size for visually disabled, color and contrast for color blindness, etc.

4. Compatibility Testing

* This is a testing type in which it validates how software behaves and runs in a different environment, web servers, hardware, and network environment.
* It ensures that software can run on different configuration, different databases, different browsers, and their versions. The testing team performs compatibility testing.

1. Phoenix college is about to incorporate a new feature that allows the pre-registered students to make fee payments in their pre-existing web-based student portal. If you are responsible for delivering a complete bug free application, what are the test cases, test scenarios and test scripts that you will write. (For test scripts, please write pseudo codes only)

To ensure the new fee payment feature in Phoenix College’s web-based student portal is bug-free, we need to create comprehensive test cases, test scenarios, and test scripts. Here’s a detailed plan:

Test Scenarios

1. Login Functionality:

* Verify that pre-registered students can log in successfully.
* Verify that non-registered users cannot access the payment feature.

1. Navigation to Payment Page:

* Verify that students can navigate to the payment page from the dashboard.

1. Payment Form Validation:

* Verify that all required fields (e.g., amount, payment method) are validated.
* Verify that invalid inputs (e.g., incorrect card number) are handled correctly.

1. Payment Processing:

* Verify that payments are processed successfully with valid details.
* Verify that payments fail with invalid details.

1. Payment Confirmation:

* Verify that students receive a confirmation message after a successful payment.
* Verify that students receive an error message if the payment fails.

1. Transaction History:

* Verify that the payment is recorded in the student’s transaction history.

1. Security:

* Verify that sensitive information (e.g., card details) is encrypted.
* Verify that the payment feature is protected against common security threats (e.g., SQL injection, XSS).

1. Usability:

* Ensure that the payment feature is user-friendly and intuitive.

Test Cases:

1. User Authentication Test Cases:

* Test Case 1.1: Verify that a pre-registered student can log in to the portal.
  + Precondition: Student has valid login credentials.
  + Steps:
    1. Navigate to the login page.
    2. Enter valid credentials (username and password).
    3. Click "Login."
  + Expected Result: Student should be logged in and redirected to the portal homepage.
* Test Case 1.2: Verify that a student cannot access the payment feature without logging in.
  + Precondition: Student is not logged in.
  + Steps:
    1. Navigate directly to the fee payment page.
  + Expected Result: Student should be redirected to the login page.

2. Payment Interface Test Cases:

* Test Case 2.1: Verify that the fee payment page displays the correct outstanding fee amount.
  + Precondition: Student is logged in.
  + Steps:
    1. Navigate to the fee payment page.
  + Expected Result: The page should display the correct outstanding fee amount for the student.

3. Payment Processing Test Cases:

* Test Case 3.1: Verify that a student can successfully make a payment using a valid credit/debit card.
  + Precondition: Student is logged in and has a valid credit/debit card.
  + Steps:
    1. Navigate to the fee payment page.
    2. Enter payment details (card number, expiry date, CVV).
    3. Click "Pay."
  + Expected Result: Payment should be processed successfully, and the student should receive a confirmation message.
* Test Case 3.2: Verify that an error message is displayed for an invalid card number.
  + Precondition: Student is logged in and enters an invalid card number.
  + Steps:
    1. Navigate to the fee payment page.
    2. Enter invalid card details.
    3. Click "Pay."
  + Expected Result: An appropriate error message should be displayed.

4. Confirmation and Receipt Test Cases:

* Test Case 4.1: Verify that a confirmation receipt is sent to the student’s email.
  + Precondition: Payment has been processed successfully.
  + Steps:
    1. Complete a payment.
    2. Check the student’s email.
  + Expected Result: The student should receive a confirmation email with payment details.

5. Error Handling Test Cases:

* Test Case 5.1: Verify that an appropriate error message is displayed for network failures during payment.
  + Precondition: Student is logged in and attempting a payment.
  + Steps:
    1. Simulate a network failure during payment processing.
  + Expected Result: An error message indicating a network issue should be displayed.

6. Data Integrity Test Cases:

* Test Case 6.1: Verify that the payment amount is correctly recorded in the student’s payment history.
  + Precondition: Student has made a payment.
  + Steps:
    1. Navigate to the payment history section.
    2. Check the recorded payment details.
  + Expected Result: The payment amount and details should match the transaction.

7. Security Test Cases:

* Test Case 7.1: Verify that payment information is transmitted securely (e.g., HTTPS).
  + Precondition: Student is on the payment page.
  + Steps:
    1. Check the URL of the payment page.
  + Expected Result: The URL should start with "https://" indicating a secure connection.

8. Usability Test Cases:

* Test Case 8.1: Verify that the payment process is intuitive and user-friendly.
  + Precondition: Student is on the payment page.
  + Steps:
    1. Complete the payment process.
  + Expected Result: The payment process should be straightforward and easy to understand.

**Test Scripts (Pseudo Code)**

**1. User Authentication Test Script (Pseudo Code):**

// Test Case 1.1: Verify student login

NavigateToLoginPage()

EnterUsername("validUsername")

EnterPassword("validPassword")

ClickLoginButton()

VerifyRedirectToHomepage()

VerifySuccessfulLogin()

**2. Payment Processing Test Script (Pseudo Code):**

// Test Case 3.1: Verify successful payment

NavigateToFeePaymentPage()

EnterCardDetails("4111111111111111", "12/24", "123")

ClickPayButton()

VerifySuccessfulPayment()

VerifyConfirmationMessage()

VerifyEmailReceipt()

**3. Error Handling Test Script (Pseudo Code):**

// Test Case 5.1: Verify network failure error message

NavigateToFeePaymentPage()

SimulateNetworkFailure()

EnterCardDetails("4111111111111111", "12/24", "123")

ClickPayButton()

VerifyNetworkErrorMessageDisplayed()