**Introduction to Software Testing**

Software testing ensures that a software application meets requirements and is free of defects.

**What is a Quality Application?**

A quality application is reliable, secure, scalable, and meets user expectations.

**Example:** A banking app should process transactions accurately, load quickly, and be secure against unauthorized access.

**Why Deliver a Quality Application?**

- Reduces post-production defects

- Enhances user satisfaction

- Saves cost and time

**Example:** If an e-commerce app crashes during a sale, users will leave, leading to revenue loss.

**How to Deliver a Quality Application?**

- Follow SDLC and STLC processes

- Implement CI/CD pipelines

- Perform rigorous testing

**Example:** A healthcare app must pass security, performance, and functional testing before release.

**What is Software Testing?**

The process of verifying and validating a software application to detect and prevent defects.

**Example:** Testing an airline booking system to ensure correct fare calculation.

**Why is Testing Important?**

- Prevents software failures

- Improves security and performance

- Ensures compliance

**Example:** In banking, a bug in interest calculation can cause major financial losses.

**How to Do Testing?**

- Understand requirements

- Create test cases

- Execute and report defects

**Example:** Testing a mobile app’s login functionality with valid/invalid credentials.

**Testing Principles**

1. Testing shows presence of defects, not absence.

2. Exhaustive testing is impossible.

3. Early testing saves time & cost.

4. Defects cluster in specific modules.

5. Testing is context-dependent.

**Example:** In e-commerce, checkout functionality needs more testing than FAQs.

**Difference Between Product and Project**

- Product: Developed for a broad market (e.g., Microsoft Office).

- Project: Developed for a specific client (e.g., Custom ERP for a company).

**QA vs QC**

- QA (Quality Assurance): Process-oriented, prevents defects (e.g., defining test strategies).

- QC (Quality Control): Product-oriented, identifies defects (e.g., executing test cases).

**Verification vs Validation**

- Verification: Ensures software meets specifications (Static Testing).

- Validation: Ensures software meets user needs (Dynamic Testing).

**Example:**

- Verification → Code review

- Validation → UI testing in an e-commerce website

**Testing Techniques**

i. Static Testing

Reviewing documents, code walkthroughs.

**Example:** Code review in Git before merging a feature.

**ii. Dynamic Testing**

Executing the software and validating functionality.

**Example:** Running test cases on a web application.

**iii. Differences**

- Static: Prevent defects (before execution).

- Dynamic: Detect defects (during execution).

**Testing Methodologies**

i. Whitebox Testing

Testing internal logic with code knowledge.

**Example:** Unit testing a function in Java.

**ii. Blackbox Testing**

Testing without knowledge of internal code.

**Example:** Checking login with correct/incorrect credentials.

**iii. Greybox Testing**

Combination of both.

Example: Testing API responses with database validation.

**iv. Differences**

- Whitebox: Developer-focused

- Blackbox: User-focused

**Testing Levels**

i. Unit Testing

Testing individual components.

**Example:** Testing a login function in Java.

**ii. Integration Testing**

Testing interaction between modules.

**Example:** Ensuring payment API integrates correctly with the order system.

**iii. System Testing**

Testing the entire application.

Example: Verifying an e-commerce website end-to-end.

**iv. System Integration Testing**

Ensuring external system integrations work.

**Example:** Checking PayPal payment processing in an online store.

**v. User Acceptance Testing (UAT)**

End users verify the application.

**Example:** Bank employees testing a new banking portal before launch.

**Testing Types**

**1. Functional Testing**

Validates the application against business requirements.

**Example:** Testing the password reset feature of an app.

**2. Non-Functional Testing**

Checks performance, security, and usability.

**i. Performance Testing**

Checking speed, scalability.

**Example:** Load testing an airline ticket booking site during a sale.

**ii. Security Testing**

Identifying vulnerabilities.

**Example:** Checking SQL injection vulnerabilities in login.

**iii. Compatibility Testing**

Ensuring application works across different OS/devices.

**Example:** Testing an app on iOS and Android.

**iv. Usability Testing**

Checking user-friendliness.

**Example:** Verifying intuitive navigation in a food delivery app.

**v. Localization Testing**

Ensuring app adapts to different languages/cultures.

**Example:** A travel app displaying correct currency and date formats.

**Testing Methods**

**1. Smoke Testing**

Basic check to see if major functionalities work.

**Example:** Checking login, dashboard, and logout after a new build.

**2. Sanity Testing**

Focused testing on a specific fix.

**Example:** Verifying if a login issue is resolved after a bug fix.

**3. Retesting**

Re-executing failed test cases.

**Example:** Running a failed checkout test again after fixing a bug.

**4. Regression Testing**

Ensuring new changes don’t break existing features.

**Example:** Verifying the search function still works after updating the homepage.

**5. Adhoc Testing**

Random testing without predefined test cases.

**Example:** Clicking around an app to find unexpected issues.

**6. Exploratory Testing**

Simultaneous learning and testing.

**Example:** Exploring a new analytics dashboard without documentation.

**7. Positive Testing**

Checking valid inputs.

**Example:** Entering correct username/password and verifying successful login.

**8. Negative Testing**

Testing the application with invalid inputs.

**Example:** Entering special characters in a phone number field and verifying error handling.

**9. Alpha Testing**

Testing performed by internal teams before public release.

**Example:** A new mobile app is tested by the in-house QA team before launching for beta users.

**10. Beta Testing**

Testing done by real users in a production-like environment.

**Example:** WhatsApp releases a beta version to selected users for feedback before public launch.

**11. Manual Testing**

Executing test cases without automation tools.

**Example:** Checking if an e-commerce site’s checkout process works by manually adding items and making a purchase.

**12. Automation Testing**

Using scripts and tools to execute test cases.

**Example:** Running Selenium scripts to validate the login functionality across different browsers.

**Software Testing Life Cycle (STLC)**

**a. Requirement Analysis**

Understanding requirements before writing test cases.

**Example:** Checking if a banking app should support two-factor authentication.

**b. Test Plan**

A document outlining test strategy, scope, and schedule.

**Example:** A test plan for a hospital management system defining which modules need testing.

**c. Test Design**

Creating test scenarios and cases.

**Test Scenario**

High-level functionality to be tested.

**Example:** Verify that a user can successfully reset their password.

**Test Case**

Step-by-step procedure to test a scenario.

**Example:**

1. Go to the login page.

2. Click "Forgot Password."

3. Enter a registered email.

4. Verify the reset link is received.

**iii. Test Review**

Reviewing test cases for completeness and correctness.

Example: Team lead reviews test cases for an online ticket booking system.

**iv. Test Case Design Techniques**

- Equivalence Partitioning: Test with one valid and one invalid input per category.

- Boundary Value Analysis: Testing limits (e.g., age field with 17, 18, 100).

- Decision Table Testing: Used for complex logic-based applications.

**d. Test Environment Setup (Test Bed)**

**Preparing the environment for testing.**

Example: Setting up a test database for an airline reservation system.

**e. Test Execution**

**i. Test Case Execution**

Running test cases manually or via automation.

**ii. Defect Creation**

Logging defects found during execution.

**iii. Defect Life Cycle**

1. New → 2. Assigned → 3. In Progress → 4. Fixed → 5. Retested → 6. Closed/Reopened

**f. Test Closure**

Summarizing the testing process, lessons learned, and defects identified.

**RTM (Requirement Traceability Matrix)**

Mapping requirements to test cases to ensure coverage.

Example: Verifying each user requirement in a banking app has a corresponding test case.

**ii. Test Report**

A summary of executed test cases and defects found.

Example: A report showing that login functionality failed 2 out of 5 test cases.

**iii. Defect Report**

A detailed log of identified defects.

**g. Difference Between Mistake, Error, Defect, Bug, and Failure**

- Mistake: Human error in coding (e.g., developer types `>` instead of `<`).

- Error: Incorrect code that results in unexpected behavior.

- Defect/Bug: A mismatch between expected and actual output.

- Failure: When a defect is encountered in production.

**h. Defect Types**

**1. Defect Leakage**

A defect missed in testing but found in production.

Example: A mobile app crashes for users after deployment due to an untested edge case.

**2. Defect Masking**

A defect is hidden by another defect.

Example: A login page defect prevents users from reaching the dashboard, hiding an existing defect in the dashboard.

**3. Defect Cascading**

One defect triggers multiple defects in later stages.

Example: A miscalculation in a billing system leads to incorrect invoices and failed payments.

**Software Development Life Cycle (SDLC)**

**Introduction to SDLC Phases**

**i. Planning**

Defining project scope, objectives, and feasibility.

Example: A fintech company planning a new loan application system.

**ii. Requirements Gathering & Analysis**

Identifying system requirements from stakeholders.

Example: A retail app requires barcode scanning for inventory management.

**iii. Design**

Architectural and UI/UX design.

Example: Designing database structure for an HR management system.

**iv. Development**

Writing code based on design specifications.

Example: Developers implement APIs for an online food delivery app.

**v. Testing**

Verifying the software meets requirements.

Example: QA team tests an online exam portal before deployment.

**vi. Deployment**

Releasing software for end users.

Example: A new banking app goes live in the App Store.

**vii. Maintenance**

Bug fixes and performance enhancements post-release.

Example: A social media app fixes a login issue after a software update.

**SDLC Approaches**

**i. Sequential Approach**

A step-by-step linear approach (e.g., Waterfall Model).

**ii. Iterative Approach**

Repeated cycles of development (e.g., Agile Model).

**iii. Incremental Approach**

Developing in small parts and integrating them later.

**SDLC Models**

**i. Waterfall Model**

A linear model where each phase follows sequentially.

**Advantages:**

- Simple and structured.

- Easy to manage for small projects.

**Disadvantages:**

- No flexibility for changes.

- Late testing leads to defect discovery at later stages.

**ii. V-Model (Verification and Validation Model)**

Testing is done in parallel with development.

**Advantages:**

- Defects are identified early.

- Test planning starts in early phases.

**Disadvantages:**

- Not suitable for dynamic changes.

- Rigid and costly for iterative modifications.

**iii. Differences Between Waterfall and V-Model**

**| Feature | Waterfall Model | V-Model |**

| Testing Phase | After development | Parallel to development |

| Flexibility | Rigid | More flexible |

| Cost of Defects | High if found late | Lower due to early testing |