QA Processes Assignment Questions

Understanding QA Basics:

- **Q1:** Define Quality Assurance (QA) and Quality Control (QC). What are the key differences between them?
- **Quality Assurance (QA)** is a *process-oriented* approach that focuses on **preventing defects** in the software development process. It involves setting up **standards, procedures, and methodologies** to ensure quality from the beginning of the project. QA activities include process audits, documentation reviews, and continuous improvement.
- **Quality Control (QC)** is a *product-oriented* approach that focuses on **identifying defects** in the final software product. It involves **actual testing and inspection** of the developed software to verify that it meets the specified requirements. QC activities include executing test cases, finding bugs, and validating outputs.

Key Differences:

Aspect	**Quality Assurance (QA)**	**Quality Control (QC)**
Focus	Process-oriented	Product-oriented
Goal	Prevent defects	Detect defects
When it occurs	During the development process	After development, during or after testing
Responsibility	Entire development team/process management	Testing team
Example Activities	Process audits, documentation reviews	Test execution, defect reporting

- **Q2:** Explain the role of a QA engineer in the software development lifecycle (SDLC).

  A **QA (Quality Assurance) engineer** plays a critical role throughout the Software Development Life Cycle (SDLC) to ensure the delivery of high-quality, reliable software. Their primary responsibility is to prevent defects and ensure that the final product meets user expectations and business requirements.

Role of a QA Engineer Across SDLC Phases:

1. **Requirement Analysis**
    - Understand and review requirements for completeness and testability
    - Raise clarifications and contribute to requirement refinement
2. **Planning**
    - Help design a test strategy and define test objectives
    - Estimate time, resources, and tools needed for testing
3. **Design**
    - Design test cases and test scenarios based on requirements
    - Define traceability matrices to ensure full test coverage
4. **Development Support**
    - Participate in code reviews (if applicable)
    - Collaborate with developers to ensure testability of code
5. **Testing**
    - Execute various types of tests (unit, integration, system, acceptance)
    - Log, track, and manage bugs
    - Perform regression and performance testing as needed
6. **Deployment**
    - Verify deployment readiness
    - Conduct smoke/sanity testing post-deployment
7. **Maintenance**
    - Monitor application in the live environment
    - Test patches and updates
    - Update test cases based on changes

- **Q3:** List the different types of testing (e.g., functional, non-functional) and explain when each type is used.

1. Functional Testing

Tests whether the software behaves according to its functional requirements.

Used When:

- Verifying features like login, form submission, search, etc.
- After development of each feature/module.

Examples:

- Unit Testing
- Integration Testing
- System Testing
- User Acceptance Testing (UAT)

2. Non-Functional Testing

Evaluates aspects not related to specific functions, such as performance and usability.

Used When:

- After major builds or before production release
- To ensure quality in speed, scalability, and user experience.

Examples:

- Performance Testing (checks speed and load handling)
- Security Testing (checks for vulnerabilities)

3. Regression Testing

Checks that new changes haven't broken existing functionality.

Used When:

- After bug fixes or adding new features
- During every sprint in Agile

4. Smoke Testing

A quick check to see if the basic functionalities work.

Used When:

- After a new build or deployment
- Before deeper testing begins

5. Sanity Testing

Focused testing of a specific area after a bug fix or change.

Used When:

- To validate specific issues without doing full regression

6. Acceptance Testing

Checks if the software meets business requirements.

Used When:

- At the end of development
- Before client approval or product launch

2. Test Planning and Strategy:

- **Q4:** What is a test plan? Create a simple test plan outline for testing a login page of a web application. Include sections like objectives, scope, test strategy, and resources.
- A **test plan** is a formal document that outlines the **objectives, scope, approach, resources, schedule, and activities** required for testing a software application. It serves as a guide for the QA team and helps ensure that all aspects of the application are tested systematically and thoroughly.

  Test Plan Outline for Login Page

  1. Test Plan ID:
  TP-LP-001
  2. Test Objective:
  To verify that the login functionality works as expected with valid and invalid inputs, and that error messages are shown correctly.

  3. Scope:
  In-Scope:
- Valid and invalid login scenarios
- Field validations
- Password masking
- Remember Me functionality
- Error and success message checks

  Out of Scope:
- Signup or password recovery pages
- Backend database validations

  4. Test Strategy:
- **Testing Type:** Functional Testing, UI Testing, Negative Testing
- **Test Approach:** Manual testing using test cases
- **Test Environment:** Chrome, Firefox, Edge browsers on Windows and Android
- **Tools:** TestRail for test case management, Postman for API check (if required)

  5. Test Cases Overview:
- Login with valid username/password
- Login with incorrect credentials
- Empty field submission

- Password field should be masked
- Session handling after login

6. Entry Criteria:

- Login module is deployed in the test environment
- Requirements and UI design are finalized
- Test data is prepared

7. Exit Criteria:

- All high and medium severity bugs are fixed
- All test cases have been executed and passed
- Test summary report has been reviewed

8. Test Resources:

- 1 QA Engineer
- Test devices (Windows PC, Android phone)
- Browsers (Chrome, Firefox, Edge)

9. Risks and Assumptions:

- UI changes during testing phase
- Backend authentication service may be unstable

- **Q5:** Explain the concept of "Test Coverage". How can you ensure high test coverage in a project?

  Test coverage is a measure of how much of the software code or functionality is tested by the test cases. It helps determine the **effectiveness and completeness** of testing and identifies untested parts of the application.
  There are different types of test coverage, such as:

- **Code coverage** (lines, branches, conditions)
- **Requirement coverage** (ensuring all features are tested)
- **Path coverage** (all possible logical paths are tested)

How to Ensure High Test Coverage

1. **Map Test Cases to Requirements**
   Create a **requirements traceability matrix** to ensure every requirement is covered by at least one test case.

2. **Automated Testing**
   Use tools like Selenium, JUnit, or Cypress to automate repetitive tests, especially for regression testing.
3. **Review Test Cases**
   Regularly review and update test cases to cover new features and edge cases.
4. **Use Coverage Tools**
   Tools like JaCoCo (Java), Istanbul (JavaScript), or Coverage.py (Python) help track how much code is actually being tested.
5. **Include Negative & Edge Cases**
   Don't just test normal flows. Include invalid inputs and boundary conditions to improve coverage.

- **Q6:** What is a test strategy? How does it differ from a test plan? Provide examples of what could be included in a test strategy document.
- A **test strategy** is a high-level document that defines the **approach and overall direction** of the testing process across a project or organization. It outlines how testing will be done, what testing types will be used, and how quality will be ensured consistently.
- It is usually created by a QA lead or test manager and serves as a guideline for writing detailed test plans.

Difference Between Test Strategy and Test Plan

Aspect	**Test Strategy**	**Test Plan**
Level	High-level (organizational or project-wide)	Detailed (module or feature-specific)
Focus	Overall testing approach	Specific testing tasks, schedules, resources
Created By	QA Manager or Test Lead	Test Engineer or QA team
Scope	Applies to entire project or multiple projects	Applies to one project or component

3. Test Case Design:

- **Q7:** What is a test case? Write test cases for a user registration feature of a website. Include valid and invalid inputs.
  A **test case** is a set of conditions or steps used to verify whether a feature or function of a software application works as expected. It includes the input data, execution steps, expected results, and actual results.
  Sample Test Cases for User Registration Feature

  Test Case 1: Register with Valid Inputs
- **Input:** Valid name, email, password, confirm password
- **Expected Result:** User account is created successfully, and user is redirected to the welcome/dashboard page

  Test Case 2: Register with Already Registered Email
- **Input:** Valid name and password, but email already exists in the system
- **Expected Result:** Display message "Email already registered"

  Test Case 3: Password and Confirm Password Mismatch
- **Input:** Valid name and email, password and confirm password are different
- **Expected Result:** Show error message "Passwords do not match"
  =
  Test Case 4: Invalid Email Format
- **Input:** Email without "@" or domain part (e.g., "usergmail.com")
- **Expected Result:** Display validation message "Enter a valid email address"

  Test Case 5: Submit Empty Form
- **Input:** All fields left blank
- **Expected Result:** Show required field messages for all inputs

  Test Case 6: Password Too Short
- **Input:** Password less than minimum required characters (e.g., 4 characters)
- **Expected Result:** Show message "Password must be at least 8 characters"

- **Q8:** Explain the components of a test case. Write a test case to verify the functionality of the "Forgot Password" feature.
  Components of a Test Case:

1. **Test Case ID** – A unique identifier for the test case (e.g., TC_FP_001)

2. **Test Case Description** – Brief description of what is being tested
3. **Preconditions** – Conditions that must be met before execution
4. **Test Steps** – Actions to perform during the test
5. **Test Data** – Inputs required to execute the test
6. **Expected Result** – The expected output or behavior
7. **Actual Result** – The actual output after execution
8. **Status** – Pass or Fail based on result comparison
9. **Remarks** – Additional notes or observations

Sample Test Case for "Forgot Password" Feature

- **Test Case ID:** TC_FP_001
- **Description:** Verify that the user can reset the password using the "Forgot Password" feature
- **Preconditions:** User account with a registered email already exists
- **Test Steps:**
    1. Navigate to the login page
    2. Click on "Forgot Password" link
    3. Enter registered email address
    4. Click on "Submit" or "Send Reset Link"
- **Test Data:** Email = user@example.com
- **Expected Result:**
  A confirmation message is displayed: “Password reset link has been sent to your email.”
  An email is sent to the provided address with the reset link.
- **Actual Result:** As expected
- **Status:** Pass

- **Q9:** What is boundary value analysis (BVA)? Create a set of test cases using BVA for an input field that accepts age (range 18–60).
  Boundary Value Analysis (BVA) is a black-box testing technique where tests are designed to include values at the **boundaries of input ranges**, because defects often occur at the edges rather than in the middle.
  Instead of testing all possible inputs, BVA focuses on:
- **Minimum value**
- **Maximum value**
- **Just below the minimum**
- **Just above the maximum**
- **A value just inside the range**

Scenario: Age Field (Valid range: 18–60)

Test Cases Using BVA:

1. **Age = 17** (just below minimum) → Should be **rejected**
2. **Age = 18** (minimum valid value) → Should be **accepted**
3. **Age = 19** (just above minimum) → Should be **accepted**
4. **Age = 59** (just below maximum) → Should be **accepted**
5. **Age = 60** (maximum valid value) → Should be **accepted**
6. **Age = 61** (just above maximum) → Should be **rejected**

4. Types of Testing:

- **Q10:** Differentiate between white-box testing and black-box testing. Provide examples of each.

Key Differences

Aspect	**White-Box Testing**	**Black-Box Testing**
Code knowledge	Required	Not required
Focus area	Internal logic and flow	Inputs, outputs, and functionality
Who performs it	Developers or technical testers	QA testers or end-users
Example	Unit testing using control statements	Checking login form with different inputs

- **Q11:** What is regression testing, and why is it important? Describe a scenario where regression testing would be necessary.
- **Regression testing** is the process of re-running existing test cases to ensure that **new code changes have not negatively impacted** the previously working functionality of the software.
- It helps confirm that recent modifications (like bug fixes, enhancements, or new features) have not introduced new bugs into existing areas of the application.

Why is Regression Testing Important?

- Prevents old bugs from reappearing
- Maintains software stability over time
- Ensures that recent changes don't break unrelated features
- Builds confidence in the reliability of the system

Example Scenario:

Suppose a developer fixes a bug in the user registration module where the system wasn't validating duplicate email addresses correctly.

After the fix is applied:

- Regression testing would include re-testing **not only the email validation** but also all related features like:
    - Password validation
    - Field-level error messages
    - Form submission and redirection
    - Database record creation

This ensures that the fix didn't accidentally break other parts of the registration process.

- **Q12:** Explain the purpose of user acceptance testing (UAT). How does it differ from functional testing?

  User Acceptance Testing (UAT) is the final phase of the software testing process, where **actual end-users or clients** test the software in a real-world or near-production environment to verify that it **meets business requirements and is ready for release**.

Purpose of UAT:

- Validate that the software fulfills user needs and business goals
- Detect any missing functionality or usability issues before release
- Ensure that workflows and features align with real-world use cases
- Gain final approval from stakeholders before product launch

Aspect	User Acceptance Testing (UAT)	Functional Testing
Who performs it	End-users, clients, or business stakeholders	QA testers or developers
Objective	To validate business and user requirements	To verify individual functions behave correctly
Focus	Real-world usability and business flows	Technical correctness of each function
Environment	Near-production or staging environment	Test or QA environment
Timing	Final phase before release	Earlier phase during development

- **Q13:** What is exploratory testing? How would you approach exploratory testing for a new feature in an application?

5. Defect Life Cycle and Management:

- **Q14:** What is a defect? Explain the defect life cycle, including the states a defect goes through from identification to closure.

Exploratory testing is an informal, creative, and adaptive software testing approach where testers actively explore the application **without predefined test cases**, using their knowledge, intuition, and experience to discover defects.
It is often used when:

- Requirements are incomplete
- The feature is new or experimental
- Rapid feedback is needed

How to Approach Exploratory Testing for a New Feature:

1. **Understand the Feature Briefly**
    - Read user stories, UI designs, and requirement notes (if available)
    - Talk to developers or product owners to understand the purpose
2. **Define a Charter (Mission)**

    - Example: “Explore the new search functionality for usability, accuracy, and performance”
3. **Start Exploring**
    - Interact with the feature like a real user
    - Try different inputs (valid, invalid, edge cases)
    - Change environments (browser, device, screen size)
4. **Take Notes**
    - Record what you tried, what worked, and what broke
    - Capture screenshots or videos of unexpected behavior
5. **Log Bugs or Improvements**
    - Report any issues found during exploration
    - Suggest improvements in usability or flow
6. **Review and Debrief**
    - Share your findings with the team
    - Update or create formal test cases if needed for regression

- **Q15:** Define the terms: severity and priority in defect management. How do they differ, and how do they affect the handling of defects?

Severity

- **Definition:** Severity describes the **impact** of a defect on the functionality of the application.
- **Assigned by:** QA/Testers
- **Focuses on:** How badly the system is broken, regardless of who uses it or how often.

  Example: A login button that doesn’t work at all is high severity, because it blocks user access.

Priority

- **Definition:** Priority indicates how **urgently** the defect should be fixed.
- **Assigned by:** Product managers or developers
- **Focuses on:** The business need or urgency of fixing the issue.

  Example: A typo on the home page is low severity but may be high priority before a launch.

Key Differences

Criteria	Severity	Priority
What it measures	Technical impact	Business urgency
Who decides	QA/Test Team	Product Owner / Developer
Fix order	Based on how critical the defect is	Based on deadlines or customer impact

How They Affect Defect Handling

- A **high severity, high priority** bug (e.g., application crash) is fixed immediately.
- A **low severity, high priority** bug (e.g., incorrect brand name) might be fixed before release.
- A **high severity, low priority** bug (e.g., crash in a rarely used admin panel) may be scheduled for a later fix.

- **Q16:** Imagine you found a critical bug during the testing phase. How would you document it, and what steps would you take to escalate it?
- **1. Bug Documentation:**
  Log the bug in a tracking tool (like Jira) with a clear title, steps to reproduce, expected and actual results, severity (critical), and screenshots or logs.
- **2. Validation:**
  Reproduce the issue to confirm it's consistent and affects key functionality.
- **3. Escalation:**
  Immediately inform the developer and QA/project lead. Mark it as a high-priority or blocker issue.
- **4. Tracking:**
  Monitor the bug's status, retest after the fix, and update the resolution in the system.

6. Testing Tools:

- **Q17:** What is the purpose of an automated testing tool? Name and briefly describe two popular automated testing tools used in the industry.

  1. Purpose of an Automated Testing Tool:

  Automated testing tools are used to run test cases automatically without human

intervention. They help improve **testing speed**, **accuracy**, and **efficiency**, especially during **regression testing**, by executing repetitive tasks consistently.

2. Popular Automated Testing Tools:

- **Selenium:**
  An open-source tool widely used for automating web applications across different browsers and platforms. It supports multiple programming languages like Java, Python, and C#.
- **Postman:**
  A popular tool for testing APIs. It allows developers and testers to send requests, receive responses, and automate API test cases using collections and test scripts.

- **Q18:** What is Selenium, and how is it used in automated testing? Write a simple script to test a login functionality using Selenium.
  Selenium is an open-source automated testing framework used to test web applications across different browsers (Chrome, Firefox, Edge, etc.). It allows testers to write test scripts in multiple programming languages such as Python, Java, or C#, and simulates user actions like clicking buttons, entering text, and navigating between pages.

  How Selenium is Used in Automated Testing

- Automates functional and regression tests for web apps
- Simulates real user behavior (e.g., typing, clicking)
- Runs test scripts across different browsers and OS
- Integrates with CI/CD tools for continuous testing
- Supports frameworks like TestNG, PyTest, Junit

  Simple Python Script to Test Login Functionality Using Selenium
  python
  CopyEdit

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
import time

driver = webdriver.Chrome()

driver.get("https://example.com/login")
```

```
driver.find_element(By.ID, "username").send_keys("testuser")
driver.find_element(By.ID, "password").send_keys("testpass")

driver.find_element(By.ID, "login-button").click()

time.sleep(3)
if "dashboard" in driver.current_url:
  print("Login successful.")
else:
  print("Login failed.")
driver.quit()
```

- **Q19:** Explain the concept of Continuous Integration (CI) and Continuous Testing. How do they improve the QA process?

  Continuous Integration (CI)

  Continuous Integration is a development practice where **code changes are automatically merged and tested frequently**, often multiple times a day. Developers push code to a shared repository, and an automated build and test process runs immediately to ensure the new code doesn't break existing functionality.

  Continuous Testing

  Continuous Testing is the process of **automatically running tests during every stage** of the software delivery pipeline. It ensures that code is continuously verified, allowing teams to detect bugs early and often.

  How CI and Continuous Testing Improve the QA Process

1. **Early Bug Detection:**
   Bugs are caught as soon as they are introduced, reducing costly fixes later.
2. **Faster Feedback:**
   Developers receive instant test results, speeding up development cycles.
3. **Improved Code Quality:**
   Automated tests ensure that changes maintain system stability and integrity.
4. **Reduced Manual Effort:**
   QA teams can focus on exploratory and advanced testing while automation handles repetitive checks.

7. Performance and Non-Functional Testing:

- **Q20:** What is performance testing? Name the different types of performance testing, such as load testing and stress testing.

  Performance testing is a type of software testing that evaluates how an application performs under a specific workload. It checks speed, scalability, stability, and responsiveness of the system.

  Types of Performance Testing:

1. **Load Testing:** Checks how the system behaves under expected user loads.
2. **Stress Testing:** Tests the system under extreme conditions to see when it breaks.
3. **Spike Testing:** Observes system behavior during sudden increases in user load.
4. **Endurance Testing:** Assesses performance over a long period to check memory leaks and stability.
5. **Scalability Testing:** Measures the system's ability to scale up (users, data, traffic).

- **Q21:** Explain how you would conduct load testing for a web application. What metrics would you measure during this process?

  Steps to Perform Load Testing:

1. **Identify Objectives:** Define what you want to test (e.g., login page under 500 users).
2. **Prepare Test Environment:** Ensure it's similar to production.
3. **Create Test Scripts:** Use tools like JMeter, LoadRunner, or Locust to simulate user behavior.
4. **Set Load Profiles:** Define number of users, ramp-up time, and duration.
5. **Run the Tests:** Execute the load test and observe the system behavior.
6. **Analyze Results:** Compare performance metrics against benchmarks.

  Key Metrics to Measure:

- Response Time
- Throughput (requests per second)
- Error Rate
- CPU and Memory Usage
- Concurrent User Support

- **Q22:** What is security testing, and why is it important? Provide examples of security vulnerabilities that can be tested in an application.

Security testing ensures that software is free from vulnerabilities and protects data from unauthorized access or malicious attacks.

Importance:

- Prevents data breaches
- Protects user privacy
- Ensures regulatory compliance (like GDPR, HIPAA)
- Builds user trust

Examples of Security Vulnerabilities:

- **SQL Injection:** Attacker manipulates database through form input.
- **Cross-Site Scripting (XSS):** Injecting malicious scripts into web pages.
- **Broken Authentication:** Weak or faulty login system.
- **Insecure APIs:** Poorly protected endpoints.
- **CSRF (Cross-Site Request Forgery):** Forces a user to perform unwanted actions.

8. Test Execution and Reporting:

- **Q23:** What is the difference between manual and automated testing? When would you use manual testing over automated testing?

Manual Testing:

Testing done manually by a human tester without using any scripts or automation tools. Testers follow test cases step by step to verify software functionality.

Automated Testing:

Testing performed using automation tools and scripts to execute tests faster, repeatedly, and with less human effort.

When to Use Manual Testing:

- For exploratory testing or UI/UX feedback
- When the project is short-term or changes frequently
- When tests are run once or rarely

When to Use Automated Testing:

- For regression tests, run often
- For large-scale projects with stable features
- When aiming for continuous integration/delivery (CI/CD)

- **Q24:** After executing a set of test cases, how would you report the results? What information should a test report contain?
  After running test cases, the results should be documented in a **Test Report**.

  Key Information a Test Report Should Contain:

- **Project Name and Module Tested**
- **Test Summary:** Total test cases executed, passed, failed, or blocked
- **Defects Found:** With severity and priority
- **Environment Details:** OS, browser, device, version
- **Execution Date and Tester's Name**
- **Comments or Observations**
  This helps stakeholders assess product quality and decide on release readiness.

- **Q25:** What is the purpose of a test summary report? Create a brief outline of what a test summary report should include after completing testing for a project.
  Purpose:
  A **Test Summary Report** gives a high-level overview of all testing activities completed during a project. It helps stakeholders understand the testing status, quality level, and any risks or issues.

  Brief Outline of a Test Summary Report:

1. Report Title and Version
2. Project and Release Name
3. Test Objectives
4. Test Scope and Features Tested
5. Testing Approach and Tools Used
6. Test Execution Summary
   - Total test cases executed
   - Passed/Failed/Blocked cases
7. Defect Summary
   - Open and closed defects

    - Severity levels
8. Test Environment Details
9. Risks or Limitations
10. Conclusion and Recommendation

9. Agile and QA Methodologies:

- **Q26:** What is Agile methodology? How does it impact the QA process in a software development project?
  Agile is a flexible, iterative approach to software development that focuses on delivering small, workable parts of a project through continuous collaboration, quick feedback, and rapid delivery. Work is divided into short development cycles called sprints (usually 1–4 weeks).

  Impact of Agile on the QA Process

  1. **Early and Continuous Testing**
     QA begins from the start of the project and continues in every sprint.
  2. **Testers Work Closely with Developers**
     QA is part of the Agile team, participating in daily stand-ups and sprint planning.
  3. **Frequent Changes are Welcomed**
     QA must adapt quickly to changing requirements and test new features continuously.
  4. **Automation Becomes Essential**
     Due to short sprint cycles, automated testing is often used to speed up regression and smoke tests.
  5. **User Stories Drive Testing**
     Test cases are written based on user stories and acceptance criteria defined by the product owner.

- **Q27:** Explain the concept of "Test-Driven Development" (TDD). How does TDD affect the role of a QA engineer?
  Test-Driven Development (TDD) is a development approach where developers **write test cases before writing the actual code**.
  The cycle follows:

  1. **Write a failing test**

2. **Write the minimum code** to pass the test

3. **Refactor the code** to improve structure and performance

4. **Repeat**

Impact on QA Engineer's Role:

- QA engineers may collaborate in designing test cases or acceptance criteria early.
- Encourages QA to think from a test-first mindset.
- Promotes a shared responsibility for quality between dev and QA.
- QA focuses more on exploratory, integration, and user acceptance testing, while unit tests are mostly handled by developers.

- **Q28:** In an Agile project, how is testing integrated into the sprint cycle? Describe the role of QA in sprint planning and retrospectives.

  In Agile, testing is **integrated into each sprint** and is not a separate phase.

  How Testing is Integrated:

- **Test cases are created** based on user stories before or at the start of the sprint.
- QA works in parallel with developers to test features as they are developed.
- **Automated tests** are often written and executed during the sprint.
- Bugs are reported and resolved within the same sprint, promoting fast feedback.

  QA's Role in Sprint Planning and Retrospective:

- **Sprint Planning:** QA helps estimate testing efforts, understands user stories, and defines acceptance criteria.
- **Daily Stand-ups:** QA shares testing status, blockers, and progress.
- **Sprint Retrospective:** QA provides feedback on what went well or what could be improved in the testing and development process.

10. Metrics and QA Process Improvement:

- **Q29:** What are some common QA metrics (e.g., defect density, test coverage, test execution rate)? Explain how they are used to measure the effectiveness of testing.

- **1. Defect Density**
  Measures the number of defects per size of code (e.g., per 1,000 lines).
  Use: Helps evaluate code quality.
- **2. Test Coverage**
  Percentage of code or functionalities covered by tests.
  Use: Ensures critical areas are tested, reduces untested risk.
- **3. Test Execution Rate**
  Ratio of executed test cases to total planned.
  Use: Tracks progress during a test cycle.
- **4. Defect Leakage**
  Defects missed in testing but found in production.
  Use: Indicates test effectiveness and areas needing improvement.

- **Q30:** What is the purpose of root cause analysis in QA? How do you perform a root cause analysis for a high-priority defect?

  Purpose:
  Root Cause Analysis identifies the underlying reason for a defect to prevent it from recurring.

  How to Perform RCA:

1. **Identify the Problem:** Review the defect report.
2. **Gather Data:** Analyze logs, code, and test cases.
3. **Use RCA Techniques:**
   - **5 Whys** (ask "why?" repeatedly until the root is found)
   - **Fishbone Diagram** (categorize causes: process, tools, people)
4. **Implement Fixes:** Resolve the root cause, not just the symptom.
5. **Document & Review:** Share findings with the team to improve practices.

- **Q31:** How do you measure the effectiveness of your testing process? Describe some key performance indicators (KPIs) used to evaluate the success of a QA team.

  Key Performance Indicators (KPIs):

- **Test Case Pass Rate:** Measures how many test cases passed.
- **Defect Detection Percentage:** (Defects found during testing / Total defects) × 100
- **Test Execution Time:** Time taken to run and complete test cycles.

- **Escaped Defects:** Number of bugs missed and found post-release.
- **Automation Coverage:** How much of testing is automated.
- **Reopen Rate:** Percentage of defects reopened after being marked fixed.

Use:
These KPIs help assess **test quality, team performance, and process efficiency**, leading to better product quality and continuous improvement.

11. Risk-Based Testing:

- **Q32:** What is risk-based testing, and how does it help prioritize test cases?
  Risk-Based Testing (RBT) is a testing approach where test cases are prioritized based on the risk of failure and its potential impact on the system.

  How It Helps Prioritize:
  - Focuses testing efforts on critical and high-risk areas first.
  - Helps manage limited time and resources effectively.
  - Increases the chances of detecting serious defects early.
  - Aligns testing with business priorities and user impact.

  Example:
  Login, payment, and data security modules would be tested before UI themes or optional features because they pose higher risk if they fail.

- **Q33:** Create a risk matrix for a new feature in an e-commerce application. Include factors such as impact, probability, and the risk mitigation strategy.

Risk Item	Impact	Probability	Risk Level	Risk Mitigation Strategy
Coupon not applied	5	4	20 (High)	Add test cases for valid/invalid coupons

	Risk Item	Impact	Probability	Risk Level	Risk Mitigation Strategy
	Wrong discount calculated	4	3	12 (Medium)	Validate calculation logic thoroughly
	Coupon not expired properly	3	4	12 (Medium)	Test expiry scenarios and backend logic
	Slow response during apply	3	2	6 (Low)	Optimize performance, add retry logic
	Invalid coupon crashes app	5	2	10 (Medium)	Add input validation and exception handling

12. Cross-Platform Testing:

- **Q34:** What is cross-browser testing? Why is it important, and how would you conduct such testing for a web application?

  Cross-browser testing is a type of testing where a web application is tested across multiple browsers (e.g., Chrome, Firefox, Safari, Edge) to ensure it behaves and appears consistently.

  Why It's Important:

- Different browsers render HTML, CSS, and JavaScript differently.
- Ensures consistent user experience across platforms.
- Prevents browser-specific bugs from reaching users.

How to Conduct Cross-Browser Testing:

1. Identify the most used browsers/devices among the target users.
2. Create functional and UI test cases.
3. Use tools like BrowserStack, LambdaTest, or Selenium Grid.
4. Test for layout issues, button functionality, forms, and performance.
5. Log and fix browser-specific defects.

- **Q35:** What is mobile testing, and what are the main challenges associated with it? Name a few tools used for mobile application testing.

  Mobile testing is the process of testing mobile applications (native, hybrid, or web) on different mobile devices and OS versions like Android and iOS.

  Main Challenges:

- Device fragmentation (many screen sizes, brands)
- Varying OS versions and hardware
- Limited memory and processing power
- Interruptions (calls, notifications)
- Network conditions (WiFi, 4G, airplane mode)

  Common Mobile Testing Tools:

- **Appium** (open-source, supports Android and iOS)
- **Espresso** (Android UI testing)
- **XCUITest** (iOS testing by Apple)
- **TestComplete** and **Kobiton** for advanced testing