**Software Testing Basics - Study Material**

**Introduction to Software Testing**

Software testing is a process of evaluating and verifying that a software application meets the required specifications and works correctly. The goal of testing is to identify defects, ensure software reliability, and improve performance.

**Why is Software Testing Important?**

* Ensures software quality and functionality.
* Detects and fixes defects early.
* Improves user experience.
* Reduces development costs by avoiding later-stage issues.
* Enhances security and prevents vulnerabilities.

**1. Manual Testing**

Manual testing is the process of testing software manually without the use of automation tools. Testers execute test cases, compare actual results with expected outcomes, and report defects if found.

**Example:**

Imagine you have an e-commerce website. A manual tester will check if the user can:

* Log in successfully.
* Add products to the cart.
* Proceed to checkout without issues.
* Receive a confirmation email after purchase.

**Advantages:**

* Simple and does not require programming knowledge.
* Allows exploratory testing.
* Effective for small projects.

**Disadvantages:**

* Time-consuming and repetitive.
* Less reliable due to human errors.

**2. Agile Testing**

Agile testing follows the Agile methodology, where testing is integrated into the software development lifecycle. Testers work closely with developers in iterative cycles to deliver high-quality software.

**Example:**

A team developing a mobile app releases small updates every two weeks. Testers continuously test new features and provide feedback, ensuring faster and more reliable releases.

**Advantages:**

* Faster feedback and defect fixing.
* Better collaboration between teams.
* Continuous improvement.

**3. API Testing**

API testing involves verifying application programming interfaces (APIs) to ensure they function correctly. APIs enable communication between software applications.

**Example:**

A weather application fetches temperature data from a remote server using an API. API testing ensures that:

* The API returns correct weather data.
* It handles incorrect requests properly.
* It performs efficiently under load.

**Tools Used:**

* Postman
* REST Assured
* SoapUI

**4. Performance Testing**

Performance testing evaluates how well software performs under different conditions, such as load and stress testing.

**Example:**

An e-commerce website experiences high traffic during a sale event. Performance testing helps ensure that the website can handle 10,000 users simultaneously without crashing.

**Types:**

* Load Testing (checks system behavior under expected load).
* Stress Testing (tests system performance under extreme conditions).

**5. Test Automation**

Test automation uses scripts and tools to execute test cases automatically, reducing manual effort and improving efficiency.

**Example:**

Instead of manually testing a login page every time a change is made, automation scripts can be created to perform the testing.

**Tools Used:**

* Selenium
* JUnit
* TestNG

**Conclusion**

Understanding different types of testing is crucial for a software tester. Depending on the job role, interview questions may focus on manual testing, Agile methodologies, API testing, performance testing, or test automation. Preparing for relevant topics based on the job requirements will increase the chances of success in an interview.

This study material provides an easy-to-understand introduction to software testing, including examples and explanations for beginners. Let me know if you need additional topics or a deeper explanation!

**Study Material: SDLC vs. STLC & Software Testing Levels**

**1. Difference Between SDLC and STLC**

**What is SDLC (Software Development Life Cycle)?**

SDLC is a structured process that software developers follow to design, develop, and test high-quality software. It ensures that the final product meets customer expectations and is delivered within the expected timeline and budget.

**Phases of SDLC:**

1. **Requirement Analysis** – Understanding the needs of the client.
2. **Planning** – Creating a roadmap for development.
3. **Design** – Structuring the software (architecture and UI/UX design).
4. **Implementation (Coding)** – Writing the actual program.
5. **Testing** – Ensuring functionality, security, and performance.
6. **Deployment** – Releasing the software to the users.
7. **Maintenance** – Handling updates and bug fixes.

**What is STLC (Software Testing Life Cycle)?**

STLC focuses only on the testing process within the SDLC. It ensures that software is defect-free before deployment.

**Phases of STLC:**

1. **Requirement Analysis** – Understanding what needs to be tested.
2. **Test Planning** – Defining the testing strategy.
3. **Test Case Development** – Creating detailed test scenarios.
4. **Test Environment Setup** – Preparing the testing infrastructure.
5. **Test Execution** – Running test cases and reporting defects.
6. **Test Closure** – Final evaluation and documentation.

**Key Differences Between SDLC and STLC**

| **Aspect** | **SDLC (Software Development Life Cycle)** | **STLC (Software Testing Life Cycle)** |
| --- | --- | --- |
| Purpose | Focuses on software development | Focuses on software testing |
| Phases | Includes design, development, and testing | Includes only testing-related activities |
| Performed By | Developers, project managers | Testers, QA engineers |
| Output | Functional software product | Defect-free, verified software |

**2. Levels of Software Testing**

Software testing is categorized into four main levels:

**1. Unit Testing (Component Testing)**

* **Objective:** Verify that individual components (units) of the software function correctly.
* **Who Performs It?** Developers.
* **Example:** Testing a single function that calculates user age from a birthdate.

**2. Integration Testing**

* **Objective:** Ensure that different modules or components work together as expected.
* **Who Performs It?** Developers or testers.
* **Example:** Testing the interaction between a login module and a database.

**3. System Testing**

* **Objective:** Validate that the entire software system meets the specified requirements.
* **Who Performs It?** Testers.
* **Example:** Checking whether an e-commerce website functions correctly end-to-end.

**4. Acceptance Testing**

* **Objective:** Verify that the software is ready for deployment and meets business needs.
* **Who Performs It?** End-users or stakeholders.
* **Example:** A client testing a mobile app before approving its launch.

**3. Defect Life Cycle (Bug Life Cycle)**

A defect (bug) goes through different stages from discovery to resolution:

| **Stage** | **Description** |
| --- | --- |
| **New** | The tester identifies and logs a new defect. |
| **Assigned** | The defect is assigned to a developer for fixing. |
| **In Progress** | The developer starts working on the fix. |
| **Fixed** | The defect is resolved by the developer. |
| **Pending Retest** | The tester verifies the fix. |
| **Retest** | The defect is tested again to confirm resolution. |
| **Closed** | If the defect is resolved, it is marked as closed. |
| **Reopened** | If the defect still exists, it is reopened and reassigned. |

**Example Scenario:**

1. A tester finds that the login button does not work and reports it (New).
2. The defect is assigned to a developer (Assigned).
3. The developer fixes the issue (Fixed).
4. The tester retests it and finds the issue is gone (Retest & Closed).

This study material provides a clear and simple understanding of SDLC, STLC, testing levels, and defect management for beginners.

**4. Types of Software Testing**

Software testing is broadly categorized into functional and non-functional testing. Below are the different types:

**Functional Testing:**

1. **Smoke Testing** – Quick checks to ensure major functionalities work.
2. **Sanity Testing** – Focused checks after minor changes to verify fixes.
3. **Regression Testing** – Ensuring new updates do not break existing functionalities.
4. **Usability Testing** – Verifying user-friendliness of the application.
5. **User Acceptance Testing (UAT)** – Confirming the software meets business needs before launch.

**Non-Functional Testing:**

1. **Performance Testing** – Checking speed and responsiveness under load.
2. **Load Testing** – Testing software behavior under heavy usage.
3. **Stress Testing** – Evaluating performance under extreme conditions.
4. **Security Testing** – Ensuring protection against vulnerabilities.
5. **Compatibility Testing** – Verifying software performance across different devices and browsers.

This section ensures a better understanding of various testing methodologies.

**Basic Study Material for Software Testing**

**1. Difference Between Test Case and Test Scenario**

**Definition:**

* **Test Scenario:** A high-level concept of what needs to be tested.
* **Test Case:** A detailed set of actions performed to verify a particular functionality.

**Example:**

**Test Scenario:**

* Verify the login functionality.

**Test Case:**

| **Step No.** | **Action** | **Expected Result** |
| --- | --- | --- |
| 1 | Open the login page | Login page should be displayed |
| 2 | Enter valid username/password | User should be logged in |
| 3 | Click the login button | Redirected to home page |

**Diagram:**

Test Scenario → Multiple Test Cases

**2. Test Analysis and Test Design**

**Definition:**

* **Test Analysis:** Identifying what to test.
* **Test Design:** Defining how to test the identified scenarios.

**Example:**

1. **Test Analysis:** Identify that login functionality needs to be tested.
2. **Test Design:** Write detailed test cases to verify login works correctly.

**Work Products:**

* **Test Analysis:** List of test scenarios.
* **Test Design:** Detailed test cases.

**3. Functional vs Non-Functional Testing**

**Definition:**

* **Functional Testing:** Tests what the system does.
* **Non-Functional Testing:** Tests how well the system performs.

**Examples:**

| **Type** | **Example** |
| --- | --- |
| Functional Testing | Login verification |
| Performance Testing | Checking system response time |
| Usability Testing | Verifying user-friendliness |
| Security Testing | Ensuring no unauthorized access |

**Diagram:**

Software Testing

├── Functional Testing

├── Non-Functional Testing

├── Performance Testing

├── Usability Testing

├── Security Testing

This material provides a basic understanding of software testing concepts in a simple manner. Let me know if you need further explanations or modifications!

**Basic Study Material on Software Testing**

**1. When Should We Start Testing in Our Project?**

**Definition:**

Software testing should begin as early as possible in the software development lifecycle (SDLC). The earlier we start testing, the easier it is to identify and fix defects, reducing overall project costs.

**Explanation:**

* Testing starts from the **requirement gathering phase**.
* The role of a tester begins by reviewing requirements to identify ambiguities or contradictions.
* Early testing helps in **reducing defects** in later stages of development.

**Example:**

Imagine a client wants a banking application. If testing starts only after coding, major defects in requirements might be overlooked. If testers review requirements early, they might catch inconsistencies like:

* Should a password reset be allowed without an OTP?
* Should there be a limit on failed login attempts?

**Diagram: Software Development Lifecycle with Testing**

| **Phase** | **Activity** | **Role of Testing** |
| --- | --- | --- |
| Requirement Gathering | Understanding needs | Reviewing requirements |
| Design | Creating UI/UX, Architecture | Ensuring logical consistency |
| Development | Writing code | Unit testing, early bug detection |
| Testing | Running test cases | Identifying defects |
| Deployment | Releasing software | Final verification |
| Maintenance | Updates and fixes | Continuous testing |

**Key Benefit:**

Early testing saves **time and money** as per the **seven principles of software testing**.

**2. How Can We Test Software Without Clear Written Requirements?**

**Definition:**

If formal requirements are missing, testers use alternative approaches to understand system behavior and validate functionality.

**Methods to Overcome Lack of Requirements:**

1. **Use Available Documentation**: Even if outdated, use old requirements, UI designs, prototypes, or wireframes.
2. **Refer to Older Versions**: If the application has an older version, compare and test based on it.
3. **Communicate with Stakeholders**: Ask developers, business analysts, and product owners about requirements.
4. **Use Exploratory Testing**: Interactively test the system by exploring its functionalities.

**Example:**

If a company has no written login requirements but an old version of the app is available, testers can:

* Check login behavior in the old version.
* Verify if the new version follows expected behaviors.
* Ask developers about new feature changes.
* Perform exploratory testing to uncover potential defects.

**3. What is Exploratory Testing?**

**Definition:**

Exploratory testing is a software testing approach where test cases are not predefined. Instead, testers learn the software while testing it, designing test cases on the go.

**Why Use Exploratory Testing?**

* **When requirements are unclear or missing.**
* **When time is limited.**
* **To find unexpected defects through intuitive exploration.**

**Process of Exploratory Testing:**

1. **Learn**: Understand the application by exploring.
2. **Design Tests**: Identify test scenarios dynamically.
3. **Execute**: Perform testing and note observations.
4. **Analyze Results**: Identify defects and document insights.

**Example:**

A tester explores a newly developed e-commerce site and notices that:

* The "Add to Cart" button does not work when filters are applied.
* Discounts are not applied at checkout despite being shown in the cart. This issue may not have been covered by predefined test cases.

**Diagram: Exploratory Testing Cycle**

+------------+ +-------------+

| Learn | ---> | Design Tests |

+------------+ +-------------+

| |

v v

+------------+ +-------------+

| Execute | ---> | Analyze |

+------------+ +-------------+

**Session-Based Exploratory Testing (SBTM):**

* Time-boxed testing (e.g., 60 or 90 minutes per session).
* Each session focuses on a specific feature.
* Uses a **test charter** (a checklist of areas to explore).

**Key Benefit:**

Exploratory testing helps uncover hidden issues and provides quick feedback in the absence of formal test cases.

**Conclusion**

* Start testing **early** to prevent defects and reduce costs.
* In the absence of written requirements, use available documents, communicate with stakeholders, and perform exploratory testing.
* Exploratory testing is valuable when time is short or requirements are unclear, helping identify unexpected defects efficiently.

This guide provides a **basic understanding** of software testing, making it easy for beginners to grasp key concepts with examples, diagrams, and structured explanations. 🚀

**Study Material for Basic Software Testing Concepts**

**1. Impact of Late Defect Removal on Cost**

**Definition**

A defect that could have been removed during the early stages of software development but is instead removed at a later stage increases the cost of fixing it.

**Explanation with Example**

Consider a software development process with the following stages:

1. **Requirement Gathering**
2. **Design**
3. **Development (Coding)**
4. **Testing**
5. **Deployment (Operation)**

If a defect is found at the **Requirement Gathering** stage, it can be fixed by modifying the document, which is inexpensive. However, if the defect is found after **Deployment**, fixing it involves high costs, as the software is already in use by customers.

**Cost Impact Table**

| **Stage Found** | **Cost to Fix (Relative)** |
| --- | --- |
| Requirement | Low |
| Design | Low |
| Development | Moderate |
| Testing | High |
| Deployment | Very High |

**Graph Representation**

A typical graph shows that as the defect is detected later in the cycle, the cost exponentially increases.

**Conclusion:** Early defect detection saves cost and effort.

**2. Change-Related Testing**

**Definition**

Change-related testing refers to the testing performed after modifications are made to a software application to ensure that the changes do not introduce new defects.

**Types of Change-Related Testing**

1. **Confirmation Testing (Retesting)**
2. **Regression Testing**

**Confirmation Testing (Retesting)**

* When a defect is fixed, the same test case is executed again to verify that the defect is truly resolved.
* **Example:** A login issue is reported and fixed. The tester then re-executes the login test to confirm the fix.

**Regression Testing**

* Ensures that recent code changes do not adversely affect existing functionalities.
* **Example:** If the login module is modified, regression testing will check other dependent functionalities, such as profile access and logout.

**Impact Analysis in Regression Testing**

* Determines the extent of changes required and the scope of testing needed.
* Helps estimate the cost and effort required for regression testing.

**3. Black-Box, White-Box, and Grey-Box Testing**

**Black-Box Testing**

* The tester does not have knowledge of the internal workings of the software.
* **Example:** A tester enters a username and password in a login form and verifies if login succeeds, without knowing how the backend processes the input.
* **Best For:** Functional Testing

**White-Box Testing**

* The tester has access to the internal code and logic of the software.
* **Example:** A developer tests an algorithm to ensure all logic branches execute correctly.
* **Best For:** Unit Testing

**Grey-Box Testing**

* The tester has partial knowledge of the internal structure of the software.
* **Example:** A tester checks database queries while testing an application’s UI behavior.
* **Best For:** Integration Testing

**Comparison Table**

| **Testing Type** | **Knowledge of Code** | **Who Performs It?** | **Example** |
| --- | --- | --- | --- |
| Black-Box | No | Testers | Testing login functionality |
| White-Box | Yes | Developers | Code coverage testing |
| Grey-Box | Partial | Testers & Developers | Checking API interactions |

**Conclusion**

Understanding different testing techniques and the impact of defects at different stages is crucial for effective software development. Early defect detection reduces costs, and proper testing strategies ensure software reliability and efficiency.

**Study Material for Basic Level Users**

**1. Understanding Test Cases: Black-Box vs. White-Box Testing**

**What is Black-Box Testing?**

* Black-box testing is a testing technique where the internal workings of the application are not known to the tester.
* Test cases are designed based on user requirements and system functionality.
* Example: Checking if a login page accepts correct username and password without knowing how the backend processes them.

**What is White-Box Testing?**

* White-box testing is a testing technique where the internal structure of the application is known to the tester.
* It involves testing code, logic, and integration between components.
* Example: A developer writing test cases to check if each function in a program returns the correct value.

**Order of Writing Test Cases**

* **Black-box test cases** are generally written first because they are based on user requirements and system specifications.
* **White-box test cases** are written later, often by developers, during coding or unit testing.

**2. V-Model in Software Testing**

* The V-Model is a software development methodology where each phase has a corresponding testing phase.
* Steps:
  1. Requirements -> **Acceptance Testing**
  2. System Design -> **System Testing**
  3. Architectural Design -> **Integration Testing**
  4. Coding -> **Unit Testing**
* **Writing Test Cases Order:** Acceptance Testing -> System Testing -> Integration Testing -> Unit Testing.
* **Execution Order:** Unit Testing -> Integration Testing -> System Testing -> Acceptance Testing.

**3. Use Case Testing**

**What is a Use Case?**

* A use case is a description of how a user interacts with a system to achieve a goal.
* It includes actors (users or external systems) and steps to complete a function.

**Example of Use Case Diagram**

| **Actor** | **System** | **Interaction** |
| --- | --- | --- |
| User | Chat System | Logs in, sends messages |
| Remote User | Remote System | Connects and interacts |

* **Use Case Testing** ensures that all interactions in a system work as expected from start to finish.

**4. Equivalence Partitioning vs. Boundary Value Analysis**

**Equivalence Partitioning**

* A testing technique that divides input data into partitions where each partition should behave the same way.
* Example:
  + If an application accepts ages 18-60, equivalence partitions are:
    - Invalid: Below 18, Above 60
    - Valid: 18-60 (Only one test case from this group is needed)

**Boundary Value Analysis**

* Focuses on testing the boundaries of input values.
* Example:
  + If valid ages are 18-60:
    - Test at **boundary values**: 17, 18, 60, 61.
* Boundary values are more likely to have errors than middle values.

| **Technique** | **When to Use** | **Example** |
| --- | --- | --- |
| Equivalence Partitioning | When dividing input data into groups | Test any number within 18-60 |
| Boundary Value Analysis | When input data is numeric or sequential | Test 17, 18, 60, 61 |

**Key Differences**

* **Equivalence Partitioning:** Tests one value per group.
* **Boundary Value Analysis:** Tests values at the edge of each group.

**5. Importance of Test Scenarios and Test Cases**

**What is a Test Scenario?**

* A high-level description of what needs to be tested.
* Example: Verify login functionality.

**What is a Test Case?**

* A detailed step-by-step document specifying inputs, execution conditions, and expected results.
* Example:
  + **Test Scenario:** Verify login functionality.
  + **Test Case:**
    1. Open the login page.
    2. Enter valid username and password.
    3. Click the login button.
    4. Verify successful login message.

**Difference Between Test Scenario and Test Case**

| **Aspect** | **Test Scenario** | **Test Case** |
| --- | --- | --- |
| Level of Detail | High-level | Detailed steps |
| Example | Verify login functionality | Enter valid credentials and check login success |

**Summary**

* **Black-box testing** is based on system functionality, **white-box testing** is based on internal code structure.
* **V-Model** organizes testing at different stages of development.
* **Use case testing** ensures all interactions work as expected.
* **Equivalence partitioning** divides input data into groups, while **boundary value analysis** tests values at the edges.
* **Test scenarios** describe what to test, while **test cases** provide detailed steps.

By understanding these basic concepts, beginners can effectively approach software testing and improve the quality of applications.

**Study Material for Basic Level Users**

**1. Requirements Traceability Matrix (RTM)**

**What is RTM?**

Requirements Traceability Matrix (RTM) is a document that connects requirements with other software development artifacts, such as test cases, defects, and other documents. It ensures that every requirement is verified through proper testing.

**Example of RTM:**

| **Requirement ID** | **Requirement Description** | **Test Case ID** | **Test Case Description** | **Status** |
| --- | --- | --- | --- | --- |
| R1 | User should be able to log in | TC1 | Verify login with valid credentials | Passed |
| R1 | User should be able to log in | TC2 | Verify login with incorrect password | Failed |
| R2 | User can reset the password | TC3 | Verify password reset feature | Passed |

**Types of Traceability:**

1. **Forward Traceability**: Links requirements to test cases.
2. **Backward Traceability**: Links test cases back to requirements.
3. **Bidirectional Traceability**: Ensures that every requirement is covered by test cases and every test case is linked to a requirement.

**Why is RTM Important?**

* Ensures complete test coverage.
* Helps in identifying missing test cases.
* Tracks changes in requirements effectively.

**2. Static vs Dynamic Testing**

**What is Static Testing?**

Static testing is a method of software testing where the code is reviewed without executing it. It helps in identifying issues early in the development phase.

**Examples of Static Testing:**

* Code Reviews
* Walkthroughs
* Inspection
* Static Analysis using tools (e.g., SonarQube, ESLint)

**What is Dynamic Testing?**

Dynamic testing is performed by executing the software to validate the functionality of the application.

**Examples of Dynamic Testing:**

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

**Comparison Table:**

| **Feature** | **Static Testing** | **Dynamic Testing** |
| --- | --- | --- |
| Execution Required | No | Yes |
| Detects Errors Early | Yes | No |
| Cost-Effective | More | Less |
| Types | Code Reviews, Walkthroughs | Unit Testing, System Testing |

**3. Test Plan**

**What is a Test Plan?**

A test plan is a document that defines the scope, objectives, approach, and resources required for testing a software application.

**Key Components of a Test Plan:**

1. **Test Scope**: Defines what will be tested.
2. **Test Objectives**: Specifies the goals of testing.
3. **Resources Required**: Lists required tools, hardware, and software.
4. **Test Environment**: Specifies where the testing will take place.
5. **Test Schedule**: Provides a timeline for testing activities.
6. **Entry & Exit Criteria**:
   * **Entry Criteria**: Conditions that must be met before testing begins.
   * **Exit Criteria**: Conditions that determine when testing is complete.
7. **Risk Management**: Identifies potential risks and their impact.

**Types of Test Plans:**

1. **Master Test Plan**: Covers all testing activities for a project.
2. **Phase Test Plan**: Separate plans for unit, integration, system, and acceptance testing.
3. **Type-Specific Test Plans**:
   * Performance Testing Plan
   * Security Testing Plan
   * Functional Testing Plan

**Example of a Simple Test Plan:**

| **Test Phase** | **Activities** | **Responsible Person** | **Status** |
| --- | --- | --- | --- |
| Unit Testing | Test individual components | Developer | Completed |
| Integration Testing | Test module interaction | Tester | In Progress |
| System Testing | Test the complete system | QA Team | Pending |
| UAT | User-based testing | Client | Not Started |

**Why is a Test Plan Important?**

* Provides a structured approach to testing.
* Ensures all functionalities are tested.
* Helps in resource management.
* Identifies potential risks early.

This study material provides a basic understanding of key testing concepts with examples, tables, and diagrams where necessary. Let me know if you need more details or modifications!

**Study Material: Basic Understanding of Testing Reports and Common Tester Mistakes**

**1. Difference Between Test Progress Report and Test Summary Report**

**What is a Test Progress Report?**

A **Test Progress Report** is a document prepared **during the testing phase** to keep stakeholders informed about the current status of the testing activities.

**Purpose:**

* To provide updates about what has been tested so far.
* To inform what will be tested next.
* To highlight any issues blocking the testing process.
* Helps in tracking progress and identifying bottlenecks.

**Example:**

Imagine you are performing **system testing** for an application. Since system testing can take weeks, you need to regularly update the project manager, team leader, or client about the progress.

A **weekly Test Progress Report** may include:

| **Report Date** | **Test Cases Executed** | **Test Cases Passed** | **Test Cases Failed** | **Issues Identified** |
| --- | --- | --- | --- | --- |
| March 1 | 50 | 45 | 5 | 2 Critical, 3 Minor |
| March 8 | 100 | 90 | 10 | 4 Critical, 6 Minor |

**What is a Test Summary Report?**

A **Test Summary Report** is a document prepared **after the testing phase is complete** to summarize the overall results of the testing process.

**Purpose:**

* Provides a final status of testing efforts.
* Summarizes key findings, test coverage, and defects found.
* Helps stakeholders decide whether the software is ready for release.

**Example:**

If you have completed system testing, your **Test Summary Report** will include:

* Number of test cases executed.
* Pass/fail ratio.
* Major defects found and their resolution status.
* Overall quality of the software.

| **Total Test Cases** | **Passed** | **Failed** | **Critical Defects** | **Minor Defects** | **Final Status** |
| --- | --- | --- | --- | --- | --- |
| 500 | 450 | 50 | 10 | 40 | Ready for Release with Minor Fixes |

**Key Differences:**

| **Feature** | **Test Progress Report** | **Test Summary Report** |
| --- | --- | --- |
| When Prepared? | During Testing | After Testing Completion |
| Purpose | Updates on ongoing tests | Summary of testing efforts |
| Frequency | Daily/Weekly/Bi-weekly | One-time (End of Testing) |
| Content | What is tested, pending tests, issues | Overall results, defects, pass/fail rate |

**2. Common Mistakes Testers Make**

**1. Poor Communication**

* Testers need to **communicate effectively** with developers, managers, and stakeholders.
* **Example:** If a tester finds a defect but doesn’t explain it properly, the developer may not understand the issue.

**2. Being Afraid to Ask Questions**

* A good tester should **ask questions** like:
  + Why is this feature implemented this way?
  + What is the expected behavior?
  + Are there any hidden requirements?
* **Example:** If a tester assumes how a feature should work without confirming, they might miss critical defects.

**3. Testing Without Understanding Requirements**

* **Before testing, testers should understand:**
  + The system functionality
  + User requirements
  + Business needs
* **Example:** If a tester doesn’t analyze the requirements, they might miss important test cases.

**4. Writing Poor Defect Reports**

* A defect report should be **clear, concise, and detailed**.
* **Bad Example:** “Login button is not working.”
* **Good Example:** “Clicking the login button does not redirect users to the dashboard; instead, it refreshes the page without logging in. Steps to reproduce: 1. Enter valid credentials, 2. Click login, 3. Observe the issue.”

**5. Missing Requirements While Writing Test Cases**

* Ensure that all requirements are covered using a **Requirement Traceability Matrix (RTM)**.

| **Requirement ID** | **Requirement Description** | **Test Case ID** | **Covered?** |
| --- | --- | --- | --- |
| RQ-001 | User should be able to log in | TC-001 | Yes |
| RQ-002 | Password should be encrypted | TC-005 | No |

**6. Ad-hoc Testing Without a Plan**

* **Monkey Testing (Random Clicks)** is not effective.
* Always follow a **structured testing approach** with a test plan or checklist.

**7. False Positives and False Negatives**

* **False Positive:** Reporting a defect that is NOT a defect.
  + **Example:** Tester reports “Login button not working,” but the issue was caused by an internet connection problem.
* **False Negative:** Missing a defect that actually exists.
  + **Example:** A security vulnerability exists, but the tester did not test for it.

**8. Handling Defect Rejection by Developers**

If a developer rejects a defect, follow these steps:

* **Communicate and Provide Evidence:** Show logs, screenshots, or videos of the issue.
* **Refer to Documentation:** Check if the defect contradicts system requirements.
* **Ask the Product Owner:** If unclear, ask the product owner for clarification.
* **Check Test Environments:** Ensure both tester and developer are using the same environment.
* **Escalate if Necessary:** If the issue persists, involve a manager.

**Example Scenario:**

1. Tester reports a defect where the login button is not working.
2. Developer rejects the defect, saying it works fine.
3. Tester checks and finds that the issue occurs only in a specific browser.
4. Tester provides screenshots and error logs.
5. Developer acknowledges the issue and fixes it.

**Conclusion**

* **Test Progress Reports** help track ongoing testing efforts.
* **Test Summary Reports** provide a final overview of testing results.
* Avoid common testing mistakes like poor communication, lack of planning, and false defect reports.
* When defects are rejected, use logical steps to validate and escalate if needed.

By following these best practices, testers can improve the quality of software testing and ensure a smoother development process.

**Study Material: Understanding Alpha Testing, Beta Testing, and Test Independence**

**1. Alpha Testing vs. Beta Testing**

**What is Testing?**

Testing is the process of evaluating software to ensure that it meets user requirements and is free of defects.

**What is Acceptance Testing?**

Acceptance testing is the final phase of software testing before the product is released. It ensures that the software meets business and customer requirements.

Two common types of acceptance testing are **Alpha Testing** and **Beta Testing**.

**Alpha Testing**

**Definition:** Alpha testing is performed at the developer's site before the product is released to real users. The goal is to find bugs before releasing the software to a wider audience.

**Key Features:**

* Conducted in a controlled environment.
* Involves in-house testers or a selected group of potential users.
* Feedback is collected directly from testers.
* Helps detect bugs before public release.

**Example:** A gaming company developing a new PlayStation game invites a small group of testers to their office to play the game and report any issues before launching it.

**Beta Testing**

**Definition:** Beta testing is conducted after alpha testing and is performed by real users in a real environment. It helps gather user feedback and discover issues that may not have been found in alpha testing.

**Key Features:**

* Conducted at users' locations.
* Open to a larger audience.
* Helps identify real-world performance and usability issues.

**Example:** A software company releases a "Beta Version" of a mobile app on the Play Store for users to test before launching the final version.

**Comparison Table**

| **Feature** | **Alpha Testing** | **Beta Testing** |
| --- | --- | --- |
| Performed by | In-house testers | Real users |
| Location | Developer's site | User's location |
| Control | Highly controlled | Less controlled |
| Purpose | Detect major issues before release | Gather user feedback and detect real-world issues |
| Duration | Short | Longer |

**2. Test Independence**

**What is Test Independence?**

Test independence means that software testing is performed by a team that is separate from the development team to ensure unbiased testing.

**Benefits of Test Independence**

1. **Unbiased Testing** – Independent testers are more likely to identify issues that developers might overlook.
2. **Objective Evaluation** – Testers provide neutral feedback without being influenced by the development team.
3. **Verification of Assumptions** – Testers verify the product’s functionality based on user requirements.
4. **Improved Quality** – A separate testing team ensures thorough and high-quality testing.

**Example:** A company hires an external testing agency to evaluate its e-commerce website to ensure that it functions correctly before launching.

**Drawbacks of Test Independence**

1. **Lack of Communication** – Independent testers may have less interaction with developers, leading to misunderstandings.
2. **Delay in Feedback** – A separate testing team might slow down issue resolution due to the reporting process.
3. **Responsibility Issues** – Developers might rely too much on testers, assuming they will find all issues.
4. **Seen as a Bottleneck** – Testing might be considered a delaying factor in the software development process.

**When to Use Test Independence?**

* Large-scale projects with high complexity.
* Projects requiring compliance with industry standards.
* Software that must be tested from a user’s perspective.

**Conclusion**

Test independence has both benefits and drawbacks. The choice to use it depends on the size and complexity of the project. A balance between independent testing and developer testing ensures high-quality software.

This study material provides a beginner-friendly explanation of Alpha Testing, Beta Testing, and Test Independence with examples and a comparison table.

**Study Material: Basic Understanding of Software Testing**

**1. Difference Between Test Techniques and Testing Tools**

**What is a Test Technique?**

A test technique is a method used to identify test conditions, test cases, and test data. It helps in designing effective test cases.

**Types of Test Techniques**

Test techniques are broadly categorized into three types:

| **Test Technique** | **Description** | **Examples** |
| --- | --- | --- |
| **Black Box Testing** | Focuses on input and output without looking at the internal code. | Equivalence Partitioning, Boundary Value Analysis, Decision Table Testing, State Transition Testing |
| **White Box Testing** | Tests internal structures of the software. | Statement Coverage, Decision Coverage, Path Coverage |
| **Experience-Based Testing** | Relies on the tester’s intuition and experience. | Error Guessing, Exploratory Testing |

Test techniques are used in test analysis and test design phases to create structured test cases.

**What is a Testing Tool?**

A testing tool is software used to support testing activities like test planning, execution, and defect management.

**Types of Testing Tools and Examples**

| **Type of Testing Tool** | **Description** | **Examples** |
| --- | --- | --- |
| **Test Management Tools** | Helps manage and plan test cases. | Google Sheets, Trello, Jira |
| **Test Automation Tools** | Automates test execution. | Selenium WebDriver, Cypress, Robot Framework |
| **Performance Testing Tools** | Tests system performance. | JMeter, HP LoadRunner |
| **API Testing Tools** | Tests APIs for functionality and security. | Postman, SoapUI, RestAssured |

Testing tools support various testing activities throughout the software development lifecycle.

**2. What is Random (Monkey) Testing?**

**Definition:** Monkey testing (or random testing) is a type of software testing where random inputs are used to test the system.

**Example:**

* Testing a login page by entering random usernames and passwords to see if the system crashes.
* Using a tool to generate random inputs to check system stability.

**Key Points:**

* Random inputs are generated without following specific test techniques.
* Less reliable but useful for finding unexpected bugs.
* Often used by beginners to check system robustness.

**3. Difference Between Positive and Negative Testing**

| **Type of Testing** | **Description** | **Example** |
| --- | --- | --- |
| **Positive Testing** | Uses valid inputs to verify expected behavior. | Logging in with a correct username and password. |
| **Negative Testing** | Uses invalid inputs to ensure the system handles errors properly. | Trying to log in with an incorrect password and checking for error messages. |

Both types of testing are crucial in software testing to ensure system stability and security.

**Conclusion**

Understanding the difference between test techniques and testing tools, as well as the importance of monkey testing and positive/negative testing, is essential for effective software testing. Using the right techniques and tools can help testers identify defects efficiently and improve software quality.

**Study Material on Software Testing (Basic Level)**

**1. Decision Table Testing**

**Definition:**

Decision Table Testing is a **black box testing technique** used to test systems where requirements are defined in terms of **rules** or **cause-effect combinations**. It helps in identifying all possible combinations of inputs and their respective outputs.

**When to Use:**

* When the system behavior is driven by multiple conditions.
* When different combinations of inputs lead to different system responses.
* When testing business rules or complex decision-making logic.

**Example:**

Imagine an online shopping platform offering discounts based on customer type and purchase amount.

| **Conditions** | **Preferred Customer** | **Order > $1000** | **Used Charge Card** | **Discount Applied** |
| --- | --- | --- | --- | --- |
| Case 1 | No | No | No | No |
| Case 2 | Yes | No | No | 5% |
| Case 3 | Yes | Yes | No | 10% |
| Case 4 | Yes | Yes | Yes | 15% |

Here, the **conditions** are the inputs, and the **discount applied** is the output. The table helps testers verify all scenarios systematically.

**2. Waterfall Model**

**Definition:**

The Waterfall Model is the **earliest software development lifecycle (SDLC) model**, following a **linear and sequential flow**. Each phase starts only after the previous phase is completed.

**Phases of the Waterfall Model:**

1. **Requirement Analysis** – Gathering project requirements.
2. **System Design** – Designing system architecture.
3. **Implementation (Coding)** – Writing the code.
4. **Testing** – Checking for bugs.
5. **Deployment** – Releasing the software.
6. **Maintenance** – Fixing issues post-release.

**How Testing is Performed:**

Testing happens **only after development is completed**, making it difficult to catch bugs early.

**Diagram:**

Requirements → Design → Development → Testing → Deployment → Maintenance

**Disadvantages:**

* Late-stage testing increases the cost of fixing defects.
* No flexibility to accommodate changes.

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

**Definition:**

The V-Model is an extension of the **Waterfall Model**, emphasizing **early testing** by associating each development phase with a corresponding testing phase.

**How It Works:**

* Development and testing happen simultaneously.
* For every development phase, there is a related **verification** and **validation** phase.

**Phases of V-Model:**

| **Development Phase** | **Testing Phase** |
| --- | --- |
| Requirement Analysis | Acceptance Testing |
| System Design | System Testing |
| Architecture Design | Integration Testing |
| Module Design | Unit Testing |

**Diagram:**

Requirements Analysis → System Design → Architecture Design → Module Design → Coding

| | | |

Acceptance Testing ← System Testing ← Integration Testing ← Unit Testing

**Advantages:**

* **Early defect detection** reduces costs.
* Improves software quality through parallel testing.

**Summary:**

* **Decision Table Testing**: Best for complex rules-based scenarios.
* **Waterfall Model**: Sequential but lacks flexibility.
* **V-Model**: Early testing ensures better quality.

This study material simplifies software testing concepts for beginners using examples, tables, and diagrams. Let me know if you need any modifications!

**Study Material: Basic Concepts of Software Testing**

**Best Practices for Writing Test Cases**

**Definition:**

A test case is a document that describes the steps, inputs, and expected outcomes to verify a particular functionality of a software application. It helps ensure the system works as expected.

**Best Practices:**

1. **Write from an End-User Perspective:**
   * Think like the user while writing test cases.
   * Example: If testing an e-commerce website, consider user actions like searching for a product, adding it to the cart, and checking out.
2. **Write Simple and Clear Steps:**
   * Use easy-to-follow instructions.
   * Example:
     1. Open the website.
     2. Click on the login button.
     3. Enter username and password.
     4. Click "Submit."
3. **Ensure Reusability of Test Cases:**
   * Avoid hardcoded values; use generic terms.
   * Example:
     1. Instead of "Password: 123456," write "Enter a valid password."
4. **Assign Priority Levels:**
   * High priority for core functionalities (e.g., login, payment processing).
   * Low priority for minor UI changes.
5. **Include Key Components in Each Test Case:**
   * Test Case ID
   * Test Case Title
   * Description
   * Pre-conditions
   * Test Steps
   * Test Data
   * Expected Result
   * Actual Result
   * Post-conditions
   * Status (Pass/Fail)
6. **Cover Both Valid and Invalid Scenarios:**
   * Example:
     1. Valid Case: Login with correct credentials.
     2. Invalid Case: Login with incorrect password.
7. **Use Proper Naming Conventions:**
   * Instead of "Login Test," use "Login with Valid Credentials."
8. **Regular Review and Maintenance:**
   * Update test cases when features change.

**Test Suite**

**Definition:**

A test suite is a collection of test cases that are grouped based on related functionalities. It helps in organizing tests and tracking execution results.

**Example of a Test Suite:**

| **Test Suite Name** | **Number of Test Cases** | **Passed** | **Failed** | **Blocked** |
| --- | --- | --- | --- | --- |
| Login Tests | 10 | 8 | 1 | 1 |
| Checkout Tests | 15 | 12 | 2 | 1 |

**Key Points:**

* A test case can belong to multiple test suites.
* Test suites help in better reporting and management.
* Test suites should be structured to cover functional, regression, and integration testing.

**Test Environment**

**Definition:**

A test environment is a combination of hardware, software, and network settings where testers execute test cases.

**Example of a Test Environment Configuration:**

| **Component** | **Example Configuration** |
| --- | --- |
| Web Server | Apache, IIS |
| Database | MySQL, PostgreSQL |
| Operating System | Windows 10, Linux |
| Browser | Chrome, Firefox |
| Network | Wi-Fi, 4G Mobile Data |
| Test Tools | Selenium, JMeter |

**Importance:**

* The test environment should closely match the real-world usage environment.
* Example: A banking app should be tested on secure networks and multiple devices to ensure security and functionality.
* Ensure all dependencies, such as APIs and third-party services, are properly configured in the test environment.

**Conclusion**

Understanding best practices for writing test cases, test suites, and test environments ensures high-quality software testing. A well-structured test case and properly configured environment help in identifying and fixing issues effectively before the software is released to users. Regular updates and maintenance of test cases improve test coverage and accuracy.

**Study Material: Software Testing Basics**

**1. Difference Between Build and Release**

**What is a Build?**

A **build** is an executable file or a compiled version of the software that is given to the testing team for verification.

* It is created by developers and sent to testers.
* The build undergoes multiple iterations of fixing and testing.
* Expected to have defects that need to be identified and fixed.
* Example: A new version of an Android app (.apk) provided to testers before launching it in the Play Store.

**What is a Release?**

A **release** is a final version of the software that is provided to end users after thorough testing.

* It is stable and approved for public use.
* May still have minor defects, but they are not critical.
* Example: A new version of WhatsApp officially released on the Play Store.

**Difference Between Build and Release (Table)**

| **Feature** | **Build** | **Release** |
| --- | --- | --- |
| Purpose | Given to testers for finding defects | Given to end users after approval |
| Stability | May contain defects | Stable for use |
| Users | Testing team | End users |
| Example | A test version of an app | An official app update |

**2. What is Test Data?**

Test data refers to the information used by testers to execute test cases and verify software behavior.

**Example of Test Data**

1. **Login Testing:**
   * Test Data: Username "testuser", Password "12345"
   * Purpose: To check if the login function works correctly.
2. **Credit Card Payment Testing:**
   * Test Data: Dummy Credit Card "4111-1111-1111-1111"
   * Purpose: To verify if the payment gateway processes transactions properly.

**Types of Test Data**

* **Valid Data:** Correct input that should pass the test.
* **Invalid Data:** Incorrect input to check error handling.
* **Boundary Data:** Values at the limits (e.g., entering maximum allowed characters in a field).

Test data can be prepared **manually** or **automatically** using tools for generating large amounts of data.

**3. Difference Between Quality Control (QC) and Quality Assurance (QA)**

**What is Quality Assurance (QA)?**

Quality Assurance focuses on **preventing defects** by improving the development process.

* Ensures that proper processes and methodologies are followed.
* Example: Writing proper test plans and using coding standards.

**What is Quality Control (QC)?**

Quality Control focuses on **finding defects** in the software by testing the application.

* Involves actual execution of test cases.
* Example: Running test cases to identify bugs in an application.

**Difference Between QA and QC (Table)**

| **Feature** | **Quality Assurance (QA)** | **Quality Control (QC)** |
| --- | --- | --- |
| Focus | Process Improvement | Finding Defects |
| Approach | Preventive | Corrective |
| Performed By | Developers, Testers | Testers |
| Example | Reviewing code quality | Running test cases |

**Diagram Representation**

Quality Management

|

|-- Quality Assurance (QA) - Process-Based (Prevention)

| |

| |-- Test Planning

| |-- Requirement Analysis

|

|-- Quality Control (QC) - Execution-Based (Detection)

|

|-- Test Execution

|-- Defect Reporting

**Summary**

* **Build** is for testers, **release** is for end users.
* **Test data** is used to run test cases and verify software.
* **QA prevents defects**, **QC finds defects**.

This study material provides a beginner-friendly overview of these key concepts in software testing.