📘 SOFTWARE TESTING

**🔹 Common Software Problems**

**❓What are the problems you encounter as software users?**

Below are common problems that users often face in faulty software:

**1. Incorrect Calculation**

* **Definition**: Software gives wrong numerical results.
* **Example**: A billing system calculates a 10% tax as $9 instead of $10.
* **Importance**: This could lead to financial losses or legal issues.

**2. Incorrect Data Edits & Ineffective Data Edits**

* **Definition**: Editing operations don’t apply correctly or don’t reflect the changes.
* **Example**: A user edits their address, but the system shows the old address later.
* **Importance**: Data integrity and user trust are compromised.

**3. Incorrect Matching and Merging of Data**

* **Definition**: When combining data sets, matching logic fails.
* **Example**: A CRM merges two different customers due to name similarity.
* **Importance**: Leads to incorrect personalization or analytics.

**4. Data Searches Yield Incorrect Results**

* **Definition**: Search functionality fails to fetch the correct or relevant data.
* **Example**: Searching “John Smith” shows unrelated users.
* **Importance**: Hurts user experience and decision-making.

**5. Incorrect Processing of Data Relationships**

* **Definition**: Logical relationships in data (e.g., foreign keys) are misused.
* **Example**: An order is shown without a corresponding user profile.
* **Importance**: Can crash systems or misinform reports.

**6. Incorrect Coding/Processing of Business Rules**

* **Definition**: Software logic doesn’t follow defined business policies.
* **Example**: Applying a discount on non-discountable items.
* **Importance**: Violates compliance and user expectations.

**7. Inadequate Software Performance**

* **Definition**: The application is slow or crashes under load.
* **Example**: E-commerce site crashes during a flash sale.
* **Importance**: Loss of revenue and reputation.

**8. Confusing or Misleading Data**

* **Definition**: Information is displayed in a way that misguides users.
* **Example**: Showing “0 items” in cart when an item is present.
* **Importance**: User errors and dissatisfaction.

**9. Inconsistent Processing**

* **Definition**: Same input gives different results at different times.
* **Example**: A report shows different totals for the same day.
* **Importance**: Reduces trust in the system.

**10. Unreliable Results or Performance**

* **Definition**: Software behaves unpredictably.
* **Example**: A button sometimes works and sometimes doesn’t.
* **Importance**: Prevents automation and consistent workflows.

**11. Incorrect or Inadequate Interfaces with Other Systems**

* **Definition**: Integrations (APIs, databases) don't function correctly.
* **Example**: Payment gateway fails to confirm transactions.
* **Importance**: Business operations and third-party dependencies fail.

**12. Incorrect File Handling**

* **Definition**: File read/write operations are flawed.
* **Example**: Uploading a profile picture corrupts the file.
* **Importance**: Leads to data loss or corruption.

**🔹 Who is Responsible for Testing?**

* **Everyone involved in development is responsible**, but **dedicated testers (QA engineers)** ensure:
  + Bugs are found early.
  + Software behaves as expected.
* **Roles involved**:
  + **Developers**: Perform unit tests.
  + **Testers/QA**: Perform system, integration, and acceptance testing.
  + **Users/Clients**: Perform acceptance and usability testing.

**🔹 Objectives of a Software Tester**

**Key Goals:**

1. **Find bugs as early as possible and ensure they get fixed**
   * Early bugs are cheaper and easier to fix.
2. **Understand the application well**
   * Domain knowledge helps testers predict where bugs might occur.
3. **Study functionality to find likely bug zones**
   * Focus on complex features or newly added ones.
4. **Study the code to test every line (in white-box testing)**
   * Coverage is essential for quality assurance.
5. **Create test cases to uncover hidden bugs**
   * Use boundary value and edge cases.
6. **Ensure software is usable and reliable**
   * Focus on real-world usage and performance.

**🔹 Objectives of Testing**

**Core Principles:**

* **Execute with the intent of finding errors**
* **Check if system meets the requirements**
* **Verify if the system is “fit for purpose”**
* **Ensure expected behavior under expected conditions**

**Characteristics of a Good Test Case:**

1. **Probability of finding an undiscovered error**
2. **Not redundant** – avoids repeating the same logic
3. **Balanced complexity** – not overly simple or complex

**🔹 Verification vs Validation**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Verification** | **Validation** |
| **Question** | Are we building the system right? | Are we building the right system? |
| **Purpose** | Ensure that code implements the design correctly | Ensure that software meets the user requirements |
| **Activities** | Reviews, walkthroughs, inspections | Testing, user acceptance |
| **Timing** | Before implementation | After implementation |
| **Tools** | Checklists, traceability matrices | Test cases, test plans |

* **Cycle**: They go hand-in-hand — after verification, we validate. If bugs are found, we repeat.

**🔹 What is Software Testing? [IEEE]**

“A process of analyzing a software item to detect the differences between existing and required conditions (that is defects / errors / bugs) and to evaluate the features of the software item.”

* It ensures correctness, completeness, security, and quality of software.

**🔹 Basic Definitions**

**✅ Test**

* **Definition**: The act of exercising software with test cases to either:
  + **Find failures**, or
  + **Demonstrate correct execution**

**✅ Test Case**

* **Definition**: A documented set of conditions or variables used to determine whether a system works correctly.
* **Contents**:
  + Author, Date, Purpose
  + Test Case ID
  + Preconditions
  + Inputs
  + Expected Outputs
  + Observed Outputs

**🔹 Software Testing Lifecycle Phases**

**1. Requirements Study**

* Understand client needs thoroughly to test what matters.
* Critical because:
  + Requirements define “what” to test.
  + Incomplete understanding leads to irrelevant or missed tests.

**2. Analysis and Planning**

* Set clear **objectives** and **test coverage**.
* Define:
  + Testing **schedule** and milestones.
  + **Resources**: Tools, staff, training.
  + **Standards and roles**.

**3. Test Case Design and Development**

* Identify components/modules.
* Design specifications and review test plans.
* Goal: **Maximum coverage with minimum tests.**

**4. Test Execution**

* Perform actual testing:
  + Code review.
  + Execute tests.
  + Simulate performance.
  + Evaluate against expected outcomes.

**5. Test Closure**

* Prepare final **summary reports**.
* Conduct **debrief** sessions.
* Archive test **documentation** for future audits.

**6. Test Process Analysis**

* Analyze test reports and:
  + Improve performance.
  + Plan for future enhancements.

**🔹 Strategy for Testing Conventional Software**

From **abstract to concrete**:

1. **System Engineering**
2. **Requirements**
3. **Design**
4. **Code**

From **narrow to broader** scope:

1. **Unit Testing**
2. **Integration Testing**
3. **Validation Testing**
4. **System Testing**

**🔹 Example: Testing a Ballpoint Pen**

**Analogous to software testing**. We can test:

* **Does the pen write in the right color and line thickness?**
* **Is the logo as per company standards?**
* **Is it safe to chew?**
* **Click mechanism durable after 100,000 clicks?**
* **Will it work after a car drives over it?**

Software, like pens, must work in **intended and unexpected scenarios**.

**🔹 Testing is Not Just for Software**

Similar in bridges, cars, etc.:

* **Verify Requirements** – Are goals defined?
* **Verify Design** – Is the solution correctly architected?
* **Verify Construction** – Is it built properly?

**Software is harder** because:

* It's **intangible**.
* Behavior can change with tiny code errors.

**🔹 Testing Approaches (Levels)**

**1. Unit Testing**

* Test **individual functions/classes/modules**
* Catches **local bugs**
* Doesn’t detect interaction problems.

**2. Integration Testing**

* Test how different units interact.
* Based on module interfaces.
* Doesn’t test full system behavior.

**3. System Testing**

* Test the **entire system as one unit**.
* Focus: **End-to-end behavior**.
* Hard to simulate rare or exceptional conditions.

**4. Acceptance Testing**

* Performed by **customer or end user**.
* Verifies if product **meets expectations** and is ready for release.

**🔹 Basic Testing Approaches**

**✅ Specification-Based (Functional / Black Box Testing)**

* Based on **requirements**, not internal code.
* Only checks:
  + **Inputs**
  + **Expected Outputs**

**✅ Structural-Based (White Box Testing)**

* Based on **internal structure and logic**.
* Requires access to **source code**.
* Validates:
  + Every **path**
  + Every **branch**
  + Every **loop**

**🔹 Summary of Testing Types (Strategies)**

|  |  |  |
| --- | --- | --- |
| **Testing Strategy** | **Basis** | **Focus Area** |
| **Black-Box** | Specifications, Requirements | Input/output behavior |
| **White-Box** | Internal Code Paths | Logic, statements, branches |